

Lesson 4 - LAB 3 – Report

Makefile:

*This Makefile compatible with CPU: Cortex-M4

```
main.c  Makefile  app.map  startup.c  linker_script.ld
1  #@copyright : Bavly-Mansour
2  CC=arm-none-eabi-
3  CFLAGS= -mcpu=cortex-m4 -gdwarf-2 -g
4  INCS= -I .
5  LIBS=
6  SRC= $(wildcard *.c)
7  OBJ= $(SRC:.c=.o)
8  As = $(wildcard *.s)
9  AsOBJ = $(As:.s=.o)
10 Project_name=unit3_lab4_cortexM4
11
12 all: $(Project_name).bin
13     @echo "=====
14     @echo "=====DONE=====
15     @echo "=====
16
17 %.o: %.c
18     $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
19     @echo "=====Compiler gcc DONE=====
20
21
22 %.o: %.s
23     $(CC)as.exe $(CFLAGS) $(INCS) $< -o $@
24     @echo "=====STARTUP DONE=====
25
26 $(Project_name).elf: $(OBJ) $(AsOBJ)
27     $(CC)ld.exe -T linker_script.ld -Map=app.map $(OBJ) $(AsOBJ) -o $@
28     cp $(Project_name).elf $(Project_name).axf
29     @echo "=====linker DONE=====
30
31 $(Project_name).bin: $(Project_name).elf
32     $(CC)objcopy.exe -O binary $(Project_name).elf $(Project_name).bin
33     @echo "=====File.bin DONE=====
34
35 $(Project_name).hex: $(Project_name).elf
36     $(CC)objcopy.exe -O binary $< $@
37     @echo "=====File.hex DONE=====
38
39 clean:
40     rm *.elf *.bin
41     @echo "=====CLEAN=====
42
43 clean_all:
44     rm *.elf *.bin *.o
45     @echo "=====CLEAN ALL=====
```

* -gdwarf-2 instead of -g with Cortex-M3.

* \$@ is for target, \$< for dependency.

* % is a generalization means all of.

Output of Makefile:

```
Bavly@DESKTOP-CNE79H4 MINGW64 ~/Desktop/Lab 3
$ make
arm-none-eabi-gcc.exe -c -mcpu=cortex-m4 -gdwarf-2 -g -I . main.c -o main.o
=====Compiler gcc DONE=====
arm-none-eabi-gcc.exe -c -mcpu=cortex-m4 -gdwarf-2 -g -I . startup.c -o startup.
o
=====Compiler gcc DONE=====
arm-none-eabi-ld.exe -T linker_script.ld -Map=app.map main.o startup.o -o unit3
_lab4_cortexM4.elf
cp unit3_lab4_cortexM4.elf      unit3_lab4_cortexM4.axf
=====linker DONE=====
arm-none-eabi-objcopy.exe -O binary unit3_lab4_cortexM4.elf unit3_lab4_cortexM4.
bin
=====File.bin DONE=====
=====
=====DONE=====
=====

Bavly@DESKTOP-CNE79H4 MINGW64 ~/Desktop/Lab 3
$ |
```

Main.c file:

```
main.c  Makefile  app.map  startup.c  linker_script.ld
1  /**
2  ****
3  * @file      : main.c
4  * @author    : Eng Bavly Mansour
5  * @brief     : Main program body
6  ****
7  */
8
9  //first define registers
10 //*****
11 #define SYSCTL_RCGC2_R      (*((volatile unsigned long*)0x400FE108))
12 #define GPIO_PORTF_DIR_R   (*((volatile unsigned long*)0x40025400))
13 #define GPIO_PORTF_DEN_R   (*((volatile unsigned long*)0x4002551C))
14 #define GPIO_PORTF_DATA_R  (*((volatile unsigned long*)0x400253FC))
15
16 int main ()
17 {
18     volatile unsigned long delay_count ;
19
20     SYSCTL_RCGC2_R  =0x20 ;
21     //set system ctl enable the gpio port SYSCTL register = 0x20
22     //delay to make sure GPIOF is up and running
23     for (delay_count = 20 ; delay_count < 200 ; delay_count ++);
24     GPIO_PORTF_DIR_R |=1<<3 ;
25     //dir is output for pin 3 port f , to set pf3 as an output
26     GPIO_PORTF_DEN_R |=1<<3 ;
27     while (1)
28     {
29         GPIO_PORTF_DATA_R |= 1<<3 ;
30         for (delay_count =0 ; delay_count < 200000 ; delay_count ++);
31         GPIO_PORTF_DATA_R &=~(1<<3);
32         for (delay_count =0 ; delay_count < 200000 ; delay_count ++);
33     }
34     return 0;
35 }
36
37 }
```

- SYSCTL_RCGC2_R = 0x20; // set system ctl enable the gpio port SYSCTL register = 0x20, delay to make sure GPIOF is up and running

