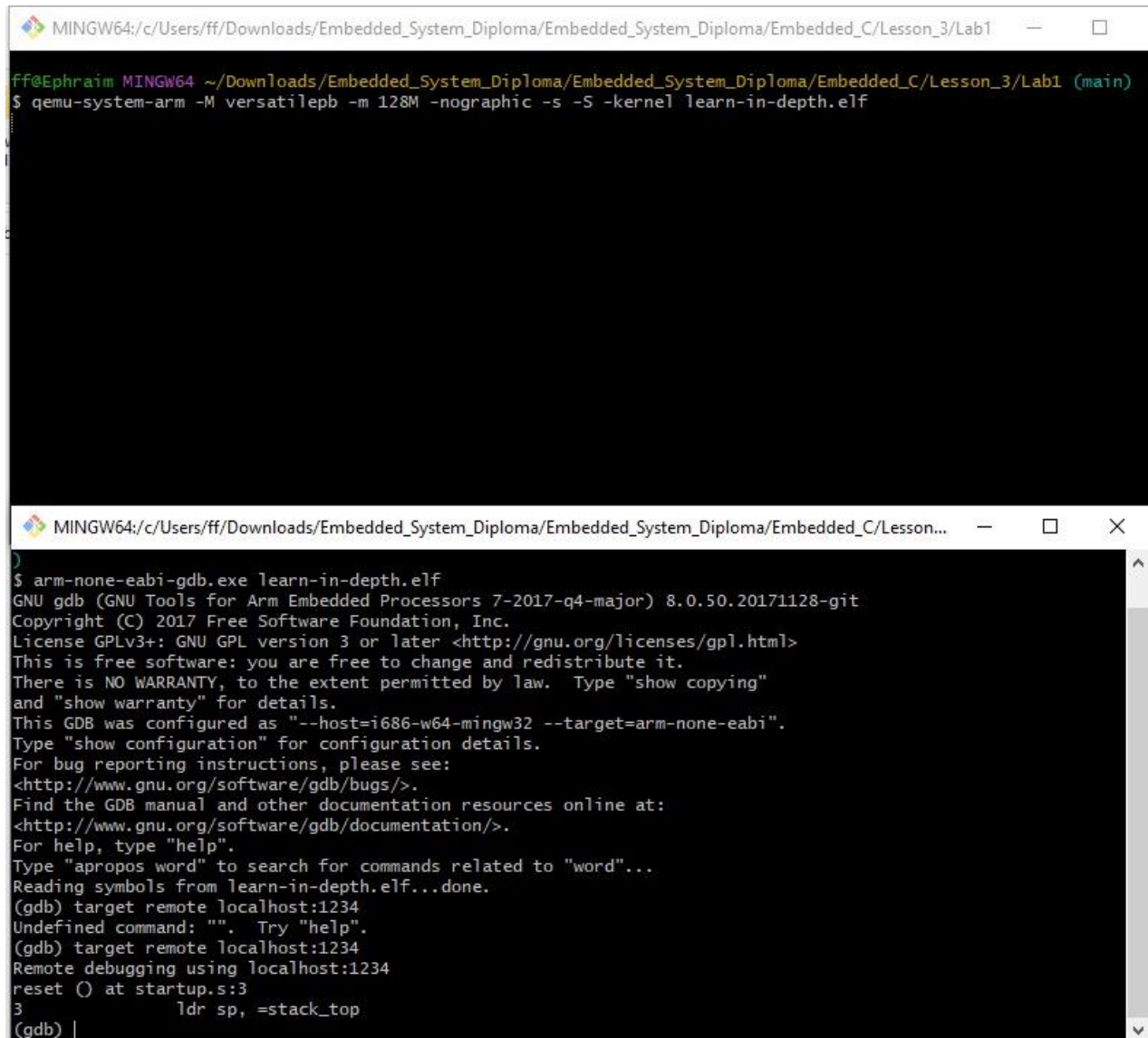


# **LESSON 3 LABS**

**Guirguis Hedia**

## Debugging Steps with Terminal :

- Start Debugging



```
MINGW64:/c/Users/ff/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab1
ff@Ephraim MINGW64: ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab1 (main)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf

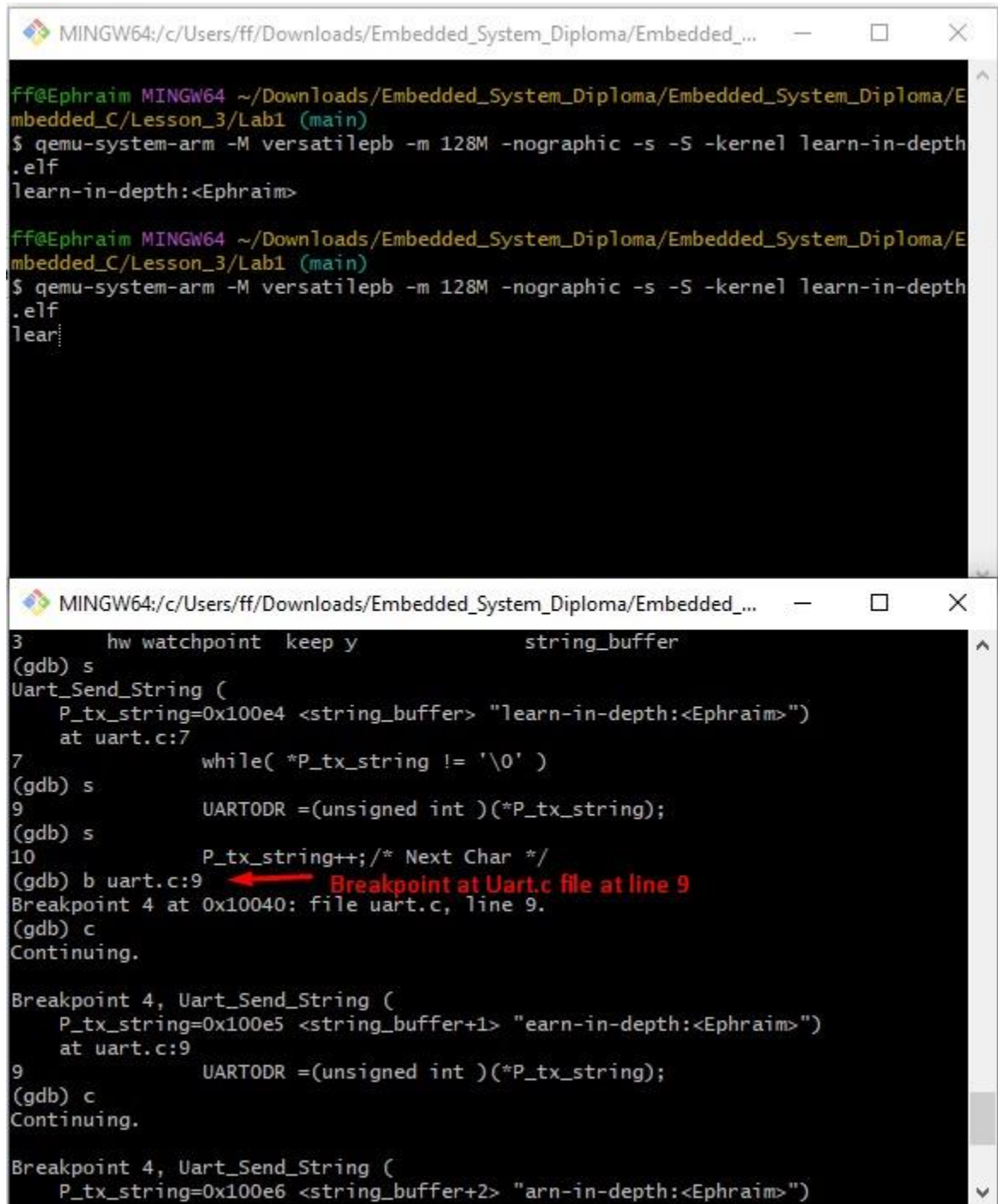
$ arm-none-eabi-gdb.exe learn-in-depth.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change 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-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from learn-in-depth.elf...done.
(gdb) target remote localhost:1234
Undefined command: ". Try "help".
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:3
3          ldr sp, =stack_top
(gdb) |
```

