Lab1

ARM Versatile PB

Name: ***Aya Sayed Abdellah***

- ***In this lab:*** ***I have to create a bare-metal software to send a "Learn-in-depth < Aya Sayed > " using UART so I have to create its files that consists of :***
    - ***Application : App.c***
    - ***UART driver : UART.c & UART.h***

Because I will build it without GUI so accordding to compilation process sequence as fig.1 I need to download cross tool-chain to compiler my bare-metal software. also I need to create Startup.s to reset the CPU and initialize some rules before calling the main and need to create Linker Script file to link all files together.

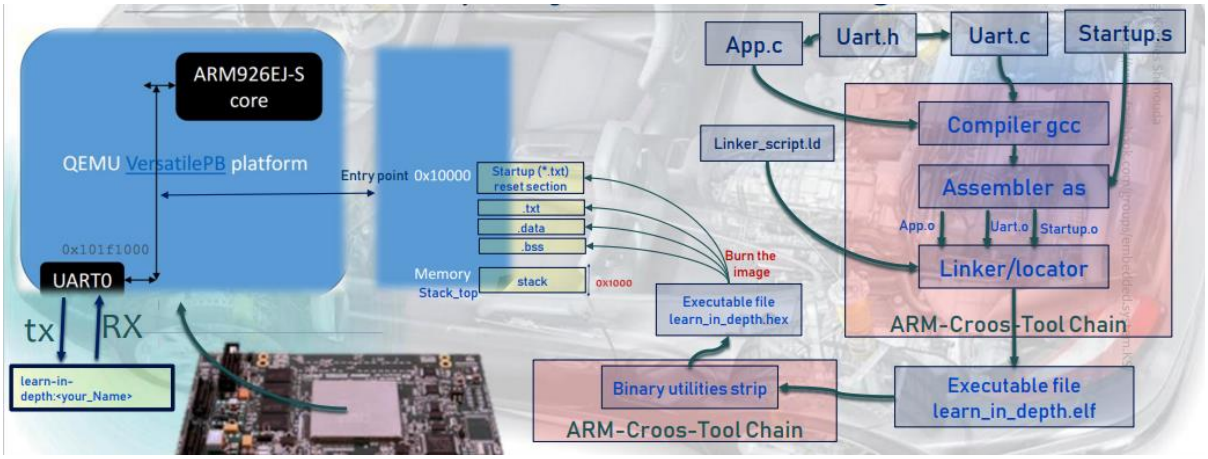


Fig1: implementation of the caming sequence

I will use Git to push my commands and Sublime as a terminal to write my bare-metal software So let's start …

I used touch file command to create my files (APP.c - UART.c - UART.h - Startup.s - Linker_Script.ld)

Then I wrote them and here they are…

APP.c | UART.c | UART.h

```c

#include "UART.h"

unsigned char string_buffer[50] = "Learn_in_depth < Aya Sayed >";
unsigned char sting;

int main()
{
    UART_SEND_DATA(string_buffer);
}
```

APP.c | UART.c | UART.h

```c
#ifndef _UART_H_
#define _UART_H_

void UART_SEND_DATA( unsigned char* p_tx_string );

#endif
```

APP.c | UART.c | UART.h

```c
#include "UART.h"

#define UARTDR *((volatile unsigned int* const)((unsigned int)0x101f1000))

void UART_SEND_DATA(unsigned char* p_tx_string)
{
    while ( *p_tx_string  != '\0' )
    {
        UARTDR = *p_tx_string;
        p_tx_string++;
    }
}
```

Startup.s | APP.c | UART.c | UART.h | Linker_Script.ld | output.map | APP.s

```asm
.global reset
reset:
        ldr  sp, = stack_top
        bl   main
stop:   b    stop
```

