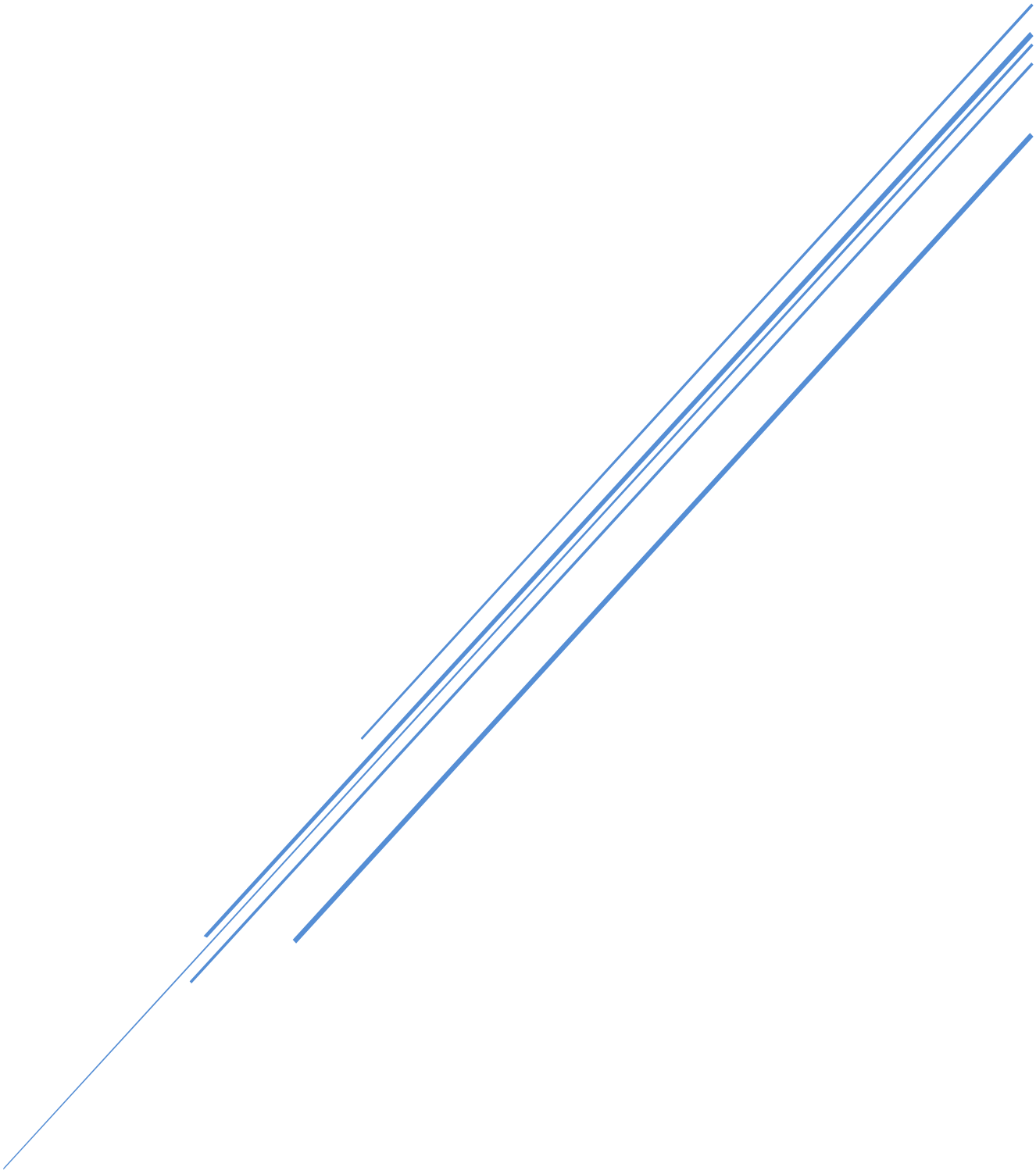
LAB 1

Lesson 2 - Embedded C

By Mostafa Mahmoud Ali



Learn-In-Depth Diploma by Eng. Kerolos Shenoda

1) Description

In this lab, I had to create BareMetal Software to send a "learn-in-depth:<Mostafa Mahmoud>" using UART of ARM VersatilePB board.

