Lab2(Embedded C lesson 3)

Gdb debugger commands

First we learn how to use debugger without any IDE then apply what we learn on eclipse to work with any code

Debugger without IDE

```
MINGW32:/h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson3/1-Continue_lesson2
edded_C/Lesson3/1-Continue_lesson2 (main)
$ arm-none-eabi-gdb.exe learn-in-depth.elf
GNU gdb (GDB) 7.5.1
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-mingw32 --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from H:\Embedded_Systems_Online_Diploma\Unit_3_Embedded_C\Lesson
3\1-Continue_lesson2\learn-in-depth.elf...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:4
4                       ldr sp, =stack_top
(gdb) l
1       @make the reset visible from other files
2       .global reset
3       reset:
4                       ldr sp, =stack_top
5                       bl main
6       stop: b stop
(gdb) b main
Breakpoint 1 at 0x100e0: file app.c, line 9.
(gdb) c
Continuing.

Breakpoint 1, main () at app.c:9
9               Uart_Send_String(string_buffer);
(gdb) b Uart_Send_String
Breakpoint 2 at 0x10100: file uart.c, line 7.
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000100e0 in main at app.c:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00010100 in Uart_Send_String at uart.c:7
(gdb) b uart.c:9
Breakpoint 3 at 0x10104: file uart.c, line 9.
(gdb) where
#0  main () at app.c:9
(gdb) c
Continuing.

Breakpoint 2, Uart_Send_String (
    P_tx_string=0x10140 <string_buffer> "Learn-in-depth:<AliTaima>")
    at uart.c:7
7               while(*P_tx_string != '\0')
(gdb) c
Continuing.

Breakpoint 3, Uart_Send_String (
    P_tx_string=0x10140 <string_buffer> "Learn-in-depth:<AliTaima>")
```

```
MINGW32:/h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson3/1-Continue_lesson2
C:\Program Files (x86)\qemu\qemu-system-arm.exe: could not load kernel 'learn-in-depth.elf

Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Le
-Continue_lesson2 (main)
$ mingw32-make clear_all
rm *.o *.elf *.bin

Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Le
-Continue_lesson2 (main)
$ mingw32-make
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s app.c -o app.o
arm-none-eabi-gcc.exe -c -I . -g -mcpu=arm926ej-s uart.c -o uart.o
arm-none-eabi-as.exe -g -mcpu=arm926ej-s startup.s  -o startup.o
arm-none-eabi-ld.exe -T linker_script.ld  app.o uart.o  startup.o -o learn-in-depth.elf
arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin

Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Le
-Continue_lesson2 (main)
$ qemu-system-arm.exe -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
Learn-in-depth:<AliTaima>
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Le
-Continue_lesson2 (main)
$ qemu-system-arm.exe -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
Learn-
```

```
Breakpoint 3, Uart_Send_String (
    P_tx_string=0x10144 <string_buffer+4> "n-in-depth:<AliTaima>") at uart.c:9
9               UART0DR = (unsigned int)(*P_tx_string);
(gdb) s
10              P_tx_string++; //next char ...
(gdb) s
7               while(*P_tx_string != '\0')
(gdb) s

Breakpoint 3, Uart_Send_String (
    P_tx_string=0x10145 <string_buffer+5> "-in-depth:<AliTaima>") at uart.c:9
9               UART0DR = (unsigned int)(*P_tx_string);
(gdb) s
10              P_tx_string++; //next char ...
```

