Unit 2 - Lesson 2 - Lab 1 Report

Creating a bare metal software from scratch

Table of content :

1- Introduction:

We will create a bare metal software together from scratch using versitalePB microcontroller based on ARM926EJ-S microprocessor , since it has great property enable us from transmitting a data when we write it on register of Uart modules , here we will use data register of port 0 that has address of 0x0 .

At first , we will create group of file : app.c , uart.c and uart.h , app.c and uart.c will include uart.h and then compile it , outputing a app.o and uart.o .

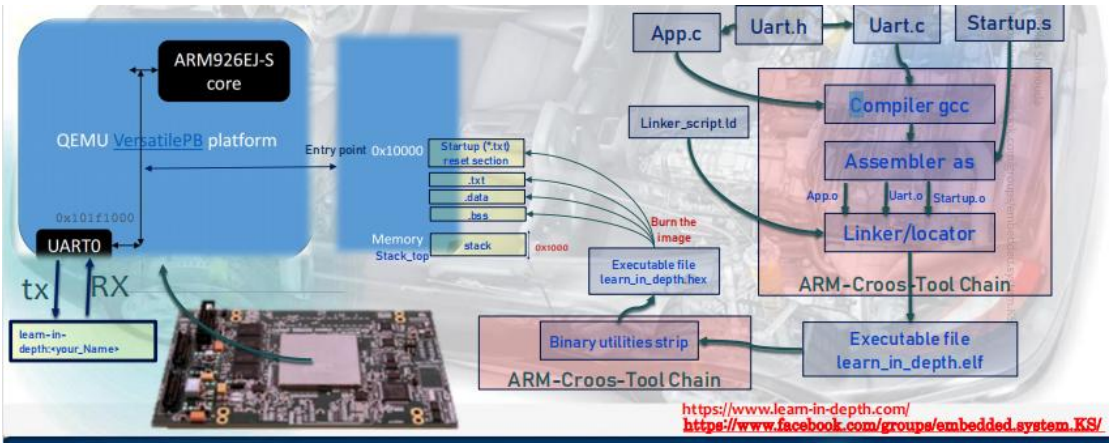
At second , we will create startup.s file containing very important information like the entry point of program and initialize the stack pointer register to enter the main function (entry point of program) , then we will compile it outputing startup.o file .

At third , we will creating linker_script.ld that containing information about the entry point of program , memory start , memory size , sections configuration and very important properties called virtual memory address (VMA) that responsible of mapping the addresses of memory at runtime and load memory address (LMA) that responsible of mapping the addresses of memory at loading time of code to physical
board .

At fifth , we will link : app.o , uart.o , startup.o and linker_script.ld files outputing Learn_in_depth.elf that containing the binary instructions of our software and some other sections to extract the binary instructions only , we will use objcopy feature of Arm-cross-tooltchain and then outputing Learn_in_depth.bin.

At last , we will simulate our software on QEMU , QEMU (short for "Quick EMUlator") is a free and open-source machine emulator and virtualizer

In figure below , the summery of Lab :



2-Source files :

2.1-App.c :

Simple c code containing header file Uart.h which containing the prototype of Uart_send_String function that prints string character by character
so after we declared the string string_name ,we passed the pointer of first element in string to be transmitted

```c
/*****************************************************************/
/*         @author : Osama Youssef                               */
/*         @file   : main.c                                      */
/*         @brief  : main function  that containg the string -we want */
/*                     transmit it-and calls Uart_send_String    */
/*                                                               */
/*****************************************************************/


 #include "Uart.h"

// define the string

unsigned char string_name[100] ="Learn-in-depth : < Osama Youssef >";


void main(void)
{

    Uart_send_String(string_name);
}
```

2.2-Uart.h:

Header file include the prototype of Uart_send_String function and some compilation macros to protect the header

```c
/*****************************************************************/
/*         @author : Osama Youssef                               */
/*         @file   : uart.h                                      */
/*         @brief  : Header file contianing the Uart_send_String */
/*                    prototype                                  */
/*****************************************************************/


// HEADER PROTECTION

#ifndef _UART_H_
#define _UART_H_

/* Uart_send_String Function Prototype */
void Uart_send_String(unsigned char * ptr_tx_string);

#endif /*_UART_H_*/

```

2.3-uart.c :

C code include the uart.h header file and then define a macro of address of data register of uart port that's equals 0x101f1000 (from specs) , after that we define a function whose argument is a pointer to char and its functionality to assign which pointer ptr_tx_string points to after checking that this value isn't equals null character and then increment the pointer to points to next char .

```c
/*****************************************************************/
/*          @author : Osama Youssef                              */
/*          @file   : uart.c                                     */
/*          @brief  : Function prints string using uart0 module  */
/*                    in versitalePB microcontroller based on    */
/*                    ARM926EJ-S microprocessor                  */
/*****************************************************************/


 #include "uart.h"

/* Register Defination */

#define UART0DR_BASE  0x101f1000

#define UART0DR  *((volatile unsigned int * const )((unsigned int *)(UART0DR_BASE)))

/* Uart_send_String Function Defination */

void Uart_send_String(unsigned char * ptr_tx_string)
{
    while(* ptr_tx_string != '\0')
    {
        UART0DR = (unsigned int) (* ptr_tx_string);
        ptr_tx_string++ ;
    }

}
```

2.4-Startup.s :