- Commands Used to Debug

```
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab1 (main)
$ arm-none-eabi-gdb.exe learn-in-depth.elf
GNU gdb (GNU Tools for Arm Embedded Processors 7-2017-q4-major) 8.0.50.20171128-
git
Copyright (C) 2017 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-w64-mingw32 --target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from learn-in-depth.elf...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:3
3       ldr sp, =stack_top
(gdb) l
1       .globl reset
2       reset:
3           ldr sp, =stack_top
4           bl main
5       stop: b stop
(gdb) b main
Breakpoint 1 at 0x10018: file app.c, line 7.
(gdb) b *0x10010
Breakpoint 2 at 0x10010: file app.c, line 6.
(gdb) s
reset () at startup.s:4
4       bl main
(gdb) s
Breakpoint 2, main () at app.c:6
6       {
(gdb) s
Breakpoint 1, main () at app.c:7
7       Uart_Send_String (string_buffer);
(gdb) print string_buffer[0]
$1 = 108 'l'
(gdb) watch string_buffer
Hardware watchpoint 3: string_buffer
(gdb) l
```

```
(gdb) watch string_buffer
Hardware watchpoint 3: string_buffer
(gdb) info breakpoints
Num      Type             Disp Enb Address      What
1        breakpoint      keep y   0x00010018 in main at app.c:7
          breakpoint already hit 1 time
2        breakpoint      keep y   0x00010010 in main at app.c:6
          breakpoint already hit 1 time
3        hw watchpoint   keep y                   string_buffer
```

- The Output During Debug



The image shows two screenshots of a MINGW64 terminal window. The top screenshot shows the execution of the program `learn-in-depth.elf` using `qemu-system-arm`. The bottom screenshot shows the same program being debugged with GDB. A breakpoint is set at `uart.c:9`, and the program is running. The GDB output shows the current state of the program, including the value of `P_tx_string` and the value of `UARTODR`.

```

MINGW64:/c/Users/ff/Downloads/Embedded_System_Diploma/Embedded_...
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab1 (main)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth
.elf
learn-in-depth:<Ephraim>

ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab1 (main)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth
.elf
lear:

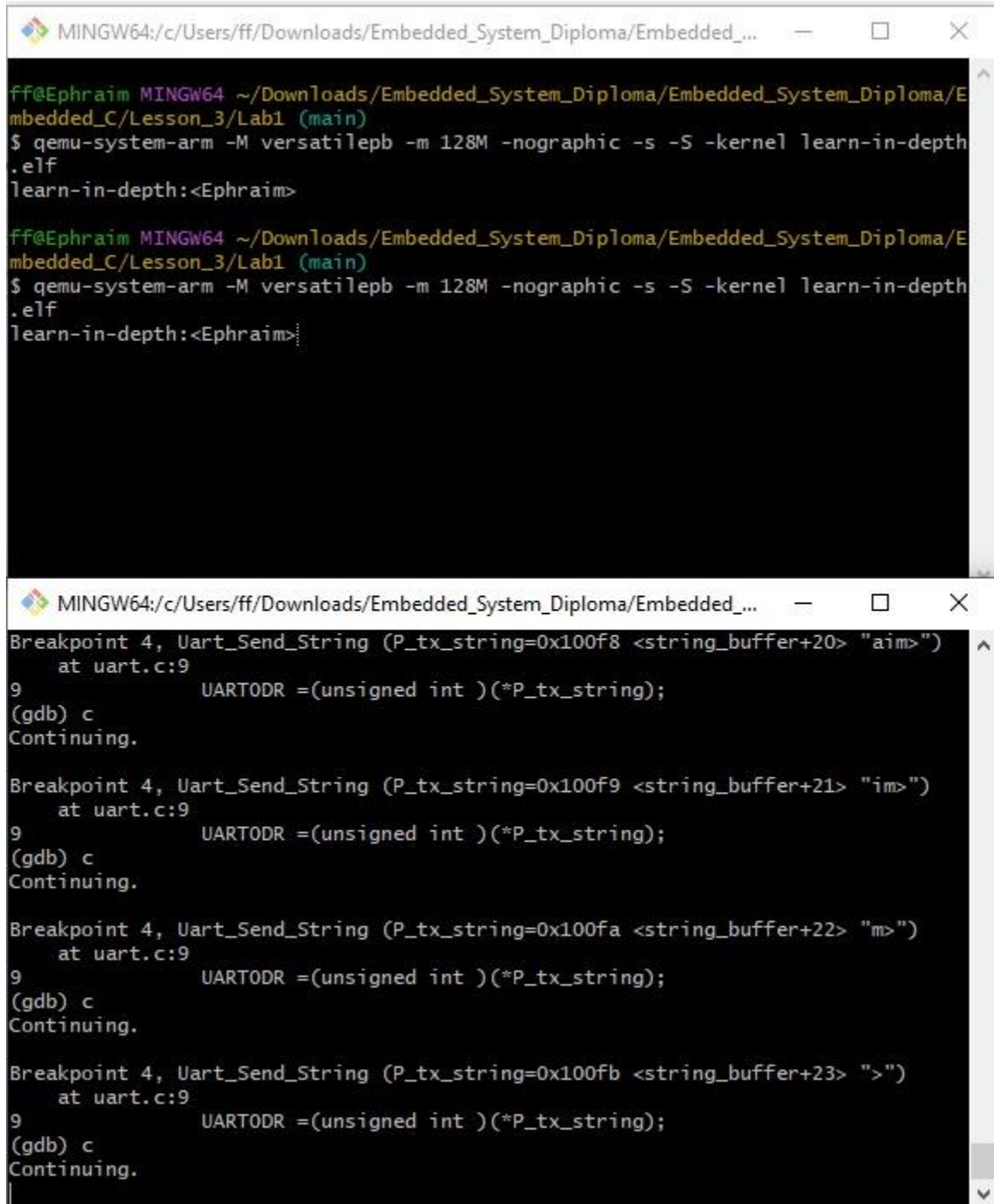
3      hw watchpoint keep y          string_buffer
(gdb) s
Uart_Send_String (
  P_tx_string=0x100e4 <string_buffer> "learn-in-depth:<Ephraim>")
  at uart.c:7
7          while( *P_tx_string != '\0' )
(gdb) s
9          UARTODR =(unsigned int )(*P_tx_string);
(gdb) s
10         P_tx_string++;/* Next Char */
(gdb) b uart.c:9 ← Breakpoint at Uart.c file at line 9
Breakpoint 4 at 0x10040: file uart.c, line 9.
(gdb) c
Continuing.

Breakpoint 4, Uart_Send_String (
  P_tx_string=0x100e5 <string_buffer+1> "earn-in-depth:<Ephraim>")
  at uart.c:9
9          UARTODR =(unsigned int )(*P_tx_string);
(gdb) c
Continuing.

Breakpoint 4, Uart_Send_String (
  P_tx_string=0x100e6 <string_buffer+2> "arn-in-depth:<Ephraim>")