Startup.s | APP.c | UART.c | UART.h | Linker_Script.ld | output.map | APP.s

```
ENTRY(reset)

MEMORY
{
    RAM ( RWX ) : ORIGIN = 0x00000000,   LENGTH = 64M
}

SECTIONS
{
    . = 0x00010000;
    .startup . :
    {
      startup.o(.text)
    }> RAM
    .text :
    {
      *(.text) *(.rodata)
    }> RAM
    .data :
    {
      *(.data)
    }> RAM
    .bss :
    {
      *(.bss)  *(COMMON)
    }> RAM
    . = . + 0x00001000;          /*10k of stack memory*/
    stack_top = . ;
}
```

Line 27, Column 25 | Tab Size: 4 | Plain Text

then using Git I pushed my commands to :

- ***Use arm-none-eani-gcc.exe cross tool-chain to compile APP.c, UART.c, UART.h and Startup.s to get APP.o, UART.o, UART.o and Startup.o .***
- ***Linking all files with linker script file to get executable file (elf image)***
- ***Getting executable file.bin .***
- ***Getting some information about filess' sections in the memory.***
- ***Check address of the entry point .***

And also I used QEMU (quick emutator) to run my target on ARM versatilePB

```
MINGW32:/c/Users/diesel/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/H...

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ touch UART.c  UART.h APP.c Linker_Script.ld Startup.s

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
arm-none-eabi-addr2line.exe*  arm-none-eabi-gcc-ar.exe*      arm-none-eabi-ld.exe*
arm-none-eabi-ar.exe*         arm-none-eabi-gcc-nm.exe*      arm-none-eabi-nm.exe*
arm-none-eabi-as.exe*         arm-none-eabi-gcc-ranlib.exe*  arm-none-eabi-objcopy.exe*
arm-none-eabi-c++.exe*        arm-none-eabi-gcov.exe*        arm-none-eabi-objdump.exe*
arm-none-eabi-c++filt.exe*    arm-none-eabi-gcov-dump.exe*   arm-none-eabi-ranlib.exe*
arm-none-eabi-cpp.exe*        arm-none-eabi-gcov-tool.exe*   arm-none-eabi-readelf.exe*
arm-none-eabi-elfedit.exe*    arm-none-eabi-gdb.exe*         arm-none-eabi-size.exe*
arm-none-eabi-g++.exe*        arm-none-eabi-gdb-py.exe*      arm-none-eabi-strings.exe*
arm-none-eabi-gcc.exe*        arm-none-eabi-gprof.exe*       arm-none-eabi-strip.exe*
arm-none-eabi-gcc-7.2.1.exe*  arm-none-eabi-ld.bfd.exe*      gccvar.bat

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$  export PATH=../ARM/bin/:$PATH

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ arm-none-eabi-gcc.exe  -c -mcpu=arm926ej-s -I . APP.c -o APP.o

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$  arm-none-eabi-gcc.exe  -c -mcpu=arm926ej-s -I . UART.c -o UART.o

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ arm-none-eabi-gcc.exe  -c -mcpu=arm926ej-s -s Startup.s -o Startup.o
Startup.s: Assembler messages:
Startup.s: Warning: end of file not at end of a line; newline inserted

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ arm-none-eabi-ld.exe  -T  Linker_Script.ld -Map=output.map Startup.o  APP.o  UART.o  -o  Learn_in_depth.
elf

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ arm-none-eabi-objcopy.exe  -O binary Learn_in_depth.elf   Learn_in_depth.bin

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ ../qemu/qemu-system-arm -M versatilepb -m 128M -nographic  -kernel  Learn_in_depth.bin
Learn_in_depth < Aya Sayed >
```