Debugger with eclipse

```
Eclipse - Debug - H:\Embedded_Systems_Online_Diploma\Unit_3_Embedded_C\Lesson3\1-Continue_lesson2\app.c - Eclipse
File  Edit  Source  Refactor  Navigate  Search  Project  Run  Window  Help

Debug
lab1_debug [C/C++ Attach to Application]
  Process
    Thread #1 1 (CPU#0 [running]) (Suspended : User Request)
      main() at app.c:9 0x100e0
  C:/ARM_TOOLCHAIN/bin/arm-none-eabi-gdb.exe (7.5.1)

Registers  Modules
Name                 Value   Description
General Registers            General Purpose and FPU Register Group

app.c  startup.s  uart.c  0x10000  startup.s  app.c

1  #include "uart.h"
2
3  unsigned char string_buffer[100] = "Learn-in-depth:<AliTaima>";
4  unsigned char const string_buffer2[100] = "to create a rodate section";
5  unsigned char const string_buffer3[100] = "to create a rodate section";
6  void main(void)
7  {
8      // Send on physical board
9      Uart_Send_String(string_buffer);
10 }

Outline  Disassembly  Enter location here
000100d8:   push    {r11, lr}
000100dc:   add     r11, sp, #4
000100e0:   ldr     r0, [pc, #4]    ; 0x100ec <main+20>
000100e4:   bl      0x100f0 <Uart_Send_String>
000100e8:   pop     {r11, pc}
000100ec:   andeq   r0, r1, r0, asr #2
Uart_Send_String:
000100f0:   push    {r11}           ; (str r11, [sp, #-4]!)
000100f4:   add     r11, sp, #0
000100f8:   sub     sp, sp, #12
000100fc:   str     r0, [r11, #-8]
00010100:   b       0x10120 <Uart_Send_String+48>
00010104:   ldr     r3, [pc, #48]   ; 0x1013c <Uart_Send_String+76>

Console  Tasks  Problems  Executables  Debugger Console  Memory
lab1_debug [C/C++ Attach to Application] C:\ARM_TOOLCHAIN\bin\arm-none-eabi-gdb.exe (7.5.1)
file learn-in-depth.elf
A program is being debugged already.
Are you sure you want to change the file? (y or n) [answered Y; input not from terms
Reading symbols from H:\Embedded_Systems_Online_Diploma\Unit_2_C_Programming\Lesson3
si
si
```

```
MINGW32:/h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson3/1-Continue_lesson2
sson3/1-Continue_lesson2 (main)
$ qemu-system-arm.exe -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
```

Makefile

I walk step by step to build my Makefile that automate everything and make it general as possible

Makefile

```
#@copyright: Ali Taima
CC=arm-none-eabi-
CFLAGS=-g -mcpu=arm926ej-s
INCS=-I .
LIBS=
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o) #same as OBJS = $(patsubst$.c, $.o, $SRC)
As = $(wildcard *.s)
AsOBJ = $(As:.s=.o)
Project_name=learn-in-depth
all: $(Project_name).bin
%.o: %.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) -o $@

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

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

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

Lab2

In lab2 we will continue on lab1 in lesson1&2, to make complex startup file with assemply and c

Startup.s

Makefile | startp.s

```
/* Startp_cortexM3.s
Eng.Ali Taima
*/
/*SRAM 0x20000000*/
.section .vectors

.word 0x20001000          /*stack top address */
.word _reset              /*1 Reset */
.word Vector_handler      /*2 NMI */
.word Vector_handler      /*3 hard Fault */
.word Vector_handler      /*4 NM Fault */
.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

_reset:
  bl main
  b .
.thumb_func
Vector_handler:
  b _reset
```

- In this file assume stack top in address within range(0x20001000)
- Make the reset section
- Make the vector_handler as general to reset everytime we call it
- Splict the section into memories
- Enable thumb instructions

Here as expected our memory sections additional to debug sections

```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Dipl
edded_C/Lesson3/2-Continue_lesson1 (main)
$  arm-none-eabi-objdump.exe startp.o -h

startp.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000008  00000000  00000000  00000034  2**1
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  0000003c  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  0000003c  2**0
                  ALLOC
  3 .vectors      00000050  00000000  00000000  0000003c  2**0
                  CONTENTS, RELOC, READONLY
  4 .ARM.attributes 00000021  00000000  00000000  0000008c  2**0
                  CONTENTS, READONLY
  5 .debug_line   0000003a  00000000  00000000  000000ad  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  6 .debug_info   00000081  00000000  00000000  000000e7  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_abbrev 00000014  00000000  00000000  00000168  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_aranges 00000020  00000000  00000000  00000180  2**3
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

- Also we can read the sympols in the .o file
    - as we see there are the sympols that we made in the startup

```
$ arm-none-eabi-nm.exe startup.o
00000000 t _reset
         U main
00000006 t vector_handler
```

Linker_scirpts.ld

```
Makefile        linker_script.ld
/* Linker script CortexM3
Eng.Ali Taima
*/
MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128K
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20K
}

SECTIONS
{
    .text : {
            *(.vectors*)
            *(.text*)
            *(.rodata)
    } > flash
    .data : {
            *(.data)
    } > flash

    .bss : {
            *(.bss)
    }> sram

}
```

Given in this lab, the origin and length for flash and sram memory so I assume them in MEMORY section