```

- Debug is Finished And The Message is Printed



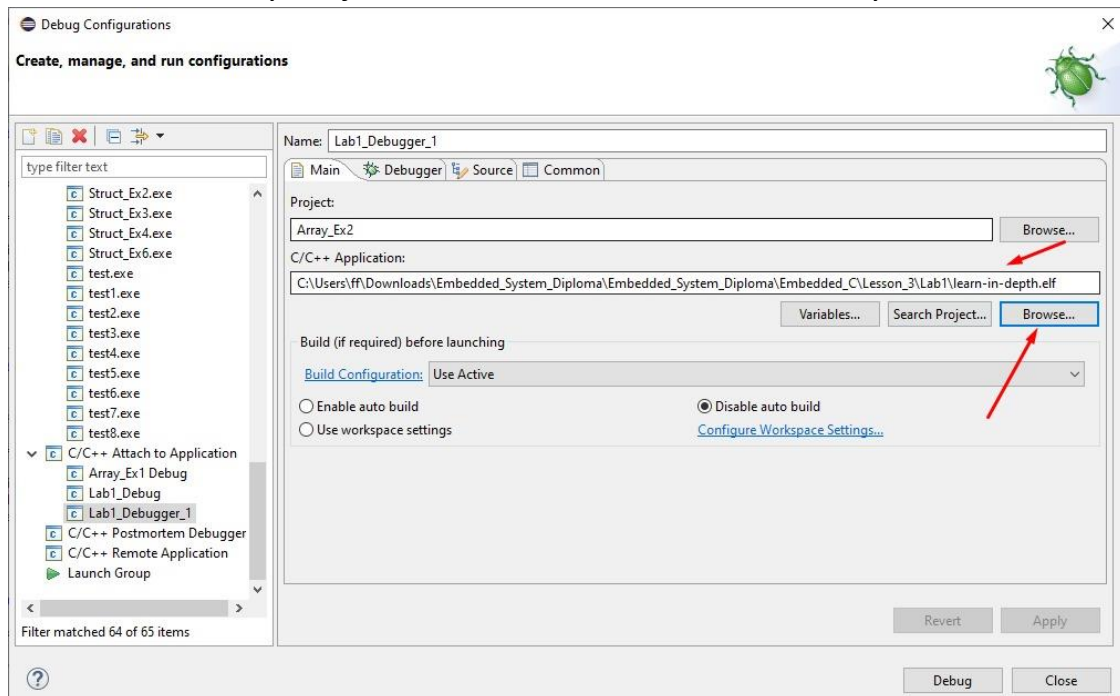
The image shows two screenshots of a terminal window. The top screenshot shows the user running the command `qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf` in a MINGW64 environment. The prompt is `learn-in-depth:<Ephraim>`. The bottom screenshot shows the same terminal window with the program running under GDB. It displays four breakpoints at `uart.c:9` for the `Uart_Send_String` function, with the string buffer contents being `"aim"`, `"im"`, `"m"`, and `">"`. The user enters `c` to continue execution, and the program prints the message `aim>`.

```
MINGW64:/c/Users/ff/Downloads/Embedded_System_Diploma/Embedded_...  
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E  
mbedded_C/Lesson_3/Lab1 (main)  
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth  
.elf  
learn-in-depth:<Ephraim>  
  
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E  
mbedded_C/Lesson_3/Lab1 (main)  
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth  
.elf  
learn-in-depth:<Ephraim>  
  
MINGW64:/c/Users/ff/Downloads/Embedded_System_Diploma/Embedded_...  
Breakpoint 4, Uart_Send_String (P_tx_string=0x100f8 <string_buffer+20> "aim")  
  at uart.c:9  
9          UARTODR =(unsigned int )(*P_tx_string);  
(gdb) c  
Continuing.  
  
Breakpoint 4, Uart_Send_String (P_tx_string=0x100f9 <string_buffer+21> "im")  
  at uart.c:9  
9          UARTODR =(unsigned int )(*P_tx_string);  
(gdb) c  
Continuing.  
  
Breakpoint 4, Uart_Send_String (P_tx_string=0x100fa <string_buffer+22> "m")  
  at uart.c:9  
9          UARTODR =(unsigned int )(*P_tx_string);  
(gdb) c  
Continuing.  
  
Breakpoint 4, Uart_Send_String (P_tx_string=0x100fb <string_buffer+23> ">")  
  at uart.c:9  
9          UARTODR =(unsigned int )(*P_tx_string);  
(gdb) c  
Continuing.  
|
```

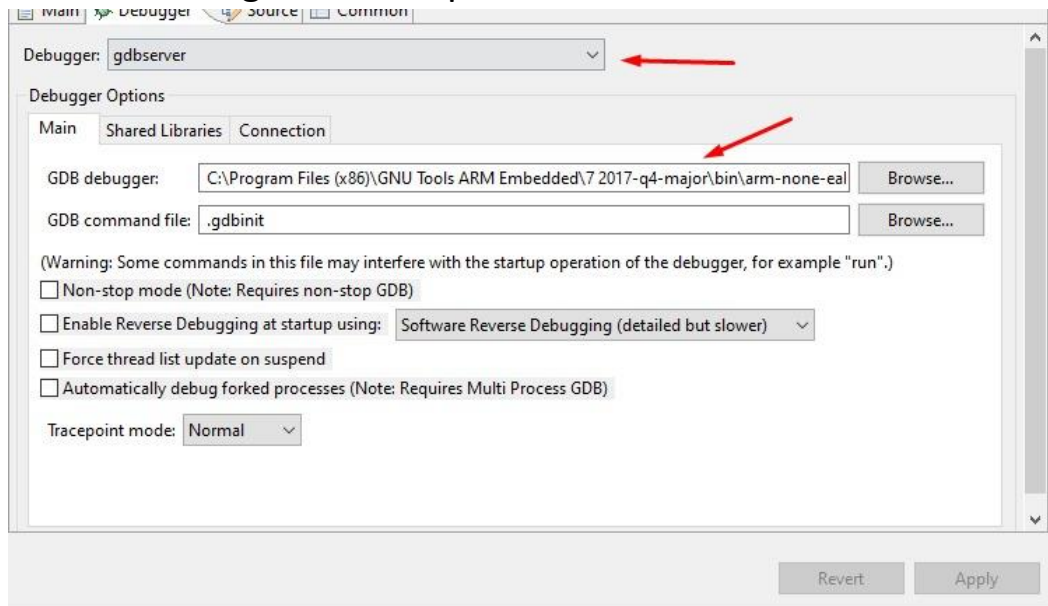
# Debugging Steps with Eclipse :