Fig: showing my commands to build my target

```
diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 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         00000020  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000032  00000000  00000000  00000054  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000086  2**0
                  ALLOC
  3 .comment      0000007f  00000000  00000000  00000086  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  00000105  2**0
                  CONTENTS, READONLY
```

```
$ 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         00000054  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000000  00000000  00000000  00000088  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  00000088  2**0
                  ALLOC
  3 .comment      0000007f  00000000  00000000  00000088  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  00000107  2**0
                  CONTENTS, READONLY
```

```
$ 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
```

```
$ 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  00010000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00000074  00010010  00010010  00010010  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .data         00000032  00010084  00010084  00010084  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000100b6  2**0
                  CONTENTS, READONLY
  4 .comment      0000007e  00000000  00000000  000100e4  2**0
                  CONTENTS, READONLY
```

last pictures showing information about their sections in the momery like: size, VMA virtial memory addresses, LMA loading memory addresses, …).

all of them by using command : arm-none-eabi-objdump.exe

hence you can create a map file to see all this details about memory sections and its symbols

for the unsolving symbols use command : arm-none-eabi-nm.exe

```
$ arm-none-eabi-nm.exe  UART.o
00000000 T UART_SEND_DATA
```

```
$ arm-none-eabi-nm.exe  APP.o
00000000 T main
00000000 D string_buffer
         U UART_SEND_DATA
```

```
$ arm-none-eabi-nm.exe  Startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop
```

and the solving symbols are …

```
$ arm-none-eabi-nm.exe  Learn_in_depth.elf
00010010 T main
00010000 T reset
000110b6 D stack_top
00010008 t stop
00010084 D string_buffer
00010030 T UART_SEND_DATA
```

then i used command : arm-none-eabi-readelf.exe to check entry point.

```
diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ arm-none-eabi-readelf.exe -a  Learn_in_depth.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:          66404 (bytes into file)
  Flags:                             0x5000200, Version5 EABI, <unknown>
  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: 8

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 010000 000010 00  AX  0   0  4
  [ 2] .text             PROGBITS        00010010 010010 000074 00  AX  0   0  4
  [ 3] .data             PROGBITS        00010084 010084 000032 00  WA  0   0  4
  [ 4] .ARM.attributes   ARM_ATTRIBUTES  00000000 0100b6 00002e 00      0   0  1
  [ 5] .comment          PROGBITS        00000000 0100e4 00007e 01  MS  0   0  1
  [ 6] .symtab           SYMTAB          00000000 010164 000160 10      7  17  4
  [ 7] .strtab           STRTAB          00000000 0102c4 000055 00      0   0  1
  [ 8] .shstrtab         STRTAB          00000000 010319 000049 00      0   0  1
```