Startup.c file for Cortex-M4:

```
main.c  Makefile  app.map  startup.c  linker_script.ld
1  /*
2  *****
3  *****
4  learn-in-depth
5  By:Bavly Mansour
6  *****
7  *****
8  */
9
10 #include <stdint.h>
11 //define STACK_Start_SP 0x20001000 //txt replacement of STACK_Start_SP "stack pointer" to address 0x20001000
12 extern int main (void);           // define external int main (void)
13 void Reset_Handler (void);       //define rest_handler function
14
15
16 /*Copying data from flash to SRAM and allocate space
17 for bss and stack according to the memory bourders */
18
19
20 void Default_Handler ()
21 {
22     Reset_Handler();
23 }
24
25 void NMI_Handler() __attribute__((weak,alias("Default_Handler")));;
26 void H_Fault_Handler() __attribute__((weak,alias("Default_Handler")));;
27
28
29 /*booking 1024 B locate by .bss through uninitialized
30 array of int 256 element (256*4=1024)*/
31
32 static unsigned long Stack_top[256];
33
34 //array of pointer to function take nothing and return void
35 void ( * const g_p_fn_Vectors[])() __attribute__((section(".vectors"))) ={
36     (void (*)()) ((unsigned long)Stack_top + sizeof(Stack_top)),
37     &Reset_Handler ,
38     &NMI_Handler ,
39     &H_Fault_Handler
40 };
41
42
```

```
main.c  Makefile  app.map  startup.c  linker_script.ld
41
42
43 extern unsigned int _E_text;
44 extern unsigned int _S_DATA;
45 extern unsigned int _E_DATA;
46 extern unsigned int _S_bss;
47 extern unsigned int _E_bss;
48 extern unsigned int _stack_top;
49
50
51 void Reset_Handler(void)
52 {
53     // copying data from ROM to RAM
54     unsigned int DATA_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;
55     unsigned char* P_src = (unsigned char*)&_E_text ;
56     unsigned char* P_dst = (unsigned char*)&_S_DATA ;
57
58     for (int i =0 ; i < DATA_size ; i++){
59         *((unsigned char *)P_dst++) = *((unsigned char*)P_src++) ;
60     }
61
62
63     // bss with zero
64     unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss ;
65     P_dst = (unsigned char*)&_S_bss ;
66     for (int i =0 ; i < bss_size ; i++ ){
67         *((unsigned char*)P_dst++) = *((unsigned char*)P_src++) ;
68     }
69     //go to main()
70     main();
71 }
72
73
74
```

- Alias attribute causes the declaration to be emitted as an alias for another symbol.
- Weak attribute causes the declaration to be emitted as a weak symbol rather than a global.
- booking 1024 B locate by .bss through uninitialized array of int 256 element (256*4=1024)

• Functions in MapFile :

```
.text          0x00000000      0x12b
*(.vectors*)
.vectors       0x00000000      0x10 startup.o
               0x00000000      g_p_fn_Vectors
*(.text*)
.text          0x00000010      0x80 main.o
               0x00000010      main
.text          0x00000090      0x98 startup.o
               0x00000090      H_Fault_Handler
               0x00000090      Default_Handler
               0x00000090      NMI_Handler
               0x0000009c      Reset_Handler
*(.rodata)
.rodata        0x00000128      0x3 main.o
               0x00000128      const_var
               0x0000012b      _E_text = .
```

- Main at 0x00000010
- g_p_fn_Vectors at 0x00000000 instead of _stack_top symbol.
- And all Handler functions at the same address of the default address due to the alias. Default_Handler at 0x00000090, while the Reset_handler has a different address at 0x0000009c.

Linker script for Cortex-M4:

```
1  /*This is linker script for LAB3 CPU=CORTEX-M4 ,by : ENG_BAVLY*/
2
3  MEMORY
4  {
5      flash(RX) : ORIGIN = 0x00000000, LENGTH = 512M
6      sram(RWX) : ORIGIN = 0x20000000, LENGTH = 512M
7      /*flash memory origin and length , flash: read and execute*/
8      /*sRAM origin and length , sram : read write execute*/
9
10 }
11 /*sections*/
12 SECTIONS {
13     .text :{
14         *(.vectors*)
15         *(.text*)
16         *(.rodata)
17         _E_text = .;          /*end of text section in flash @ 0x08000b3*/
18     }> flash
19
20     /*everything inside .text put it in flash */
21
22     .data :{
23         _S_DATA = .;          /*start of data section @ 0x08000b3 the same address of end text section */
24         *(.data)
25         _E_DATA = .;          /*end of data section */
26     }>sram AT> flash          /*virtual memory :sram */
27
28     .bss :{
29         _S_bss = . ;          /*start of bss section */
30         *(.bss*)
31         _E_bss = . ;
32     }>sram
33
34     /* bss are the uninitialized variables = zeros so ignore them from the flash */
35
36 }
37
38
39
```

- According to memory borders:
 - _E_text is the end of .text section
 - _S_DATA is the start of .data section
 - _S_bss is the start of .bss section
- .text section includes .rodata section.