- Debug Configuration to Start Debugging in Eclipse

## 1. Choose Any Project file and Select The Real elf file you want to Debug

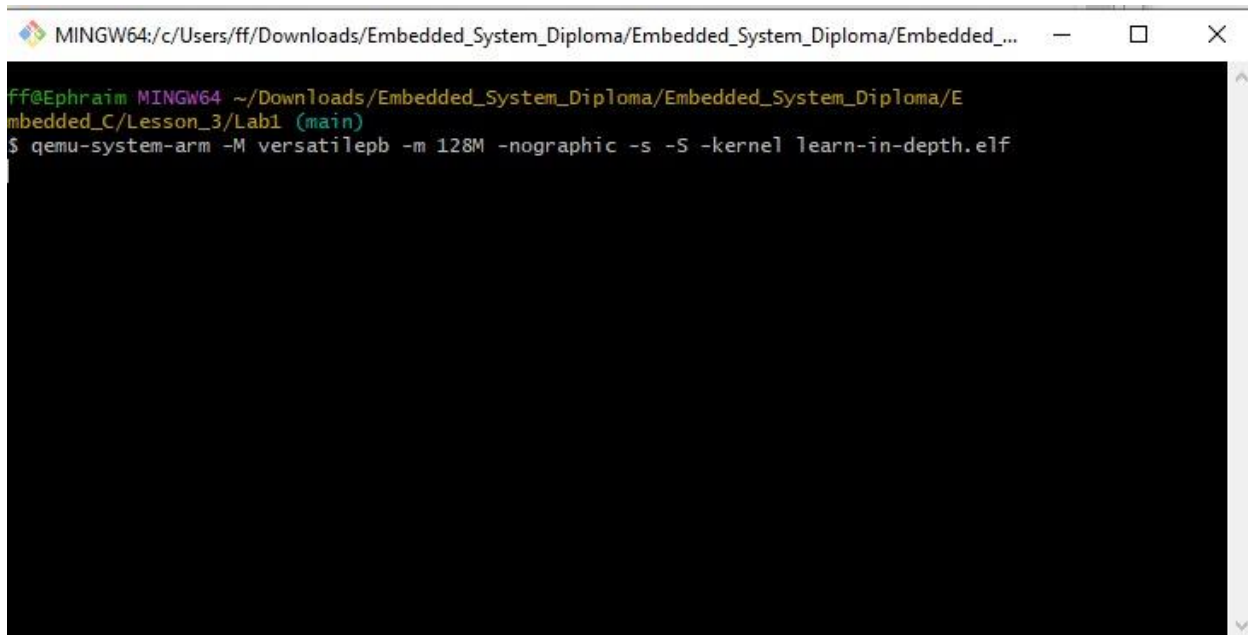


## 2. Choose gdbserver option and the arm toolchain



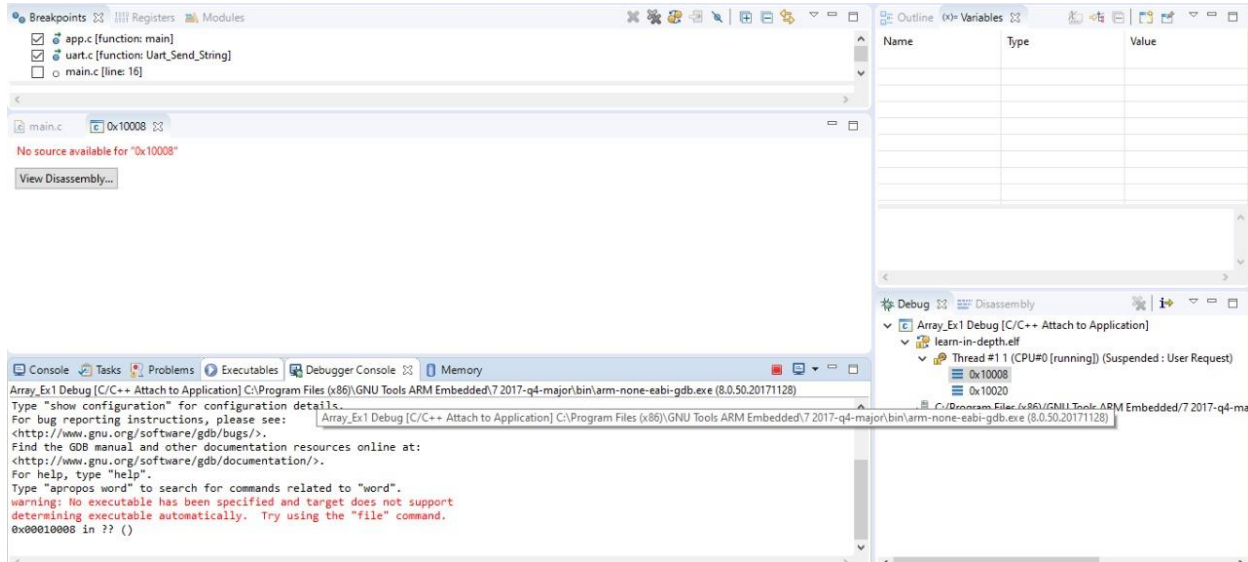


### 3. Write a Command To Open The Board Virtually in Qemu



```
MINGW64:/c/Users/ff/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_...
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E...
mbedded_C/Lesson_3/Lab1 (main)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
```

### 4. Start Debug In Eclipse But you Will Find This Error



To Solve This Error

you should put The elf File in The Project Folder you choose

Then write file Command To Start Debugging

## 5. To See The Project file Direction Path

Write pwd command to see The path of The project you choose  
Then Put the elf File in The folder

```
pwd  
Working directory C:\Users\ff\workspace\C_Programming\Array_Ex3. Then Copy the elf file to This path
```

## 6. Use File Command To Read Symbols and Start debug

```
Working directory C:\Users\ff\workspace\C_Programming\Array_Ex3.  
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 terminal]  
Reading symbols from learn-in-depth.elf...done.
```

Use file Command to Read Symbol

## 7. The Result During Debug

The screenshot displays a debugger interface with the following components:

- Breakpoints:** A list of breakpoints for `app.c (function: main)`, `uart.c (function: Uart_Send_String)`, and `main.c (line: 16)`.
- Code Editor:** Shows the source code for `main.c` and `uart.c`. The code includes UART registers and a function `Uart_Send_String` that sends a string to the UART.
- Console:** Displays the output of the program, showing the string `learn-in-depth:<Ephraim>` being sent to the UART.
- Debugger Console:** Shows the command `file learn-in-depth.elf` and its output, indicating that the file is an ELF executable.
- Terminal:** A terminal window showing the command `gemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf` and its output, which is the same string `learn-in-depth:<Ephraim>`.



## Lesson 3 Lab 2 with Startup.s file :

- Main.c file

```
#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)

typedef union {
    vint32_t    all_fields;
    struct {
        vint32_t    reserved:13 ;
        vint32_t    pin13:1 ;
    }pin;
}R_ODR_t;

volatile R_ODR_t* R_ODR=(volatile R_ODR_t*) (GPIOA_BASE+0x0C);
unsigned char g_variables[3]={1,2,3};
unsigned char const const_variables[3]={1,2,3};
unsigned char volatile bss_variables[3];

int main(void)
{
    RCC_APB2ENR |=RCC_IOPAEN;
    GPIOA_CRH &=0xFF0FFFFFF ;
    GPIOA_CRH |=0x00200000 ;
    while(1)
    {
        //      GPIO_ODR |= (1<<13); //Set Bit 13
        R_ODR->pin.pin13=1;
        for (int i=0 ;i<5000;i++);

        //      GPIO_ODR &=~(1<<13); //Clear Bit 13
        R_ODR->pin.pin13=0;
        for (int i=0 ;i<5000;i++);

    }

    return 0;
}
```

To Increase .data Section Size

To Create .rodata Section

To Create Size For .bss Section

- Startup.s file

```

/*Startup_cortexM3.s
Guirguis hedia*/

/*SRAM 0x20000000*/

.section .vectors /*command to indicate The output will be in .vectors */

.word _stack_top /*stack top address*/ _stack_top Sampol Will Be Calculated in Linker_script file
.word _reset /* 1 Reset */
.word Vector_handler /* 2 NMI */
.word Vector_handler /* 3 Hard Fault*/
.word Vector_handler /* 4 MM 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 /* 18 ... */
/* On to IRQ67 */

.section .text /*command to indicate The output will be in .text*/
_reset:
    bl main
    b .

.thumb_func /*to Accept 16 bit instructions*/
Vector_handler:
    b _reset

```

- Linker\_Script file

```

/*Linker Script CortexM3
Guirguis Hedia
*/
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)
        . = ALIGN(4) ;
        _E_DATA = . ;
    }>flash

    .bss : {
        _S_bss = . ;
        *(.bss*)
        _E_bss = . ;
        . = ALIGN(4) ;
        . = . +0x1000;
        _stack_top = .;
    }> sram
}