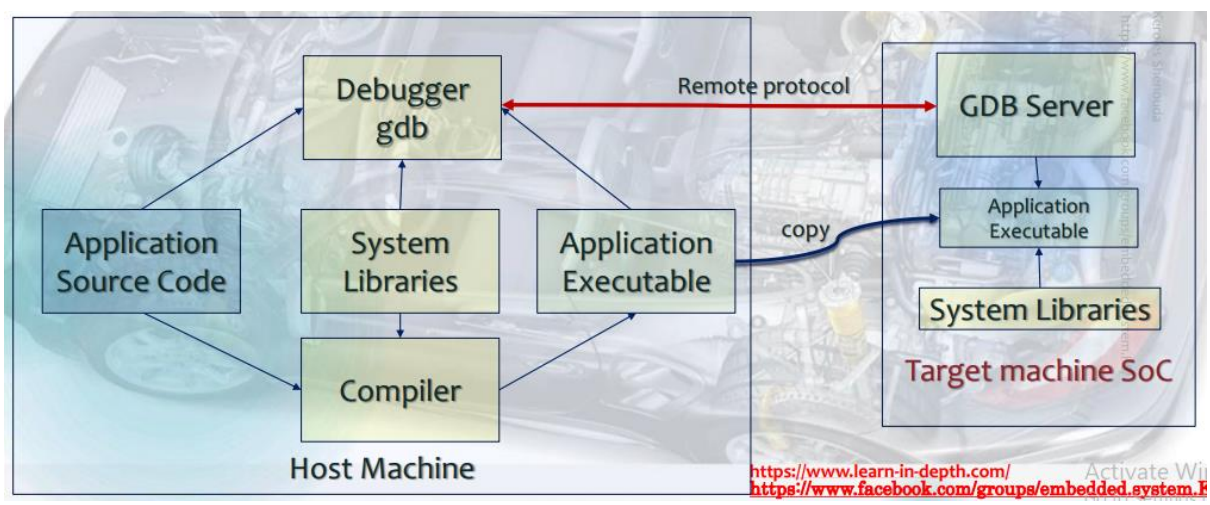
And this is the output.map showing in details the memory

```
Startup.s   APP.c   UART.c   UART.h   Linker_Script.ld   output.map   APP.s

Memory Configuration

Name             Origin             Length             Attributes
RAM              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       0x74
 *(.text)
 .text           0x00010010       0x20 APP.o
                 0x00010010                main
 .text           0x00010030       0x54 UART.o
                 0x00010030                UART_SEND_DATA
 *(.rodata)

.glue_7          0x00010084        0x0
 .glue_7         0x00010084        0x0 linker stubs

.glue_7t         0x00010084        0x0
 .glue_7t        0x00010084        0x0 linker stubs

.vfp11_veneer    0x00010084        0x0
 .vfp11_veneer   0x00010084        0x0 linker stubs

.v4_bx           0x00010084        0x0
 .v4_bx          0x00010084        0x0 linker stubs
```

```
 .glue_7         0x0001007c        0x0
  .glue_7        0x00000000        0x0 linker stubs

 .glue_7t        0x0001007c        0x0
  .glue_7t       0x00000000        0x0 linker stubs

 .vfp11_veneer   0x0001007c        0x0
  .vfp11_veneer  0x00000000        0x0 linker stubs

 .v4_bx          0x0001007c        0x0
  .v4_bx         0x00000000        0x0 linker stubs

 .iplt           0x0001007c        0x0
  .iplt          0x00000000        0x0 Startup.o

 .rel.dyn        0x0001007c        0x0
  .rel.iplt      0x00000000        0x0 Startup.o

 .data           0x0001007c       0x34
  *(.data)
  .data          0x0001007c        0x0 Startup.o
  .data          0x0001007c       0x34 APP.o
                 0x0001007c                string_buffer
  .data          0x000100b0        0x0 UART.o

 .igot.plt       0x000100b0        0x0
  .igot.plt      0x00000000        0x0 Startup.o

 .bss            0x000100b0        0x1
  *(.bss)
  .bss           0x000100b0        0x0 Startup.o
  .bss           0x000100b0        0x0 APP.o
  .bss           0x000100b0        0x0 UART.o
  *(COMMON)
  COMMON         0x000100b0        0x1 APP.o
                 0x000100b0                sting
                 0x000110b1                . = (. + 0x1000)
                 0x000110b1                stack_top = .
LOAD APP.o
LOAD UART.o
LOAD Startup.o
OUTPUT(Learn_in_depth.elf elf32-littlearm)

 .ARM.attributes
                 0x00000000       0x2e
  .ARM.attributes
                 0x00000000       0x22 Startup.o
  .ARM.attributes
                 0x00000022       0x32 APP.o
  .ARM.attributes
                 0x00000054       0x32 UART.o

 .comment        0x00000000       0x11
  .comment       0x00000000       0x11 APP.o
                                  0x12 (size before relaxing)
  .comment       0x00000000       0x12 UART.o

 .debug_line     0x00000000       0xac
  .debug_line    0x00000000       0x3a Startup.o
  .debug_line    0x0000003a       0x35 APP.o
  .debug_line    0x0000006f       0x3d UART.o
```