Assembly file -as we mentioned before-has very important information like stack top address and entry point of program (main) , if main program doesn't end it will jump to stop label , and after that it will branch to stop again and so on .

```
.global reset

reset:
   ldr sp , =stack_top
   bl main
stop:
    b stop

```

2.5-linker_script.ld :

It is very important file containing the layout of memory section and size , at first you will see the ENTRY keyword that's refer to the reset section (entry point of program) , and then you will see the memory specifications like size and begin point then you will see the memory sections like .text section , its roll to group all .text sections of all files enters the linker and start at address 0x00010000 , .data section , .bss section and .comment section .
After that we will reserve a 0x1000 memory size and start push from top 0x0001100 .

```
ENTRY(reset)

MEMORY
{

    MEM(rwx): ORIGIN = 0x00000000 , LENGTH = 64M

}

SECTIONS
{
    . = 0x00010000 ;
  .startup . :
    {
       startup.o(.text)

    }> MEM
  .text:
   {

    *(.text)

   }>MEM
  .data:
   {

     *(.data)

   }> MEM
  .bss:
   {

     *(.bss)

  }>MEM
  .comment:
   {

    *(.comment) *(COMMON)

  }> MEM

  . = . + 0x1000 ;

  stack_top = . ;
}
```

3-Section Headers

In these group of files , we will be considered about .test , .data , .rodata , .bss and other section will be removed at final executable file , we will determine the size of each section , VMA and LMA .

3.1 - app.o header :

1- .text section : has size of 24 bytes (containing the instructions set of file).
2- .data section : has size of 100 bytes (containing global and static data of file).
3- .bss section : has size of 0 bytes (since no uninitialized data in ROM).
4- .rodata section : has size of 0 bytes (since no global constant data in file).
Since file is relocatable file , all VMA's and LMA's is 0x00000000.

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ 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
```

3.2 - uart.o header :

1.text section : has size of 80 bytes (containing the instructions set of file).
2.data section : has size of 0 bytes (since no global and static data in file).
3.bss section : has size of 0 bytes (since no uninitialized data in ROM).
4.rodata section : has size of 0 bytes (since no global constant data in file).
Since file is relocatable file , all VMA's and LMA's is 0x00000000.

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ arm-none-eabi-objdump.exe -h uart.o

uart.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000050  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000084  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000084  2**0
                  ALLOC
  3 .comment      00000012  00000000  00000000  00000084  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  00000096  2**0
                  CONTENTS, READONLY
```

3.3-startup.o header :

1- .text section : has size of 16 bytes (containing the instructions set of file).
2- .data section : has size of 0 bytes (since no global and static data of file).
3- .bss section : has size of 0 bytes (since no uninitialized data in ROM).
4- .rodata section : has size of 0 bytes (since no global constant data in file).
Since file is relocatable file , all VMA's and LMA's is 0x00000000

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ 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         00000010  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000044  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000044  2**0
                  ALLOC
  3 .ARM.attributes 00000022  00000000  00000000  00000044  2**0
                  CONTENTS, READONLY
```

3.4-learn_in_depth.elf header :

1-.startup section : has size of 16 bytes at address 0x00010000 (containing the instructions set of startup file) .
2-.text section : has size of 104 bytes at address 0x00010010 (containing the instructions set of source files) .
3-.data section : has size of 100 bytes at address 0x00010078 (containing global and static data of file).
4-.bss section : has size of 0 bytes (since no uninitialized data in ROM).
5-.rodata section : has size of 0 bytes (since no global constant data in file).
Since there is one memory ---> VMA = LMA .

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ arm-none-eabi-objdump.exe -h learn_in_depth.elf

learn_in_depth.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         00000068  00010010  00010010  00008010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000064  00010078  00010078  00008078  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .comment      00000011  000100dc  000100dc  000080dc  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 0000002e  00000000  00000000  000080ed  2**0
                  CONTENTS, READONLY
```

4-Symbol Tables

4.1 -app.o symbol table :

Resolved symbols : main , string_name
Unresolved symbols : Uart_send_String since its defines in uart.o , it will be resolved in linker phase .
All addresses are virtual addresses it will be located at linker by locator counter.

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_name
         U Uart_send_String
```

4.2 - uart.o symbol table :

Resolved symbols : Uart_send_String
The address is virtual address it will be located at linker by locator counter.

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ arm-none-eabi-nm.exe uart.o
00000000 T Uart_send_String
```

4.3 - startup.o symbol table :

Resolved symbols : .reset , stop

Unresolved symbols : main , reset ,stack_top , it will be resolved in linker phase .
All addresses are virtual addresses it will be located at linker by locator counter.

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ arm-none-eabi-nm.exe startup.o
00000000 t .reset
         U main
         U reset
         U stack_top
00000008 t stop
```

4.4 - learn_in_depth.elf symbol table :

All symbol are resolved .
All address are physical addresses since the locator of linker locate all layout of memory .

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ arm-none-eabi-nm.exe learn_in_depth.elf
00010010 T main
00010000 T reset
000110ed D stack_top
00010008 t stop
00010078 D string_name
00010028 T Uart_send_String
```

5-Execution of software

Here , we will see the output of our program in QEMU tool .

```
Dell@OsamaYoussef MINGW64 /e/Unit_2_lesson_2_lab1
$ ../qemu/qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn_in_depth.bin
Learn-in-depth : < Osama Youssef >
```

Thank you for your time