- I split the SECTIONS into (.text, .data, .bss)
    - In .text, I order them by putting .vectors, .text then .rodata and store them in the flash
    - In .data, I collect all .data sections and store them in flash also(here we should move the data from ROM to RAM but we ignore it in this time)
    - In .bss, I collect all .bss sections and store them in sram

Makefile

We will make the same Makefile that we make at the beginning of the lecture with different name of the project_name

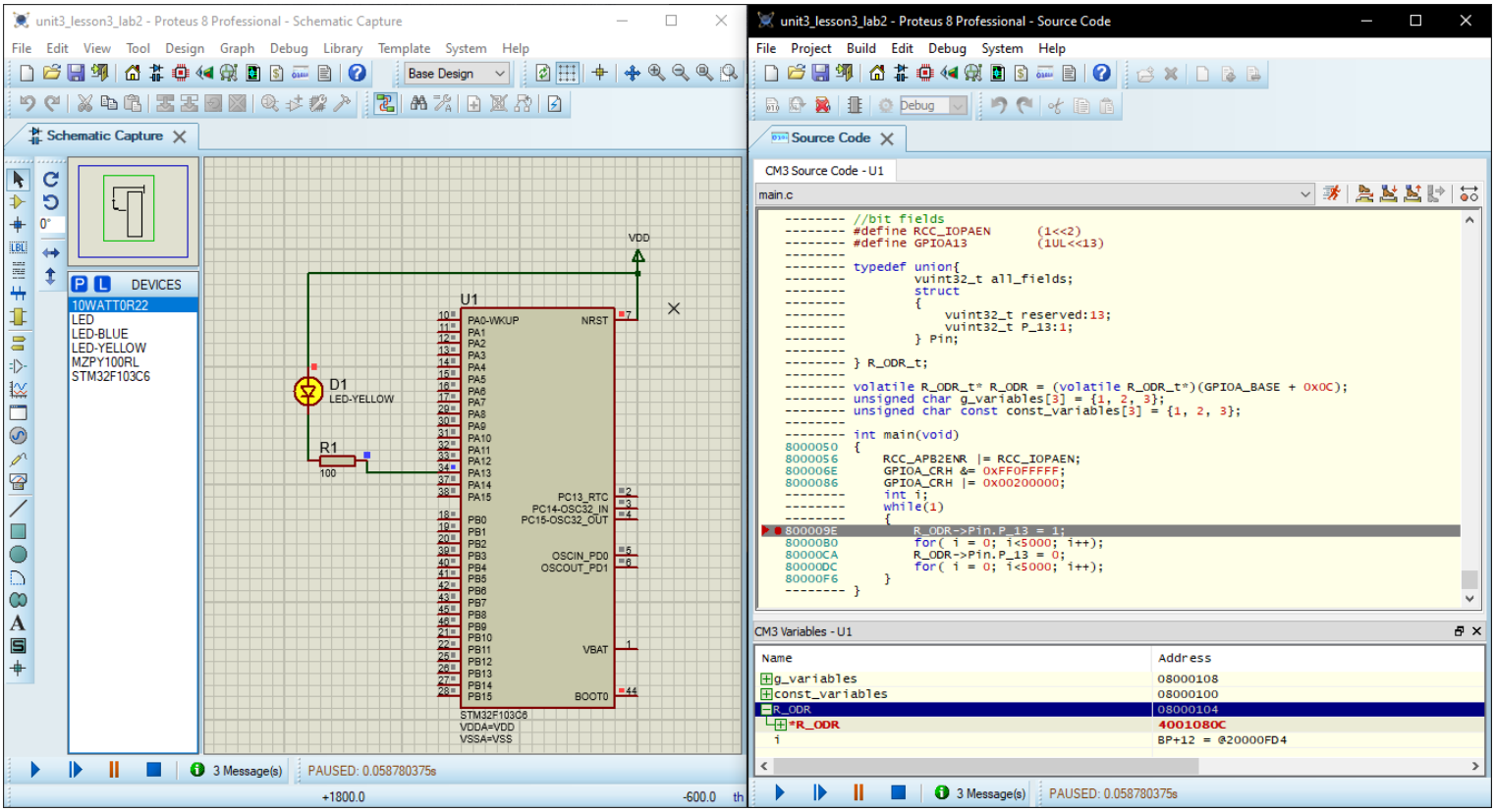
```makefile
#@copyright: Ali Taima
CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2
INCS=-I .
LIBS=
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o) #same as OBJS = $(patsubst$.c, $.o, $SRC)
As = $(wildcard *.s)
AsOBJ = $(As:.s=.o)
Project_name=learn_in_depth_cortex_m3

all: $(Project_name).hex
    @echo "----------Build is Done-----------"
%.o: %.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) -o $@ -Map=Map_file.map

$(Project_name).hex: $(Project_name).elf
    $(CC)objcopy.exe -O binary $< $@

clear_all:
    rm *.o *.elf *.hex

clear:
    rm *.elf *.hex