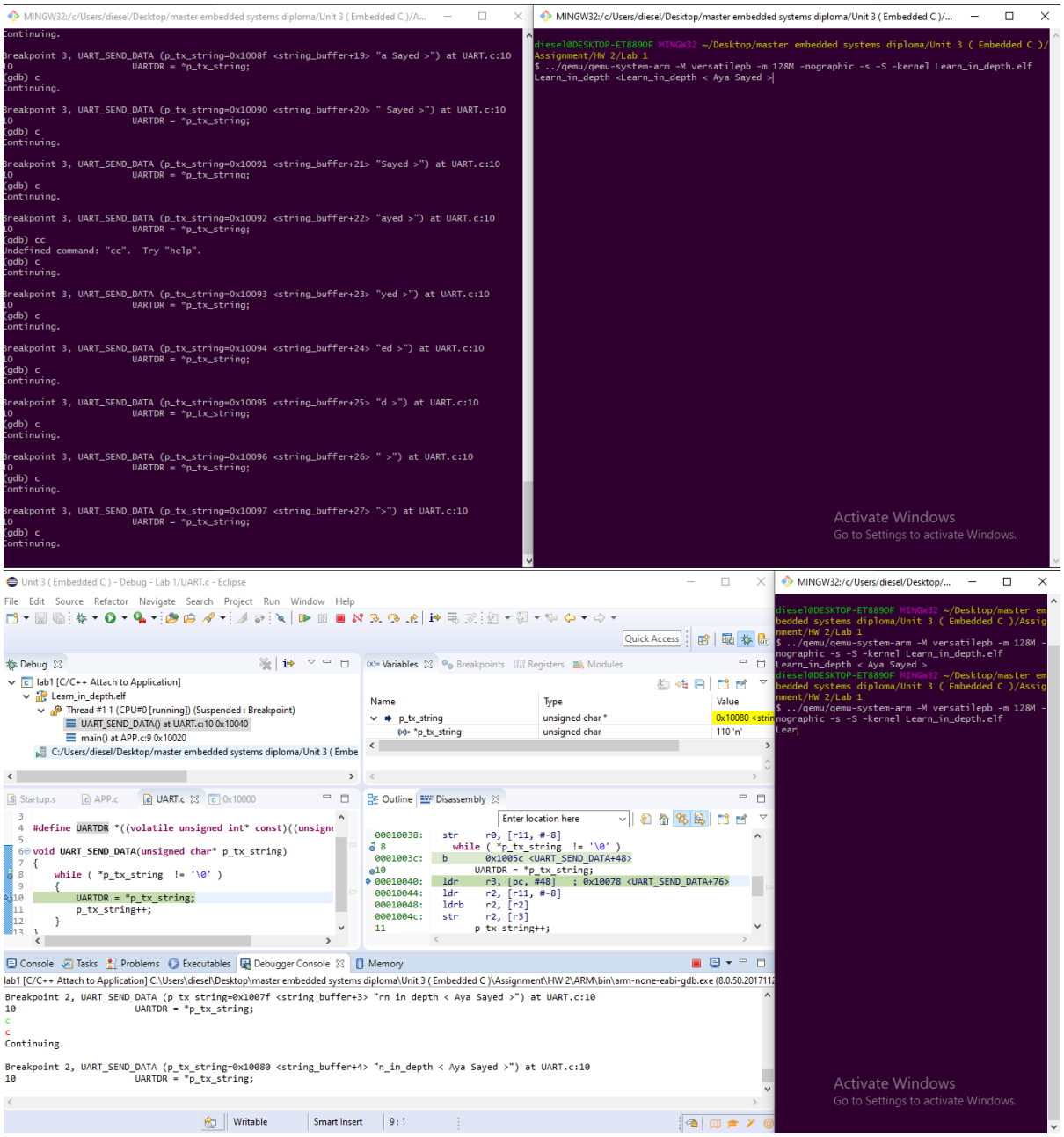
```
APP.c    output.map    UART.c    UART.h

 .debug_info     0x00000000      0x17a
  .debug_info    0x00000000       0x96 Startup.o
  .debug_info    0x00000096       0x88 APP.o
  .debug_info    0x0000011e       0x5c UART.o

 .debug_abbrev   0x00000000       0xca
  .debug_abbrev  0x00000000       0x14 Startup.o
  .debug_abbrev  0x00000014       0x65 APP.o
  .debug_abbrev  0x00000079       0x51 UART.o

 .debug_aranges  0x00000000       0x60
  .debug_aranges
                 0x00000000       0x20 Startup.o
  .debug_aranges
                 0x00000020       0x20 APP.o
  .debug_aranges
                 0x00000040       0x20 UART.o

 .debug_loc      0x00000000       0x58
  .debug_loc     0x00000000       0x2c APP.o
  .debug_loc     0x0000002c       0x2c UART.o

 .debug_str      0x00000000       0xc8
  .debug_str     0x00000000       0xa6 APP.o
  .debug_str     0x000000a6       0x22 UART.o
                                  0xa0 (size before relaxing)

 .debug_frame    0x00000000       0x54
  .debug_frame   0x00000000       0x2c APP.o
  .debug_frame   0x0000002c       0x28 UART.o
```