```

flash memory will begin at address 0x08000000 with Size 128k  
 SRAM memory will begin at address 0x20000000 with Size 20 k

The first Address here in 0x08000000 store The address which be used by Sp  
 The processor use this address to jump to Stack\_Top in SRAM

This Command To Make The Memory Alignment To Increase The Performance

Stack\_top Will be Located after .bss Section with Size =4K Byte  
 This Will Be Shown in Map\_file.map

- Make file

```
#@Copyright : Guinguis Hedia

CC=arm-none-eabi-
CFLAG=-gdwarf-2 -mcpu=cortex-m3 Processor Name
INCS=-I .
LIBS=
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
As= $(wildcard *.s)
AsOBJ= $(As:.s=.o)

Project_Name=learn_in_depth_cortex_m3

all: $(Project_Name).bin
    @echo "=====Build is Done=====

startup.o: startup.s
    $(CC)as.exe $(CFLAGS) $< -o $@

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

$(Project_Name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $@ -Map=Map_file.map To Generate Map_file.map

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

clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin
```

- Building is Done

Name	Date modified
learn_in_depth_cortex_m3.bin	9/1/2022 7:19 PM
learn_in_depth_cortex_m3.elf	9/1/2022 7:19 PM
linker_script.ld	9/1/2022 4:39 PM
log.txt	8/30/2022 11:28 PM
main.c	9/1/2022 6:08 PM
main.o	9/1/2022 7:19 PM
makefile	9/1/2022 5:42 PM
Map_file.map	9/1/2022 7:19 PM
startup.o	9/1/2022 7:19 PM
startup.s	9/1/2022 4:39 PM

```
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab2 with Startup_dot_s (main)
$ mingw32-make.exe clean_all
rm *.o *.elf *.bin

ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab2 with Startup_dot_s (main)
$ mingw32-make.exe
arm-none-eabi-gcc.exe -c -I . main.c -o main.o
arm-none-eabi-as.exe startup.s -o startup.o
arm-none-eabi-ld.exe -T linker_script.ld main.o startup.o -o learn_in_depth_cortex_m3.elf -Map=Map_file.map
arm-none-eabi-objcopy.exe -O binary learn_in_depth_cortex_m3.elf learn_in_depth_cortex_m3.bin
=====Build is Done=====

ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab2 with Startup_dot_s (main)
$
```

- Map File Details

1	Allocating common symbols		
2	Common symbol	size	file
3			
4	bss_variables	0x3	main.o
5			
6	Memory Configuration		
7			
8	Name	Origin	Length
9	flash	0x08000000	0x00020000
10	sram	0x20000000	0x00005000
11	*default*	0x00000000	0xffffffff
12			Attributes
13			xr
14			xrw
15	Linker script and memory map		
16			
17			
18	.text	0x08000000	0x133
19	*(.vectors*)		
20	.vectors	0x08000000	0x50 startup.o
21	*(.text*)		
22	.text	0x08000050	0xd4 main.o
23			main
24	.text	0x08000124	0xc startup.o
25	*(.rodata)		
26	.rodata	0x08000130	0x3 main.o
27			const_variables
28			_E_text = .
29	.glue_7	0x08000134	0x0
30	.glue_7	0x08000134	0x0 linker stubs
31			
32	.glue_7t	0x08000134	0x0
33	.glue_7t	0x08000134	0x0 linker stubs
34			
35	.vfp11_veneer	0x08000134	0x0
36	.vfp11_veneer	0x08000134	0x0 linker stubs
37			
38	.v4_bx	0x08000134	0x0
39	.v4_bx	0x08000134	0x0 linker stubs
40			
41	.iplt	0x08000134	0x0
42	.iplt	0x08000134	0x0 main.o
43			
44			
45	.data	0x08000134	0x8
46			_S_DATA = .
47	*(.data)		
48	.data	0x08000134	0x7 main.o
49			R_ODR
50			g_variables
51	.data	0x0800013b	0x0 startup.o
52			. = ALIGN (0x4)
53	*fill*	0x0800013b	0x1
54			_E_DATA = .
55			
56	.igot.plt	0x0800013c	0x0
57	.igot.plt	0x0800013c	0x0 main.o
58			
59	.bss	0x20000000	0x1003
60			_S_bss = .
61	*(.bss*)		
62	.bss	0x20000000	0x0 main.o
63	.bss	0x20000000	0x0 startup.o
64			_E_bss = .
65			. = ALIGN (0x4)
66			. = (. + 0x1000)
67	*fill*	0x20001000	0x1000
68			_stack_top = .
69	COMMON	0x20001000	0x3 main.o
70			bss_variables
71	LOAD main.o		
72	LOAD startup.o		
73	OUTPUT(learn_in_depth_cortex_m3.elf elf32-littlearm)		
74			
75	.comment	0x00000000	0x7e
76	.comment	0x00000000	0x7e main.o
77			0x7f (size before relaxing)
78	.ARM.attributes		
79		0x00000000	0x2e

The Begin Address of Flash

SRAM Begin With Address 0x20000000

The First Section In Flash Memory is .vectors and The first Address in The .vector Store The Address of The sp

.data Section Begin with Address 0x08000134 and end with Address 0x0800013c

.bss Section Begin with Address 0x20000000 in SRAM and end in The Same Address

\_Stack\_top begin with Address 0x20001000 and End in Address 0x20000000 with means its Size is 0x1000 "4K Byte" As Expected



- Use Objdump Command to See Main file Sections

```

MINGW64~/c:/Users/ff/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_System_Diploma/E
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab2 with Startup_dot_s (main)
$ arm-none-eabi-objdump.exe -h main.o

main.o:          file format elf32-littlearm

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          000000d4  00000000  00000000  00000034  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000007  00000000  00000000  00000108  2**2
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  00000000  00000000  0000010f  2**0
    ALLOC
  3 .rodata        00000003  00000000  00000000  00000110  2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment       0000007f  00000000  00000000  00000113  2**0
    CONTENTS, READONLY
  5 .ARM.attributes 00000030  00000000  00000000  00000192  2**0
    CONTENTS, READONLY

```

- Use Objdump Command to See elf file Sections

The screenshot shows a Windows Command Prompt running MINGW64. The user has executed the command `$ arm-none-eabi-objdump.exe -h learn_in_depth_cortex_m3.elf`. The output displays the file format as `elf32-littlearm` and lists several sections:

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	00000133	08000000	08000000	00010000	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.data	00000008	08000134	08000134	00010134	2**2
	CONTENTS, ALLOC, LOAD, DATA					
2	.bss	00001003	20000000	20000000	00020000	2**2
	ALLOC					
3	.comment	0000007e	00000000	00000000	0001013c	2**0
	CONTENTS, READONLY					
4	.ARM.attributes	0000002e	00000000	00000000	000101ba	2**0
	CONTENTS, READONLY					

Handwritten red annotations include:

- .text Section in Flash Memory**: An arrow points to the VMA value `08000000` for the `.text` section.
- .data Section in Flash Memory**: An arrow points to the VMA value `08000134` for the `.data` section.
- The Next Lab We will move it to SRAM MEMORY**: This note is associated with the `.data` section.
- ..bss in SRAM Section**: An arrow points to the VMA value `20000000` for the `.bss` section.

The terminal prompt at the bottom shows the user returning to the main directory: `ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E... mbedded_C/Lesson_3/Lab2 with Startup_dot_s (main)`.