```

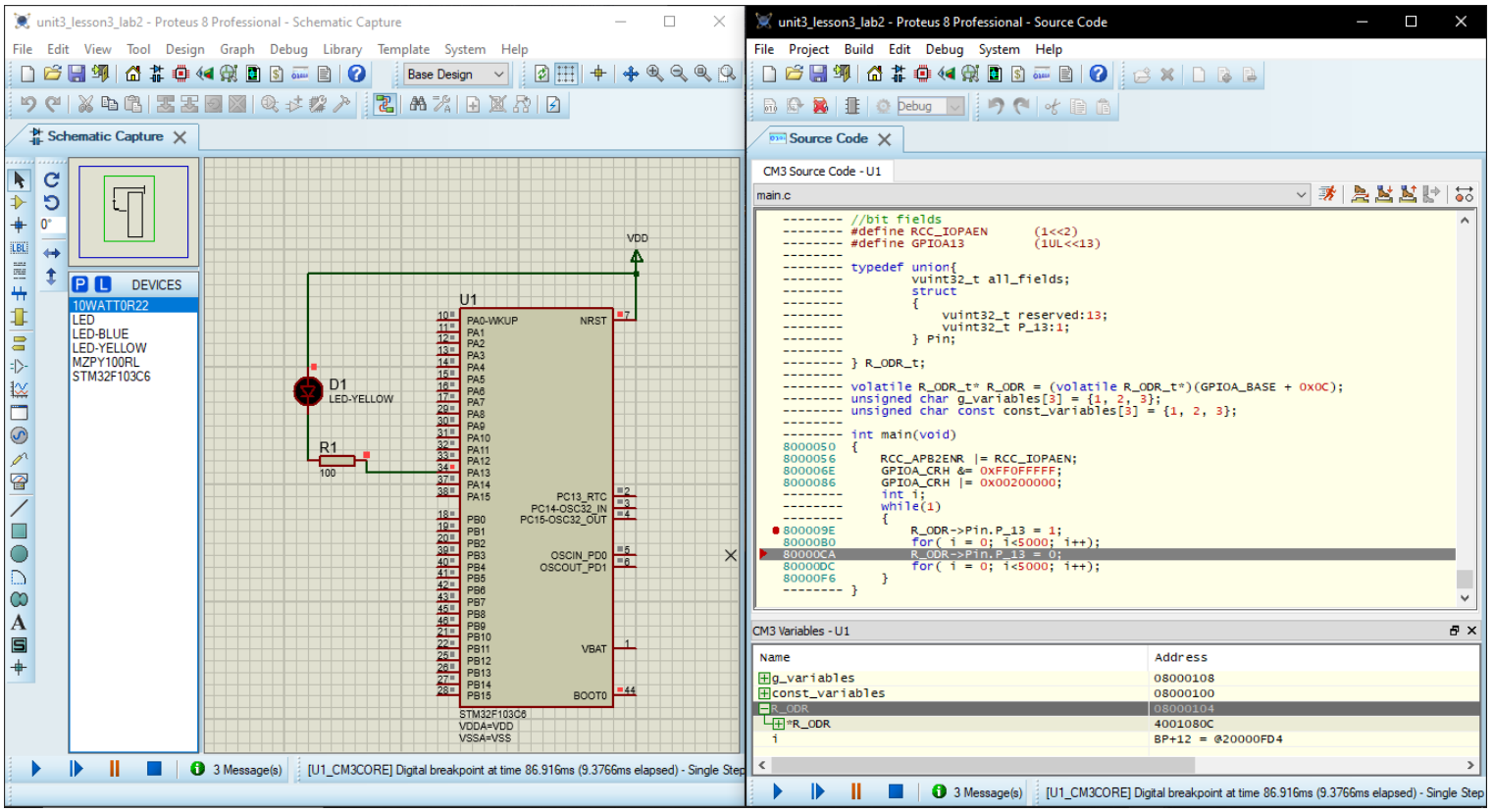
Also we can open our Mapfile to make sure from our memory sections

```
Memory Configuration

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

Linker script and memory map


.text           0x08000000      0x110
 *(.vectors*)
 .vectors       0x08000000       0x1c startp.o
                0x08000000                vectors
 *(.text*)
 .text          0x0800001c       0x48 startp.o
                0x0800001c                Reset_Handler
                0x08000028                NMI_Handler
                0x08000034                H_fault_Handler
                0x08000040                NM_Fault_Handler
                0x0800004c                Bus_Fault_Handler
                0x08000058                Usage_Fault_Handler
 .text          0x08000064       0xa8 main.o
                0x08000064                main
 *(.rodata)
 .rodata        0x0800010c        0x4 main.o
                0x0800010c                const_variables