now I need to debug my result using GDB accordding to the following concept



there are 2 way to debug it :

- ***Using GDB with Git.***
- ***Using GDB with GUI (Eclipse).***



And finally I make a Makefile to make my build easier than before …

MINGW32:/c/Users/diesel/Desktop/master embedded systems diploma/Unit 3 (Embedded C)/Assignment/H...

```
diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ make
arm-none-eabi-gcc.exe  -c -mcpu=arm926ej-s -g -s Startup.s -o Startup.o
Startup.s: Assembler messages:
Startup.s: Warning: end of file not at end of a line; newline inserted
arm-none-eabi-gcc.exe  -c -mcpu=arm926ej-s -g -I . APP.c -o APP.o
arm-none-eabi-gcc.exe  -c -mcpu=arm926ej-s -g -I . UART.c -o UART.o
arm-none-eabi-ld.exe   -T  Linker_Script.ld  -Map=output.map APP.o UART.o Startup.o -o  Learn_in_depth.elf

arm-none-eabi-objcopy.exe  -O binary Learn_in_depth.elf Learn_in_depth.bin
C:\WinAVR-20100110\utils\bin\make.exe: Interrupt/Exception caught (code = 0xc00000fd, addr = 0x4217b3)

diesel@DESKTOP-ET889OF MINGW32 ~/Desktop/master embedded systems diploma/Unit 3 ( Embedded C )/Assignment/
HW 2/Lab 1
$ ../qemu/qemu-system-arm -M versatilepb -m 128M -nographic -kernel Learn_in_depth.bin
Learn_in_depth < Aya Sayed >
```

C:\Users\diesel\Desktop\master embedded systems diploma\Unit 3 (Embedded C)\Assignment\HW 3\Lab (startup.s)\...

File Edit Selection Find View Goto Tools Project Preferences Help

Startup.s | output.map | Makefile | main.c | Linker_Script.ld

```
#Author: Aya Sayed
#Master Embedded System Diploma <Learn in depth>

CC = arm-none-eabi-
INCS = -I .
LIBS =
CFLAGS = -mcpu=cortex-m3 -gdwarf-2
C = $(wildcard *.c)
COBJ = $(C:.c=.o)
AS = $(wildcard *.s)
ASOBJ = $(AS:.s=.o)
Project_Name = LAB1_ARM_CORTEX-M3


all: $(Project_Name).bin
	@echo "------------------Build is done------------------"

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

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

$(Project_Name).elf: $(ASOBJ) $(COBJ)
	$(CC)ld.exe   -T  Linker_Script.ld $(LIBS) -Map=output.map $(COBJ) $(ASOBJ) -

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

clean_all:
	rm *.o *.elf *.bin
	@echo "Everything clean"

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

Line 1, Column 1 | Tab Size: 4 | Makefile