- Memory configuration according to linker script memory borders:

```

1
2 Allocating common symbols
3 Common symbol      size      file
4
5 bss_var            0x3        main.o
6
7 Memory Configuration
8
9 Name              Origin              Length              Attributes
10 flash             0x00000000      0x20000000         xr
11 sram               0x20000000      0x20000000         xrw
12 *default*         0x00000000      0xffffffff
13
14 Linker script and memory map
15
16
17 .text              0x00000000      0x12b
18 *(.vectors*)
19 .vectors           0x00000000      0x10 startup.o
20                   0x00000000      g_p_fn_Vectors
21 *(.text*)
22 .text              0x00000010      0x80 main.o
23                   0x00000010      main
24 .text              0x00000090      0x98 startup.o
25                   0x00000090      H_Fault_Handler
26                   0x00000090      Default_Handler
27                   0x00000090      NMI_Handler
28                   0x0000009c      Reset_Handler
29 *(.rodata)
30 .rodata            0x00000128      0x3 main.o
31                   0x00000128      const_var
32                   0x0000012b      _E_text = .
33
34 .glue_7            0x0000012c      0x0
35 .glue_7            0x0000012c      0x0 linker stubs
36
37 .glue_7t           0x0000012c      0x0
38 .glue_7t           0x0000012c      0x0 linker stubs
39
40 .vfp11_veneer      0x0000012c      0x0
41 .vfp11_veneer      0x0000012c      0x0 linker stubs
42
43 .v4_bx             0x0000012c      0x0
44 .v4_bx             0x0000012c      0x0 linker stubs
45
46 .iplt              0x0000012c      0x0
47 .iplt              0x0000012c      0x0 main.o
48
49 .rel.dyn           0x0000012c      0x0
50 .rel.iplt          0x0000012c      0x0 main.o

```


• Debugging in Keil uvision output:

The screenshot displays the Keil uVision IDE interface. The top menu bar includes File, Edit, View, Project, Flash, Debug, Peripherals, Tools, SVCS, Window, and Help. The toolbar contains various icons for file operations, editing, and debugging.

The **Registers** window on the left shows the state of the processor registers. The **Disassembly** window in the center displays the assembly code, with the following instructions highlighted:

```
52: {  
53: // copying data from ROM to RAM  
0x0000009C B580 PUSH {r7,lr}  
0x0000009E B086 SUB sp,sp,#0x18  
0x000000A0 AF00 ADD r7,sp,#0x00  
54: unsigned int DATA_size = (unsigned char*)&_E_DATA - (unsigned char*)&_S_DATA;  
0x000000A2 4A1C LDR r2,[pc,#112] ; @0x00000114  
0x000000A4 4B1C LDR r3,[pc,#112] ; @0x00000118  
0x000000A6 1AD3 SUBS r3,r2,r3  
0x000000A8 607B STR r3,[r7,#0x04]  
55: unsigned char* P_src = (unsigned char*)&_E_text ;  
0x000000AA 4B1C LDR r3,[pc,#112] ; @0x0000011C  
0x000000AC 617B STR r3,[r7,#0x14]  
56: unsigned char* P_dst = (unsigned char*)&_S_DATA ;  
57:  
0x000000AE 4B1A LDR r3,lob.#104] ; @0x00000118
```

The **main.c** source code window shows the following code:

```
35  
36 /* Loop forever */  
37 int i;  
38 RCC_APB2ENR |= (1<<2);  
39 GPIOA_CHR &= 0xfffff;  
40 GPIOA_CHR |= 0x00200000;  
41 while (1) {  
42  
43 GPIOA_ODR |= 1<<13;  
44 // set 1 to a bit |= 1<<n  
45 for (i = 0 ; i < 5000 ; i++)  
46 GPIOA_ODR &= (1<<13);  
47 // clear a bit &= ~(1<<n)  
48 for (i=0; i< 5000 ; i++);  
49  
50 }  
51 return 0;
```

The **startup.c** source code window shows the following code:

```
1 /*  
2 *****  
3 *****  
4 learn-in-depth  
5 By:Bavly Mansour  
6 *****  
7 *****  
8 */  
9  
10 #include <stdint.h>  
11 //define STACK_Start_SP 0x20001000 //txt replacement of STACK_Start_<br/>12 extern int main (void); // define external int main (void)  
13 void Reset_Handler (void); //define reset_handler function  
14  
15  
16 //Copying data from flash to SRAM and allocate space  
17 for bss and stack according to the memory bouders */
```

The **Command** window at the bottom shows the following text:

```
Running with Code Size Limit: 32K  
Load "C:\\Users\\Bavly\\Desktop\\Lab 3\\unit3_lab4_cortexM4.axf"
```

The **Memory** window shows the address field.

The status bar at the bottom indicates the simulation is running, with a time of 93.77100006 sec, L52 C:1, and CAP NUM SCRL OVR: R/W.