```

Lab works on proteus

Finally I can see the project work on proteus and I can debuge it also 😊





Startup.c

```c
/*startup.c for CortexM3
Eng.Ali Taima*/
#include <stdint.h>
#define STACK_Start_SP 0x20001000
// I tell the compiler that it define in another file
extern int main(void);
void Reset_Handler(void)
{
    main();
}
void NMI_Handler(void)
{
    Reset_Handler();
}
void H_fault_Handler(void)
{
    Reset_Handler();
}
void NM_Fault_Handler(void)
{
    Reset_Handler();
}
void Bus_Fault_Handler(void)
{
    Reset_Handler();
}
void Usage_Fault_Handler(void)
{
    Reset_Handler();
}
uint32_t vectors[] __attribute__((section(".vectors"))) ={
    STACK_Start_SP,
    (uint32_t) &Reset_Handler,
    (uint32_t) &NMI_Handler,
    (uint32_t) &H_fault_Handler,
    (uint32_t) &NM_Fault_Handler,
    (uint32_t) &Bus_Fault_Handler,
    (uint32_t) &Usage_Fault_Handler,
};
```

I can write the startup with C because the CortextM family set the sp set by default by the processor so we can continue after it to run our startup as .c code

```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson3/2-Continue_lesson1 (main)
$ arm-none-eabi-nm.exe learn_in_depth_cortex_m3.elf
0800004c T Bus_Fault_Handler
0800010c T const_variables
08000114 D g_variables
08000034 T H_fault_Handler
08000064 T main
08000040 T NM_Fault_Handler
08000028 T NMI_Handler
08000110 D R_ODR
0800001c T Reset_Handler
08000058 T Usage_Fault_Handler
08000000 T vectors
```

But there is a problem → that vector table consume our memory so I will solve this problem by using weak and alias concepts

```c
/*startup.c for CortexM3
Eng.Ali Taima*/
#include <stdint.h>
#define STACK_Start_SP 0x20001000
// I tell the compiler that it define in another file
extern int main(void);
void Reset_Handler(void)
{
    main();
}
void Default_Handler()
{
    Reset_Handler;
}
void NMI_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
void H_fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
void NM_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")));;

uint32_t vectors[] __attribute__((section(".vectors"))) ={
    STACK_Start_SP,
    (uint32_t) &Reset_Handler,
    (uint32_t) &NMI_Handler,
    (uint32_t) &H_fault_Handler,
    (uint32_t) &NM_Fault_Handler,
    (uint32_t) &Bus_Fault_Handler,
    (uint32_t) &Usage_Fault_Handler,
};
```

```
Ali Mohamed Taima@AliTaima MINGW32 /h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson3/2-Continue_lesson1 (main)
$ arm-none-eabi-nm.exe learn_in_depth_cortex_m3.elf
08000028 W Bus_Fault_Handler
080000dc T const_variables
08000028 T Default_Handler
080000e4 D g_variables
08000028 W H_fault_Handler
08000034 T main
08000028 W NM_Fault_Handler
08000028 W NMI_Handler
080000e0 D R_ODR
0800001c T Reset_Handler
08000028 W Usage_Fault_Handler
08000000 T vectors
```

Now we can see how alias save our memory

If I want to write my Handler (I check it by make an empty definition)

linker_script.ld | Makefile | startup.c | main.c | Map_file.map

```c

typedef volatile unsigned int vuint32_t;
#include <stdint.h>
//register address
#define RCC_BASE        0x40021000
#define GPIOA_BASE      0x40010800
#define RCC_APB2ENR     *(volatile uint32_t *)(RCC_BASE   + 0x18)
#define GPIOA_CRH       *(volatile uint32_t *)(GPIOA_BASE + 0x04)
#define GPIOA_ODR       *(volatile uint32_t *)(GPIOA_BASE + 0x0C)
//bit fields
#define RCC_IOPAEN      (1<<2)
#define GPIOA13         (1UL<<13)


extern void NMI_Handler(void)
{

}
extern void Bus_Fault_Handler(void)
{

}
```