- Use nm Command To See Symbols Details in Elf file

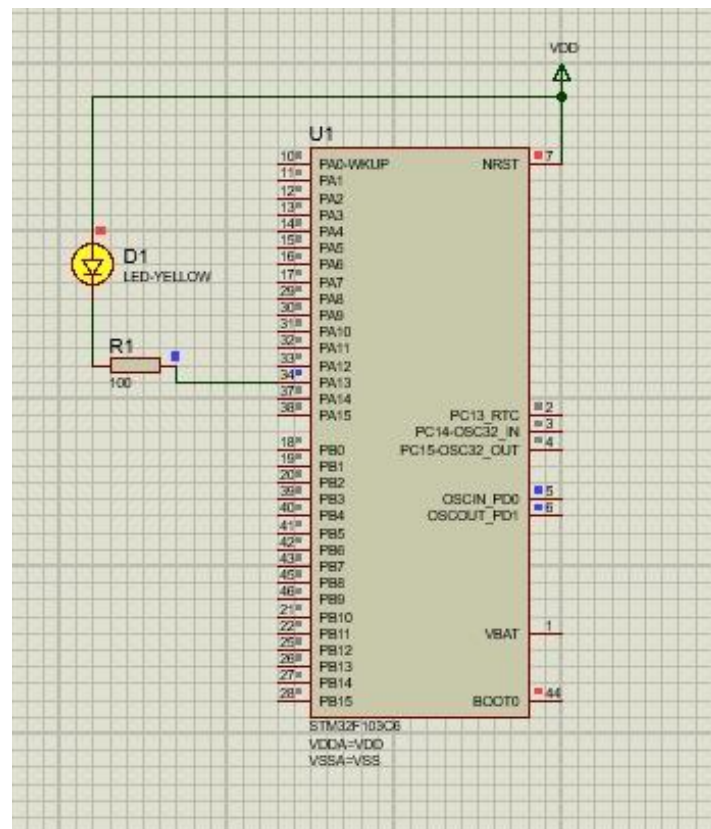
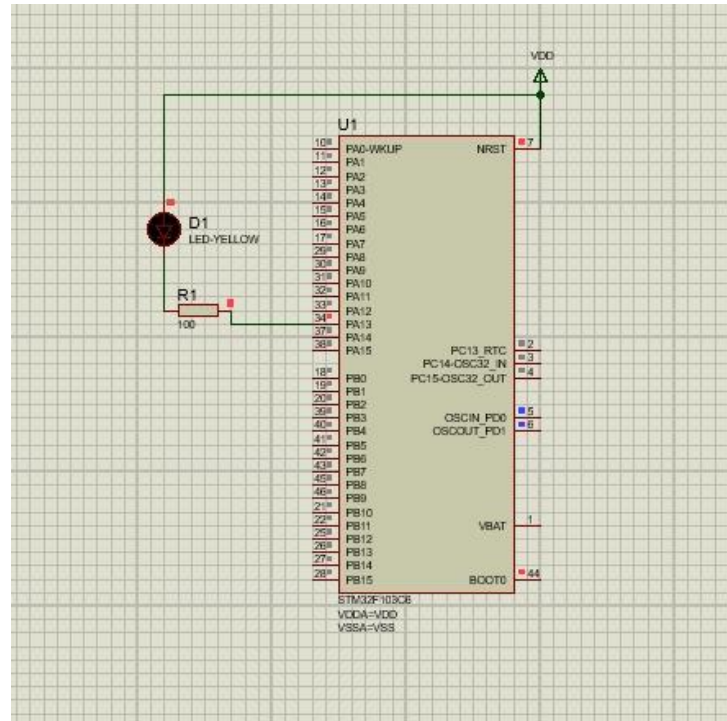
```
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab2 with Startup_dot_s (main)
$ arm-none-eabi-nm.exe learn_in_depth_cortex_m3.elf
00000000 B _E_bss
0800013c D _E_DATA
08000133 T _E_text
08000124 t _reset
20000000 B _S_bss
08000134 D _S_DATA
20001000 B _stack_top
20001000 B bss_variables
08000130 T const_variables
08000138 D g_variables
08000050 T main
08000134 D R_ODR
0800012c t Vector_handler
```

Symbols with Addresses

- Use Readelf Command To Check About The Entry Point

```
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/Embedded_C/Lesson_3/Lab2 with Startup_dot_s (main)
$ arm-none-eabi-readelf.exe -a learn_in_depth_cortex_m3.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                                ELF32
  Data:                                   2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                UNIX - System V
  ABI Version:                           0
  Type:                                  EXEC (Executable file)
  Machine:                                ARM
  Version:                                0x1
  Entry point address:                    0x80000000
  Start of program headers:               52 (bytes into file)
  Start of section headers:              66692 (bytes into file)
  Flags:                                  0x5000200, Version5 EABI, soft-float ABI
  Size of this header:                     52 (bytes)
  Size of program headers:                 32 (bytes)
  Number of program headers:                2
  Size of section headers:                 40 (bytes)
  Number of section headers:                9
  Section header string table index:      8
```

- Output in Proteus



## Lesson 3 Lab 3 with Startup.c file :

- Main.c file

```
main.c
21 typedef volatile unsigned int vint32_t;
22
23 //Registers Address
24 #define RCC_BASE 0x40021000
25 #define GPIOA_BASE 0x40010800
26 #define RCC_APB2ENR *(volatile uint32_t *) (RCC_BASE+0x18)
27 #define GPIOA_CRH *(volatile uint32_t *) (GPIOA_BASE+0x04)
28 #define GPIOA_ODR *(volatile uint32_t *) (GPIOA_BASE+0x0C)
29 //Bit Fields
30 #define RCC_IOPAEN (1<<2)
31 #define GPIOA13 (1UL<<13)
32
33
34 typedef union {
35     vint32_t all_fields;
36     struct {
37         vint32_t reserved:13 ;
38         vint32_t pin13:1 ;
39     }pin;
40 }R_ODR_t;
41
42 volatile R_ODR_t* R_ODR=(volatile R_ODR_t*)(GPIOA_BASE+0x0C);
43 unsigned char g_variables[3]={1,2,3};
44 unsigned char const const_variables[3]={1,2,3};
45 unsigned char volatile bss_Variable[3];
46 int main(void)
47 {
48     RCC_APB2ENR |=RCC_IOPAEN;
49     GPIOA_CRH &=0xFF0FFFFF ;
50     GPIOA_CRH |=0x00200000 ;
51     while(1)
52     {
53         // GPIO_ODR |= (1<<13); //Set Bit 13
54         R_ODR->pin.pin13=1;
55         for (int i=0 ;i<5000;i++);
56
57         // GPIO_ODR &=~(1<<13); //Clear Bit 13
58         R_ODR->pin.pin13=0;
59         for (int i=0 ;i<5000;i++);
60
61     }
62
63     return 0;
```



- Startup.c File

```
#include <stdint.h>
#define STACK_Start_SP 0x20001000
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 MM_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler"))) ;
void Bus_Fault(void) __attribute__ ((weak, alias ("Default_Handler"))) ;
void Usage_Fault_Handler(void) __attribute__ ((weak, alias ("Default_Handler"))) ;

extern unsigned int _stack_top;
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,
    (uint32_t) &Usage_Fault_Handler

};

extern unsigned int _E_text ;
extern unsigned int _S_DATA ;
extern unsigned int _E_DATA ;
extern unsigned int _S_bss ;
extern unsigned int _E_bss ;

void Reset_Handler(void)
{
    //copy data Section From Flash 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;

    for(int i=0;i<DATA_size;i++)
    {
        *((unsigned char *)P_dst++) = *((unsigned char *)P_src++) ;
    }

    //init .bss section in SRAM =0
    unsigned int bss_size =(unsigned char*) &_E_bss - (unsigned char*)&_S_bss ;
    P_dst=(unsigned char*)&_S_bss;
    for(int i=0 ;i<bss_size;i++)
    {
        *((unsigned char *)P_dst++) = (unsigned char)0 ;
    }

    //jump main()
    main();
}
```