2) Requirements

- ***Install QEMU (Quick EMUlator).***
- ***Install GNU ARM Embedded Toolchain***
- ***Implement the C code files.***
- ***Create both the startup code and the linker script.***

3) C Code Implementation

File Name	Implementation

uart.h	uart.h
	`#ifndef _UART_H_`
	`#define _UART_H_`
	
	`void UART_sendString(unsigned char* P_tx_string);`
	
	`#endif`

uart.c	uart.c > UART_sendString(unsigned char *)
	`#include "uart.h"`
	
	`/* UART Register */`
	`#define UART0DR *((volatile unsigned int* const)((unsigned int*)0x101f1000))`
	
	`void UART_sendString(unsigned char* P_tx_string)`
	`{`
	`/* Loop until the end of the string */`
	`while(*P_tx_string != '\0')`
	`{`
	`UART0DR = (unsigned int)(*P_tx_string); /* Transmit char */`
	`P_tx_string++; /* Move to the next char */`
	`}`
	`}`

app.c	app.c > main(void)
	`#include "uart.h"`
	
	`// Note: 100 Bytes in decimal -> 64 in hexa`
	
	`// Stored inside the .data section`
	`unsigned char string_buffer1[100] = "Learn-in-depth:<Mostafa Mahmoud>";`
	
	`// Stored inside the .rodata section`
	`unsigned char const string_buffer2[100] = "Learn-in-depth:<Mostafa Mahmoud>";`
	
	`void main(void)`
	`{`
	`UART_sendString(string_buffer1);`
	`}`

4) Generating (app/uart).o objects files

- *Using GNU ARM-Cross-toolchain "arm-none-eabi-gcc.exe"*

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded
 C/Lesson 2/Lab 1
$ arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . app.c -o app.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded
 C/Lesson 2/Lab 1
$ ls
app.c  app.o  uart.c  uart.h

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded
 C/Lesson 2/Lab 1
$ arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . uart.c -o uart.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded
 C/Lesson 2/Lab 1
$ ls
app.c  app.o  uart.c  uart.h  uart.o
```

5) Navigate the .obj files (relocatable images)

- *Using ARM-Cross toolchain Bin Utilities (objdump)*

  Where -h → Display the contents of the section headers.

  -D → Display assembler contents of all sections

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit
n 2/Lab 1
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000018  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b0  2**0
                  ALLOC
  3 .debug_info   0000006c  00000000  00000000  000000b0  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 0000005a  00000000  00000000  0000011c  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000002c  00000000  00000000  00000176  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  000001a2  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   00000035  00000000  00000000  000001c2  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    0000007d  00000000  00000000  000001f7  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  00000274  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000032  00000000  00000000  00000286  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000002c  00000000  00000000  000002b8  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

Sections With Debugging Info

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . uart.c -o uart.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . app.c -o app.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ ls
app.c  app.o  uart.c  uart.h  uart.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000018  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b0  2**0
                  ALLOC
  3 .comment      00000012  00000000  00000000  000000b0  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  000000c2  2**0
                  CONTENTS, READONLY
```

Sections Without Debugging Info

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . app.c -o app.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ arm-none-eabi-gcc.exe -c -mcpu=arm926ej-s -I . uart.c -o uart.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
n 2/Lab 1
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000018  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000064  00000000  00000000  0000004c  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b0  2**0
                  ALLOC
  3 .rodata       00000064  00000000  00000000  000000b0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .comment      00000012  00000000  00000000  00000114  2**0
                  CONTENTS, READONLY
  5 .ARM.attributes 00000032  00000000  00000000  00000126  2**0
                  CONTENTS, READONLY
```

Sections After adding Const Data
Without Debugging Info

