Report

- **Debugging:**

-To debug the code we have to make sure that we have debug sections like that

```
ziade@LAPTOP-SO076FNK MINGW64 /d/Lab_1
$ readelf.exe learn-in-depth.elf -S
There are 16 section headers, starting at offset 0x85ac:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .startup          PROGBITS        00010000 008000 000010 00  AX  0   0  4
  [ 2] .text             PROGBITS        00010010 008010 000068 00  AX  0   0  4
  [ 3] .data             PROGBITS        00010078 008078 000064 00  WA  0   0  4
  [ 4] .ARM.attributes   ARM_ATTRIBUTES  00000000 0080dc 00002e 00      0   0  1
  [ 5] .comment          PROGBITS        00000000 00810a 000011 01  MS  0   0  1
  [ 6] .debug_line       PROGBITS        00000000 00811b 0000ac 00      0   0  1
  [ 7] .debug_info       PROGBITS        00000000 0081c7 000103 00      0   0  1
  [ 8] .debug_abbrev     PROGBITS        00000000 0082ca 0000bf 00      0   0  1
  [ 9] .debug_aranges    PROGBITS        00000000 008390 000060 00      0   0  8
  [10] .debug_loc        PROGBITS        00000000 0083f0 000058 00      0   0  1
  [11] .debug_str        PROGBITS        00000000 008448 000069 01  MS  0   0  1
  [12] .debug_frame      PROGBITS        00000000 0084b4 000054 00      0   0  4
  [13] .shstrtab         STRTAB          00000000 008508 0000a1 00      0   0  1
  [14] .symtab           SYMTAB          00000000 00882c 000210 10     15  28  4
  [15] .strtab           STRTAB          00000000 008a3c 000057 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  D (mbind), y (purecode), p (processor specific)
```

-To start debugging we will open gdb circuit in qemo tool for board that we debug on called versatilepb using this command

```
ziade@LAPTOP-SO076FNK MINGW64 /d/Lab_1
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth
.elf
```

-Then we run gdb

```
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 D:\Lab_1\learn-in-depth.elf...done.
(gdb)
```

-To target the server

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:3
3               ldr sp,= stack_top
(gdb)
```

-Some commands

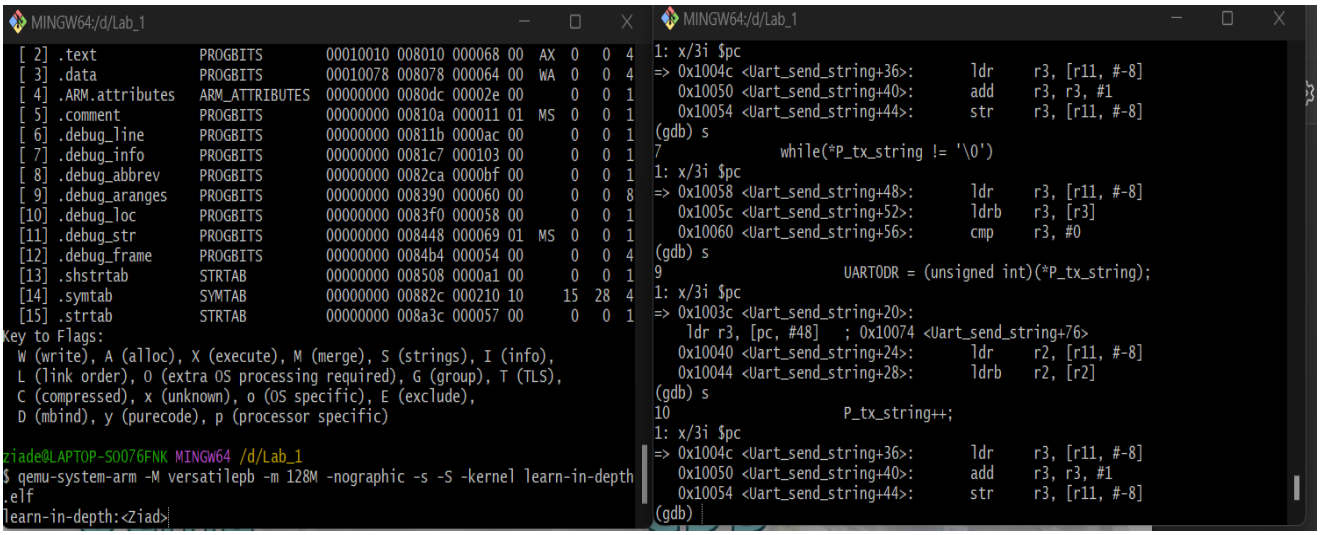
```
(gdb) b Uart_send_string
Breakpoint 3 at 0x10038: file uart.c, line 7.
(gdb) c
Continuing.

Breakpoint 1, reset () at startup.s:4
4               bl main
(gdb) display/3i $pc
1: x/3i $pc
=> 0x10004 <reset+4>:   bl      0x10010 <main>
   0x10008 <stop>:      b       0x10008 <stop>
   0x1000c <stop+4>:    ldrdeq  r1, [r1], -r12
(gdb) l
1       .globl reset
2       reset:
3               ldr sp,= stack_top
4               bl main
5       stop: b stop(gdb) c
Continuing.