To Write A Command to Compiler

This Command To Can rewrite The Defination of This Function in Onther File

This Command to make The Function Refer to The address of ;Reset\_Handler function and will be has its Special Address if it has A Defination

This Functions To Initialize The Content of Vector Table

This Command To Make this Section Output in .vectors Section

Calculate The Size of .data Section and copy it From flash to SRAM

Calculate The Size of .bss Section And initialize it with 0 in SRAM memory

Branch to The main



- LinkerScript File

```

/*Linker Script CortexM3
Guirguis Hedia
*/
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)
        . = ALIGN(4);
        _E_DATA = .;
    }>sram AT> flash

    .bss : {
        _S_bss = .;
        *(.bss*)
        _E_bss = .;
        . = ALIGN(4);
        . = . +0x1000;
        _stack_top = .;
    }>sram
}

```

Flash Memory Begin with Address 0x08000000 and its size =128k bytes

SRAM Memory Begin with Address 0x20000000 and its size =20k bytes

.text Section Will Be in Flash which Address begin with 0x08000000

.data Section will be in Flash which address =0x08000000 at Burn Time But Moves to SRAM in Execution Time which Address =0x20000000

This Command To Make The Memry Alignment Determine The Size of Stack with 4k Byte

- Make File

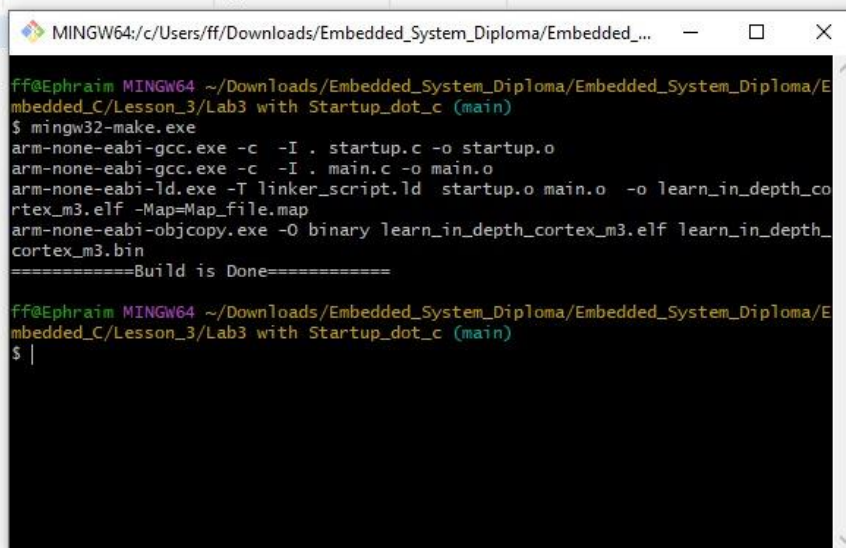
```

1  #!copyright : Guilguis media
2
3  CC=arm-none-eabi-
4  CFLAG=-gdwarf-2 -mcpu=cortex-m3
5  INCS=-I .
6  LIBS=
7  SRC= $(wildcard *.c)
8  OBJ= $(SRC:.c=.o)
9  As= $(wildcard *.s)
10 AsOBJ= $(As:.s=.o)
11
12 Project_Name=learn_in_depth_cortex_m3
13
14 all: $(Project_Name).bin
15     @echo "=====Build is Done=====
16 #startup.o: startup.s
17 # $(CC)as.exe $(CFLAGS) $< -o $@
18
19 %.o: %.c
20     $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@
21
22
23 $(Project_Name).elf: $(OBJ) $(AsOBJ)
24     $(CC)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(AsOBJ) -o $@ -Map=Map_file.map
25
26 $(Project_Name).bin: $(Project_Name).elf
27
28     $(CC)objcopy.exe -O binary $< $@
29
30 clean_all:
31     rm *.o *.elf *.bin
32
33 clean:
34     rm *.elf *.bin

```

- Build is Done

learn\_in\_depth\_cortex\_m3.bin  
learn\_in\_depth\_cortex\_m3.elf  
linker\_script.ld  
log.txt  
main.c  
main.o  
makefile  
Map\_file.map  
startup.c  
startup.o



```

MINGW64/c:/Users/ff/Downloads/Embedded_System_Diploma/Embedded_...
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab3 with Startup_dot_c (main)
$ mingw32-make.exe
arm-none-eabi-gcc.exe -c -I . startup.c -o startup.o
arm-none-eabi-gcc.exe -c -I . main.c -o main.o
arm-none-eabi-ld.exe -T linker_script.ld startup.o main.o -o learn_in_depth_co
rtex_m3.elf -Map=Map_file.map
arm-none-eabi-objcopy.exe -O binary learn_in_depth_cortex_m3.elf learn_in_dept
cortex_m3.bin
=====Build is Done=====
ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab3 with Startup_dot_c (main)
$ |

```

- Map file Details

Allocating common symbols			
Common symbol	size	file	
bss_Variable	0x3	main.o	

Memory Configuration

Flash begin at 0x08000000 & SRAM Begin at 0x20000000

Name	Origin	Length	Attributes
flash	0x08000000	0x00020000	xr
sram	0x20000000	0x00050000	xrw
*default*	0x00000000	0xffffffff	

Linker script and memory map

.vector Section Begin With 0x08000000 , At The Beginning of Flash Memory

```
.text          0x08000000      0x1ff
*(.vectors*)
.vectors       0x08000000      0x1c startup.o
               0x08000000              vectors
*(.text*)
.text          0x0800001c      0x10c startup.o
               0x0800001c              H_Fault_Handler
               0x0800001c              MM_Fault_Handler
               0x0800001c              Usage_Fault_Handler
               0x0800001c              Bus_Fault
               0x0800001c              Default_Handler
               0x0800001c              NMI_Handler
               0x08000038              Reset_Handler
.text          0x08000128      0xd4 main.o
               0x08000128              main
*(.rodata)
.rodata        0x080001fc      0x3 main.o
               0x080001fc              const_variables
               0x080001ff              _E_text = .

.glue_7        0x08000200      0x0
.glue_7        0x08000200      0x0 linker stubs

.glue_7t       0x08000200      0x0
.glue_7t       0x08000200      0x0 linker stubs

.vfp11_veneer  0x08000200      0x0

.rel.plt       0x08000200      0x0 startup.o

.data          0x20000000      0x8 load address 0x080001ff
               0x20000000              _S_DATA = .
*(.data)
.data          0x20000000      0x0 startup.o
.data          0x20000000      0x7 main.o
               0x20000000              R_ODR
               0x20000004              g_variables
               0x20000008              = ALIGN (0x4)
*fill*        0x20000007      0x1
               0x20000008              _E_DATA = .

.igot.plt      0x20000008      0x0 load address 0x08000207
.igot.plt      0x20000008      0x0 startup.o

.bss           0x20000008      0x1003 load address 0x08000207
               0x20000008              _S_BSS = .
*(.bss*)
.bss           0x20000008      0x0 startup.o
.bss           0x20000008      0x0 main.o
               0x20000008              _E_bss = .
               0x20000008              = ALIGN (0x4)
               0x20001008              = (. + 0x1000)
*fill*        0x20000008      0x1000
               0x20001008              _stack_top = .
COMMON        0x20001008      0x3 main.o
               0x20001008              bss_Variable

LOAD startup.o
LOAD main.o
OUTPUT (learn_in_depth_cortex_m3.elf elf32-littlearm)

.comment       0x00000000      0x7e
.comment       0x00000000      0x7e startup.o
.comment       0x0000007e      0x7f (size before relaxing)
.comment       0x0000007e      0x7f main.o
```