```
app.o:     file format elf32-littlearm


Disassembly of section .text:

00000000 <main>:
   0: e92d4800    push  {fp, lr}
   4: e28db004    add   fp, sp, #4
   8: e59f0004    ldr   r0, [pc, #4]   ; 14 <main+0x14>
   c: ebfffffe    bl 0 <UART_sendString>
  10: e8bd8800    pop   {fp, pc}
  14: 00000000    andeq r0, r0, r0

Disassembly of section .data:

00000000 <string_buffer1>:
   0: 7261654c    rsbvc r6, r1, #76, 10   ; 0x13000000
   4: 6e692d6e    cdpvs 13, 6, cr2, cr9, cr14, {3}
   8: 7065642d    rsbvc r6, r5, sp, lsr #8
   c: 3c3a6874    ldccc 8, cr6, [sl], #-464  ; 0xfffffe30
  10: 74736f4d    ldrbtvc  r6, [r3], #-3917  ; 0xf4d
  14: 20616661    rsbcs r6, r1, r1, ror #12
  18: 6d68614d    stfvse   f6, [r8, #-308]!  ; 0xfffffecc
  1c: 3e64756f    cdpcc 5, 6, cr7, cr4, cr15, {3}
   ...

Disassembly of section .rodata:

00000000 <string_buffer2>:
   0: 7261654c    rsbvc r6, r1, #76, 10   ; 0x13000000
   4: 6e692d6e    cdpvs 13, 6, cr2, cr9, cr14, {3}
   8: 7065642d    rsbvc r6, r5, sp, lsr #8
   c: 3c3a6874    ldccc 8, cr6, [sl], #-464  ; 0xfffffe30
  10: 74736f4d    ldrbtvc  r6, [r3], #-3917  ; 0xf4d
  14: 20616661    rsbcs r6, r1, r1, ror #12
  18: 6d68614d    stfvse   f6, [r8, #-308]!  ; 0xfffffecc
  1c: 3e64756f    cdpcc 5, 6, cr7, cr4, cr15, {3}
   ...
```

Generating the disassembly file (app.s) from the binary
Using Command: ***$ arm-none-eabi-objdump.exe -D app.o > app.s***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Un
$ arm-none-eabi-objdump.exe -s app.o

app.o:     file format elf32-littlearm

Contents of section .text:
 0000 00482de9 04b08de2 04009fe5 feffffeb  .H-.............
 0010 0088bde8 00000000                    ........
Contents of section .data:
 0000 4c656172 6e2d696e 2d646570 74683a3c  Learn-in-depth:<
 0010 4d6f7374 61666120 4d61686d 6f75643e  Mostafa Mahmoud>
 0020 00000000 00000000 00000000 00000000  ................
 0030 00000000 00000000 00000000 00000000  ................
 0040 00000000 00000000 00000000 00000000  ................
 0050 00000000 00000000 00000000 00000000  ................
 0060 00000000                             ....
Contents of section .rodata:
 0000 4c656172 6e2d696e 2d646570 74683a3c  Learn-in-depth:<
 0010 4d6f7374 61666120 4d61686d 6f75643e  Mostafa Mahmoud>
 0020 00000000 00000000 00000000 00000000  ................
 0030 00000000 00000000 00000000 00000000  ................
 0040 00000000 00000000 00000000 00000000  ................
 0050 00000000 00000000 00000000 00000000  ................
 0060 00000000                             ....
Contents of section .comment:
 0000 00474343 3a202847 4e552920 342e372e  .GCC: (GNU) 4.7.
 0010 3200                                 2.
Contents of section .ARM.attributes:
 0000 41310000 00616561 62690001 27000000  A1...aeabi..'...
 0010 0541524d 39323645 4a2d5300 06050801  .ARM926EJ-S.....
 0020 09011204 14011501 17031801 19011a01  ................
 0030 1e06                                 ..
```