Breakpoint 2, main () at app.c:6
6               Uart_send_string(string_buffer);
1: x/3i $pc
=> 0x10018 <main+8>:    ldr     r0, [pc, #4]    ; 0x10024 <main+20>
   0x1001c <main+12>:   bl      0x10028 <Uart_send_string>
   0x10020 <main+16>:   pop     {r11, pc}
(gdb) l
1       #include "uart.h"
2       unsigned char string_buffer[100]= "learn-in-depth:<Ziad>";
3
4       void main(void)
5       {
6               Uart_send_string(string_buffer);
7       }(gdb) watch string_buffer
Hardware watchpoint 4: string_buffer
(gdb) watch string_buffer
Hardware watchpoint 5: string_buffer
(gdb) print string_buffer[0]
$1 = 108 'l'
(gdb) print string_buffer
$2 = "learn-in-depth:<Ziad>", '\000' <repeats 78 times>
(gdb) backtrace
#0  main () at app.c:6
(gdb) c
Continuing.

Breakpoint 3, Uart_send_string (
    P_tx_string=0x10078 <string_buffer> "learn-in-depth:<Ziad>") at uart.c:7
7               while(*P_tx_string != '\0')
1: x/3i $pc
=> 0x10038 <Uart_send_string+16>:       b       0x10058 <Uart_send_string+48>
   0x1003c <Uart_send_string+20>:
    ldr r3, [pc, #48]   ; 0x10074 <Uart_send_string+76>
   0x10040 <Uart_send_string+24>:       ldr     r2, [r11, #-8]
(gdb) l
```

-We will step in C until uart.c and we will find that the string will printed character by character on the qemo terminal :



- **Makefile for lab 1:**

```
#@ copyright : Eng.Ziad

CC=arm-none-eabi-
CFLAGS= -mcpu=arm926ej-s  -g
INCS=-I .
LIBS=
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
AS = $(wildcard  *.s)
ASOBJ = $(AS:.s=.o)

project_name=learn-in-depth

all: $(project_name).bin
    @echo "------all build is done------"

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

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

$(project_name).elf: $(OBJ) $(ASOBJ)
    $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@

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


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

- **Lab2:**

- Board name : STM32f103c8t6 arm-cortex-m3 based .

- Flash starts with 0x08000000

- Sram starts with 0x20000000

1.Main .c

```c
typedef volatile unsigned int vuint32_t;

#include <stdint.h>

#define RCC_BASE        0x40021000
#define PORTA_BASE      0x40010800
#define RCC_IO          (1<<2)
#define RCC_APB2ENR    *((volatile uint32_t *) (RCC_BASE + 0x18))
#define GPIOA_CRH    *((volatile uint32_t *) (PORTA_BASE + 0x04))
#define GPIOA_ODR    *((volatile uint32_t *) (PORTA_BASE + 0x0c))

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*)(PORTA_BASE + 0x0c);
unsigned char g_variables[3]={1,2,3};
unsigned char const const_variables[3]={1,2,3};

volatile int i;
int main(void)
{


    RCC_APB2ENR |= RCC_IO ;    //to set bit 2
    GPIOA_CRH &= 0xff0fffff; //to protect the bits
    GPIOA_CRH |= 0x00200000; //to make bits(20>>24)=2

    while(1)
    {
        R_ODR->pin.p_13= 1 ;            //set bit 13
        for( i=0 ; i<5000 ; i++);    //random delay
        R_ODR->pin.p_13= 0;           //to clear bit 13
        for(  i=0 ; i<5000 ; i++);

    }



    return 0;
}
```

2.Startup.s

```
/* startup_cortexM3.s
Eng.Ziad
*/
/*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 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 SysTrick */
.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
```

3.Linker script

```
/* linker script CortexM3
Eng.Ziad
*/

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


}
```

4.Make file

```
#@ copyright : Eng.Ziad

CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2
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 "------all build is done------"

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

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

$(project_name).elf: $(OBJ) $(ASOBJ)
    $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@

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


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


```

- **Lab 2 with Startup.c**

1.main.c

```c
 */
typedef volatile unsigned int vuint32_t;

#include <stdint.h>

#define RCC_BASE        0x40021000
#define PORTA_BASE      0x40010800
#define RCC_IO          (1<<2)
#define RCC_APB2ENR    *((volatile uint32_t *) (RCC_BASE + 0x18))
#define GPIOA_CRH      *((volatile uint32_t *) (PORTA_BASE + 0x04))
#define GPIOA_ODR      *((volatile uint32_t *) (PORTA_BASE + 0x0c))

extern void NMI_Handler(void)
{

}
extern void Bus_Fault(void)
{

}
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*)(PORTA_BASE + 0x0c);
unsigned char g_variables[3]={1,2,3};
unsigned char const const_variables[3]={1,2,3};


volatile int i;
int main(void)
{


    RCC_APB2ENR |= RCC_IO;    //to set bit 2
    GPIOA_CRH &= 0xff0fffff; //to protect the bits
    GPIOA_CRH |= 0x00200000; //to make bits(20>>24)=2

    while(1)
    {
        R_ODR->pin.p_13= 1 ;            //set bit 13
        for( i=0 ; i<5000 ; i++);    //random delay
        R_ODR->pin.p_13= 0;           //to clear bit 13
        for(  i=0 ; i<5000 ; i++);

    }