.data Section Here in SRAM with Address 0x20000000

.bss Section Begin with Address 0x20000008 in The SRAM Memory

Stack\_Top with Address 0x20001008  
Which Means The Stack Memory with Size 0x1000 "4k Byte"

.data Section Here in SRAM with Address 0x20000000

.bss Section Begin with Address 0x20000008 in The SRAM Memory

Stack\_Top with Address 0x20001008  
Which Means The Stack Memory with Size 0x1000 "4k Byte"

- Use Command Objdump To For main.o File

```

$ arm-none-eabi-objdump.exe -h main.o

main.o:      file format elf32-littlearm

Sections:
Idx Name          Size      VMA           LMA           File off  Algn
  0 .text          000000d4  00000000  00000000  00000034  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data          00000007  00000000  00000000  00000108  2**2
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss           00000000  00000000  00000000  0000010f  2**0
    ALLOC
  3 .rodata        00000003  00000000  00000000  00000110  2**2
    CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment       0000007f  00000000  00000000  00000113  2**0
    CONTENTS, READONLY
  5 .ARM.attributes 00000030  00000000  00000000  00000192  2**0
    CONTENTS, READONLY

```

- Use Command Objdump To For The output Elf file

learn\_in\_depth\_cortex\_m3.elf: file format elf32-littlearm

Idx	Name	Size	VMA	LMA	File off	Algn
0	.text	000001ff	08000000	08000000	00010000	2**2
	CONTENTS, ALLOC, LOAD, READONLY, CODE					
1	.data	00000008	20000000	080001ff	00020000	2**2
	CONTENTS, ALLOC, LOAD, DATA					
2	.bss	00001003	20000008	08000207	00020008	2**2
	ALLOC					
3	.comment	0000007e	00000000	00000000	00020008	2**0
	CONTENTS, READONLY					
4	.ARM.attributes	00000030	00000000	00000000	00020086	2**0
	CONTENTS, READONLY					

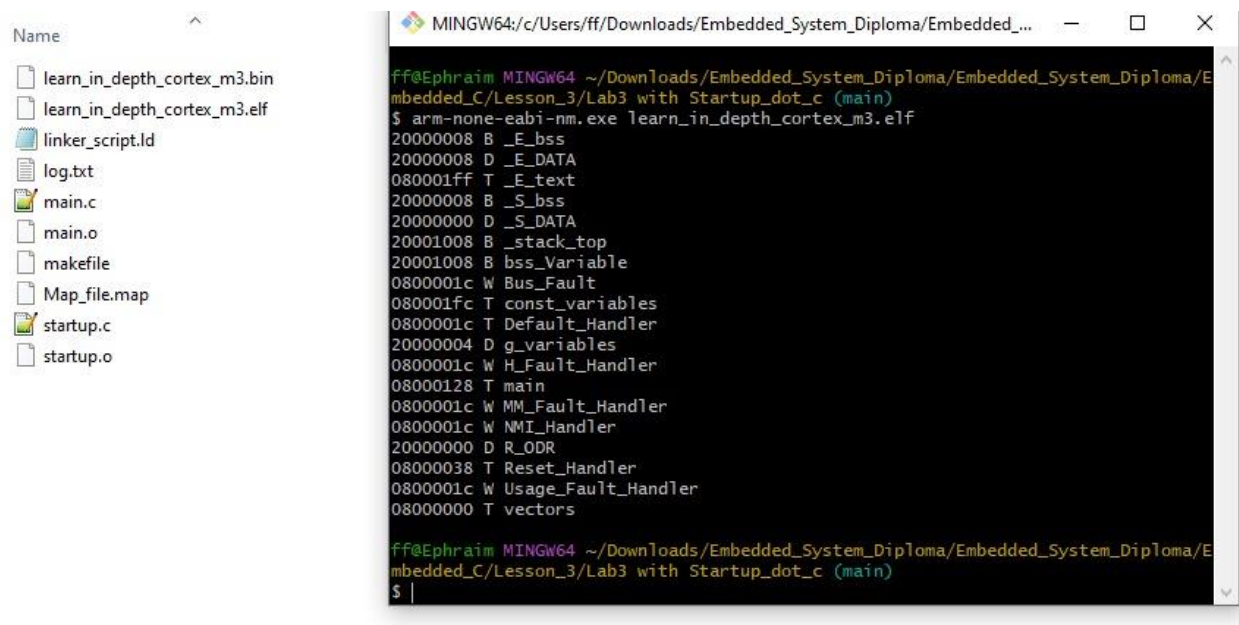
.text Section with Address 0x08000000 in Flash in The burn Time and in The Execution time

.data Section with Address 0x20000000 in SRAM in Exection time and with Address 0x80001ff in Flash in load Time

.bss Section with Address 0x20000000 in Execution Time



- Use Command nm to See The Symbols of Project elf



The image shows a file explorer on the left with a list of files: learn\_in\_depth\_cortex\_m3.bin, learn\_in\_depth\_cortex\_m3.elf, linker\_script.ld, log.txt, main.c, main.o, makefile, Map\_file.map, startup.c, and startup.o. To the right is a terminal window titled 'MINGW64: c:/Users/ff/Downloads/Embedded\_System\_Diploma/Embedded\_...' showing the command `arm-none-eabi-nm.exe learn_in_depth_cortex_m3.elf` and its output. The output lists various symbols and their types, such as `20000008 B _E_bss`, `20000008 D _E_DATA`, `080001ff T _E_text`, `20000008 B _S_bss`, `20000000 D _S_DATA`, `20001008 B _stack_top`, `20001008 B bss_Variable`, `0800001c W Bus_Fault`, `080001fc T const_variables`, `0800001c T Default_Handler`, `20000004 D g_variables`, `0800001c W H_Fault_Handler`, `08000128 T main`, `0800001c W MM_Fault_Handler`, `0800001c W NMI_Handler`, `20000000 D R_ODR`, `08000038 T Reset_Handler`, `0800001c W Usage_Fault_Handler`, and `08000000 T vectors`.

```

ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab3 with Startup_dot_c (main)
$ arm-none-eabi-nm.exe learn_in_depth_cortex_m3.elf
20000008 B _E_bss
20000008 D _E_DATA
080001ff T _E_text
20000008 B _S_bss
20000000 D _S_DATA
20001008 B _stack_top
20001008 B bss_Variable
0800001c W Bus_Fault
080001fc T const_variables
0800001c T Default_Handler
20000004 D g_variables
0800001c W H_Fault_Handler
08000128 T main
0800001c W MM_Fault_Handler
0800001c W NMI_Handler
20000000 D R_ODR
08000038 T Reset_Handler
0800001c W Usage_Fault_Handler
08000000 T vectors

ff@Ephraim MINGW64 ~/Downloads/Embedded_System_Diploma/Embedded_System_Diploma/E
mbedded_C/Lesson_3/Lab3 with Startup_dot_c (main)
$

```

- Use Command Readelf

```

$ arm-none-eabi-readelf.exe -a learn_in_depth_cortex_m3.elf
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                           ELF32
  Data:                               2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                        0
  Type:                               EXEC (Executable file)
  Machine:                            ARM
  Version:                            0x1
  Entry point address:                0x8000000
  Start of program headers:           52 (bytes into file)
  Start of section headers:          132108 (bytes into file)
  Flags:                              0x5000200, Version5 EABI, soft-float ABI
  Size of this header:                 52 (bytes)
  Size of program headers:             32 (bytes)
  Number of program headers:           2
  Size of section headers:            40 (bytes)
  Number of section headers:           9
  Section header string table index:  8

Section Headers:
 [Nr] Name                Type              Addr             Off             Size      ES Flg Lk Inf Al

```



- Output In Proteus