Display the full content of all sections requested

6) StartUp Code

1. A Simple Startup:

- ***Create a reset section and call main ().***
- ***Initialize Stack***

startup.s

```
1    .globl reset
2
3    reset:
4        ldr sp, =stack_top
5        bl main
6
7    stop: b stop
```

2. To generate the startup code object file

- ***Use the arm toolchain assembler → arm-none-eabi-as.exe***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - En
n 2/Lab 1
$ arm-none-eabi-as.exe -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - En
n 2/Lab 1
$ ls
app.c  app.o  app.s  startup.o  startup.s  uart.c  uart.h  uart.o

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - En
n 2/Lab 1
$ arm-none-eabi-objdump.exe -h startup.o

startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000000c  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000040  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000040  2**0
                  ALLOC
  3 .ARM.attributes 00000022  00000000  00000000  00000040  2**0
                  CONTENTS, READONLY
```

7) Linker Script

- ***The Linker file (linkerScript.lb)***

```
ENTRY(reset)  /* Define the entry point of the application */

/* List of the memory sections */
MEMORY
{
    Mem (rwx) : ORIGIN = 0x00000000, LENGTH = 64M
}

SECTIONS
{
    . = 0x10000 ;  /* According to the specs */
    .startup . :
    {
        startup.o(.text)
    }> Mem

    .text :
    {
        *(.text) *(.rodata)   /* merging all remaining .txt and .rodata (input sections) to the output.txt section */
    }> Mem

    .data :
    {
        *(.data)
    }> Mem

    .bss :
    {
        *(.bss) *(COMMON)
    }> Mem

    . = . + 0x1000 ;    /* 4KB of stack Memory */
    stack_top = . ;
}
```

- ***Using ARM-Cross toolchain Bin Utilities (nm) To read the symbols.***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_buffer1
00000000 R string_buffer2
         U UART_sendString

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ arm-none-eabi-nm.exe uart.o
00000000 T UART_sendString
```

Note:
As app.o, startup.o, and uart. o are all object files (relocatable files) then these symbols don't have their actual address yet

- ***Linking all objects and producing the executable and map files***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ arm-none-eabi-ld.exe -T linkerScript.lb app.o uart.o startup.o -o Mostafa_LearnInDepth.elf -Map=Map_file.map

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ ls
Map_file.map  Mostafa_LearnInDepth.elf  app.c  app.o  app.s  linkerScript.lb  startup.o  startup.s  uart.c  uart.h  uart.o
```

- ***Analyze the executable file (.elf file)***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedd
it 3 - Embedded C/Lesson 2/Lab 1
$ arm-none-eabi-nm.exe Mostafa_LearnInDepth.elf
00010010 T main
00010000 T reset
00011140 D stack_top
00010008 t stop
000100dc D string_buffer1
00010078 T string_buffer2
00010028 T UART_sendString
```

Symbols in the executable

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3
$ arm-none-eabi-objdump.exe -h Mostafa_LearnInDepth.elf

Mostafa_LearnInDepth.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .startup      00000010  00010000  00010000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         000000cc  00010010  00010010  00008010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000064  000100dc  000100dc  000080dc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  00008140  2**0
                  CONTENTS, READONLY
  4 .comment      00000011  00000000  00000000  0000816e  2**0
                  CONTENTS, READONLY
```

Sections in the executable

- ***Analyze the map file.***

```
Map_file.map > Linker script and memory map > .text > app.o

Memory Configuration

Name             Origin             Length             Attributes
Mem              0x00000000         0x04000000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map

                 0x00010000                 . = 0x10000

.startup         0x00010000       0x10
 startup.o(.text)
 .text           0x00010000       0x10 startup.o
                 0x00010000                 reset