```

2.startup.c

```c
//startup.c
//Eng.Ziad

#include <stdint.h>
extern int main(void);

void Reset_Handler(void);
void Default_handler();


//to make it easy to overrid & all have the same handler to minimize the size

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() __attribute__ ((weak,alias("Default_handler")));;
void Usage_Fault_Handler() __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 _S_DATA;
extern unsigned int _E_DATA;
extern unsigned int _S_bss;
extern unsigned int _E_bss;
extern unsigned int _E_text;

void Reset_Handler (void)
{
    volatile int i;
    //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;

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

    }

    // initialize the .bss with zero
    unsigned int bss_size = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss ;
    P_dst = (unsigned char*)&_S_bss ;
```

```c
    for(i=0 ; i<bss_size ; i++)
    {
        *((unsigned char*)P_dst++)=(unsigned char)0;
    }

    // jump to main

    main();
}
void Default_handler()
{
    Reset_Handler();
}
```

3.linker script

```
/* linker script CortexM3
Eng.Ziad
*/

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


}
```

4. make file

```
#@ copyright : Eng.Ziad

CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m3 -mthumb -gdwarf-2
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 "------all build is done------"

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

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

$(project_name).elf: $(OBJ) $(ASOBJ)
    $(CC)ld.exe -T linker_script.ld -Map=map_file.map $(OBJ) $(ASOBJ) -o $@

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


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

- .data section has LMA within flash range and it will be copied

to sram so it has VMA within start of sram as we want

- .bss section has VMA within sram range .

```
ziade@LAPTOP-SO076FNK MINGW64 /d/lab2.c
$ arm-none-eabi-objdump.exe -h learn-in-depth_cortex_m3.elf

learn-in-depth_cortex_m3.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000001e0  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000008  20000000  080001e0  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001004  20000008  080001e8  00010008  2**2
                  ALLOC
  3 .debug_info   00000301  00000000  00000000  00010008  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 000001c6  00000000  00000000  00010309  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    000000e8  00000000  00000000  000104cf  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000040  00000000  00000000  000105b7  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   0000014d  00000000  00000000  000105f7  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    0000015e  00000000  00000000  00010744  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000011  00000000  00000000  000108a2  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000108b3  2**0
                  CONTENTS, READONLY
 11 .debug_frame  000000a4  00000000  00000000  000108e8  2**2
                  CONTENTS, READONLY, DEBUGGING
```

- As we Mention NMI_Fault handler & Bus_handler in main.c(override on weak symbol in main.c)

So they didn’t take the same address of Default_handler and they take different addresses

-Lets see the map file

```
 *(.rodata)
 .rodata        0x080001dc        0x4 main.o
                0x080001dc                const_variables
                0x080001e0                _E_text = .

.glue_7         0x080001e0        0x0
 .glue_7        0x00000000        0x0 linker stubs

.glue_7t        0x080001e0        0x0
 .glue_7t       0x00000000        0x0 linker stubs

.vfp11_veneer   0x080001e0        0x0
 .vfp11_veneer  0x00000000        0x0 linker stubs

.v4_bx          0x080001e0        0x0
 .v4_bx         0x00000000        0x0 linker stubs

.iplt           0x080001e0        0x0
 .iplt          0x00000000        0x0 main.o

.rel.dyn        0x080001e0        0x0
 .rel.iplt      0x00000000        0x0 main.o

.data           0x20000000        0x8 load address 0x080001e0
                0x20000000                _S_DATA = .
 *(.data)
 .data          0x20000000        0x8 main.o
                0x20000000                R_ODR
                0x20000004                g_variables
 .data          0x20000008        0x0 startup.o
                0x20000008                _E_DATA = .

.igot.plt       0x20000008        0x0 load address 0x080001e8
 .igot.plt      0x00000000        0x0 main.o

.bss            0x20000008     0x1004 load address 0x080001e8
```

ALGIN

```
                0x20000008                _E_DATA = .

.igot.plt       0x20000008        0x0 load address 0x080001e8
 .igot.plt      0x00000000        0x0 main.o

.bss            0x20000008     0x1004 load address 0x080001e8
                0x20000008                _S_bss = .
 *(.bss)
 .bss           0x20000008        0x0 main.o
 .bss           0x20000008        0x0 startup.o
                0x20000008                . = ALIGN (0x4)
                0x20000008                _E_bss = .
                0x20000008                . = ALIGN (0x4)
                0x20001008                . = (. + 0x1000)
 *fill*         0x20000008     0x1000
                0x20001008                _stack_top = .
 COMMON         0x20001008        0x4 main.o
                0x20001008                i
```