We can see that my handlers take different addresses when I make an definition for them

```
Ali Mohamed Taima@AliTaima MINGW32
/h/Embedded_Systems_Online_Diploma/Unit_3_Emb
edded_C/Lesson3/2-Continue_lesson1 (main)
$ arm-none-eabi-nm.exe
learn_in_depth_cortex_m3.elf
08000040 T Bus_Fault_Handler
080000f4 T const_variables
08000028 T Default_Handler
080000fc D g_variables
08000028 W H_fault_Handler
0800004c T main
08000028 W NM_Fault_Handler
08000034 T NMI_Handler
080000f8 D R_ODR
0800001c T Reset_Handler
08000028 W Usage_Fault_Handler
08000000 T vectors
```

How to copy data(data create .bss sections)

Now we want to copy data from ROM to RAM and initialize the .bss section

Linker script

linker_script.ld | Makefile | startup.c | main.c

```
/* Linker script CortexM3
Eng.Ali Taima
*/
MEMORY
{
    flash(RX) : ORIGIN = 0x08000000, LENGTH = 128K
    sram(RWX) : ORIGIN = 0x20000000, LENGTH = 20K
}

SECTIONS
{
    .text : {
            *(.vectors*)
            *(.text*)
            *(.rodata)
            _E_text = .;
    } > flash
    .data : {
            _S_DATA = .;
            *(.data)
            _E_DATA = .;

    } > sram AT> flash

    .bss : {
            _S_bss = .;
            *(.bss)
            . = ALIGN(4);
            _E_bss = .;

            . = ALIGN(4);
            . = . + 0x1000;
            _stack_top = .;

    }> sram

}
```

I align data and set the stack top

Move data

Startup.c

linker_script.ld | Makefile | startup.c | main.c

```c
/*startup.c for CortexM3
Eng.Ali Taima*/
#include <stdint.h>
#define STACK_Start_SP 0x20001000
// I tell the compiler that it define in another file
extern unsigned int _stack_top;
extern unsigned int _S_DATA;
extern unsigned int _E_DATA;
extern unsigned int _S_bss;
extern unsigned int _E_bss;
extern unsigned int _E_text;
extern int main(void);
void Reset_Handler(void);
void Default_Handler()
{
    Reset_Handler;
}
void NMI_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
void H_fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler")));;
void NM_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")));;

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) &NM_Fault_Handler,
    (uint32_t) &Bus_Fault_Handler,
    (uint32_t) &Usage_Fault_Handler,
};


void Reset_Handler(void)
{
    //copy data from ROM to RAM
    unsigned int DATA_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
    unsigned char* P_src = (unsigned char*)&_E_text;
    unsigned char* P_dst = (unsigned char*)&_S_DATA;
    int i;
    for(i = 0; i<DATA_size; i++)
    {
        *((unsigned char*)P_dst++) = *((unsigned char*)P_src++);
    }
    // init the .bss with zero
    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
    P_dst = (unsigned char*)&_S_bss;
    for(i = 0; i<bss_size ;i++)
    {
        *((unsigned char*)P_dst++) = (unsigned char)0;
    }
    // jump to main
    main();
}
```

Now we can see the effect on our memory in

```
Ali Mohamed Taima@AliTaima MINGW32
/h/Embedded_Systems_Online_Diploma/Unit_3_Embedded_C/Lesson3/2-Continue_lesson1 (main)
$  arm-none-eabi-objdump.exe learn_in_depth_cortex_m3.elf -h

learn_in_depth_cortex_m3.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000019c  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  0800019c  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001000  20000008  080001a4  00010008  2**0
                  ALLOC
  3 .debug_info   000002f8  00000000  00000000  00010008  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 000001bf  00000000  00000000  00010300  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000f4  00000000  00000000  000104bf  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000040  00000000  00000000  000105b3  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   0000014e  00000000  00000000  000105f3  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    000001ac  00000000  00000000  00010741  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000011  00000000  00000000  000108ed  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000108fe  2**0
                  CONTENTS, READONLY
 11 .debug_frame  000000a4  00000000  00000000  00010934  2**2
                  CONTENTS, READONLY, DEBUGGING
```

Proteus run

Proteus run very well 😊