.text            0x00010010       0xcc
 *(.text)
 .text           0x00010010       0x18 app.o
                 0x00010010                 main
 .text           0x00010028       0x50 uart.o
                 0x00010028                 UART_sendString
 *(.rodata)
 .rodata         0x00010078       0x64 app.o
                 0x00010078                 string_buffer2

.glue_7          0x000100dc        0x0
 .glue_7         0x00000000        0x0 linker stubs

.glue_7t         0x000100dc        0x0
 .glue_7t        0x00000000        0x0 linker stubs

.vfp11_veneer    0x000100dc        0x0
 .vfp11_veneer   0x00000000        0x0 linker stubs

.v4_bx           0x000100dc        0x0
 .v4_bx          0x00000000        0x0 linker stubs
```

```
.vfp11_veneer    0x000100dc        0x0
 .vfp11_veneer   0x00000000        0x0 linker stubs

.v4_bx           0x000100dc        0x0
 .v4_bx          0x00000000        0x0 linker stubs

.iplt            0x000100dc        0x0
 .iplt           0x00000000        0x0 startup.o

.rel.dyn         0x000100dc        0x0
 .rel.iplt       0x00000000        0x0 startup.o

.data            0x000100dc       0x64
 *(.data)
 .data           0x000100dc        0x0 startup.o
 .data           0x000100dc       0x64 app.o
                 0x000100dc                 string_buffer1
 .data           0x00010140        0x0 uart.o

.igot.plt        0x00010140        0x0
 .igot.plt       0x00000000        0x0 startup.o

.bss             0x00010140        0x0
 *(.bss)
 .bss            0x00010140        0x0 startup.o
 .bss            0x00010140        0x0 app.o
 .bss            0x00010140        0x0 uart.o
 *(COMMON)
                 0x00011140                 . = (. + 0x1000)
                 0x00011140                 stack_top = .
LOAD app.o
LOAD uart.o
LOAD startup.o
OUTPUT(Mostafa_LearnInDepth.elf elf32-littlearm)
```

- ***Use the readelf Binary utilities to make sure the entry point at address 0x10000***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Less
$ arm-none-eabi-readelf.exe -a Mostafa_LearnInDepth.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:               0x10000
  Start of program headers:          52 (bytes into file)
  Start of section headers:          33224 (bytes into file)
  Flags:                             0x5000002, has entry point, Version5 EABI
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         1
  Size of section headers:           40 (bytes)
  Number of section headers:         9
  Section header string table index: 6

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 0000cc 00  AX  0   0  4
  [ 3] .data             PROGBITS        000100dc 0080dc 000064 00  WA  0   0  4
  [ 4] .ARM.attributes   ARM_ATTRIBUTES  00000000 008140 00002e 00      0   0  1
  [ 5] .comment          PROGBITS        00000000 00816e 000011 01  MS  0   0  1
  [ 6] .shstrtab         STRTAB          00000000 00817f 000049 00      0   0  1
  [ 7] .symtab           SYMTAB          00000000 008330 000190 10      8  19  4
  [ 8] .strtab           STRTAB          00000000 0084c0 000066 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

8) Generating the Binary file

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ arm-none-eabi-objcopy.exe -O binary Mostafa_LearnInDepth.elf Mostafa_LearnInDepth.bin

mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ ls
Map_file.map              Mostafa_LearnInDepth.elf  app.o  linkerScript.lb  startup.s  uart.h
Mostafa_LearnInDepth.bin  app.c                     app.s  startup.o        uart.c     uart.o
```

9) Run the program in the QEMU Simulator ***("VersatilePB physical Board")***

```
mosta@LAPTOP-J66NQ41S MINGW64 /e/Mastering Embedded Systems/Unit 3 - Embedded C/Lesson 2/Lab 1
$ qemu-system-arm.exe -M versatilepb -m 128M -nographic -kernel Mostafa_LearnInDepth.bin
Learn-in-depth:<Mostafa Mahmoud>
```