4/23/2023

Lab1

Embedded C Lesson 3

Mostafa Mohamed Edrees
LEARN-IN-DEPTH

Lab1

Required:

You have to create a bare metal Software to send a “learn-in-depth :< Your_Name >” using UART.

Physical Board:

VersatilePB

Processor:

Arm926ej-s

With debug information & Makefile.

Name:

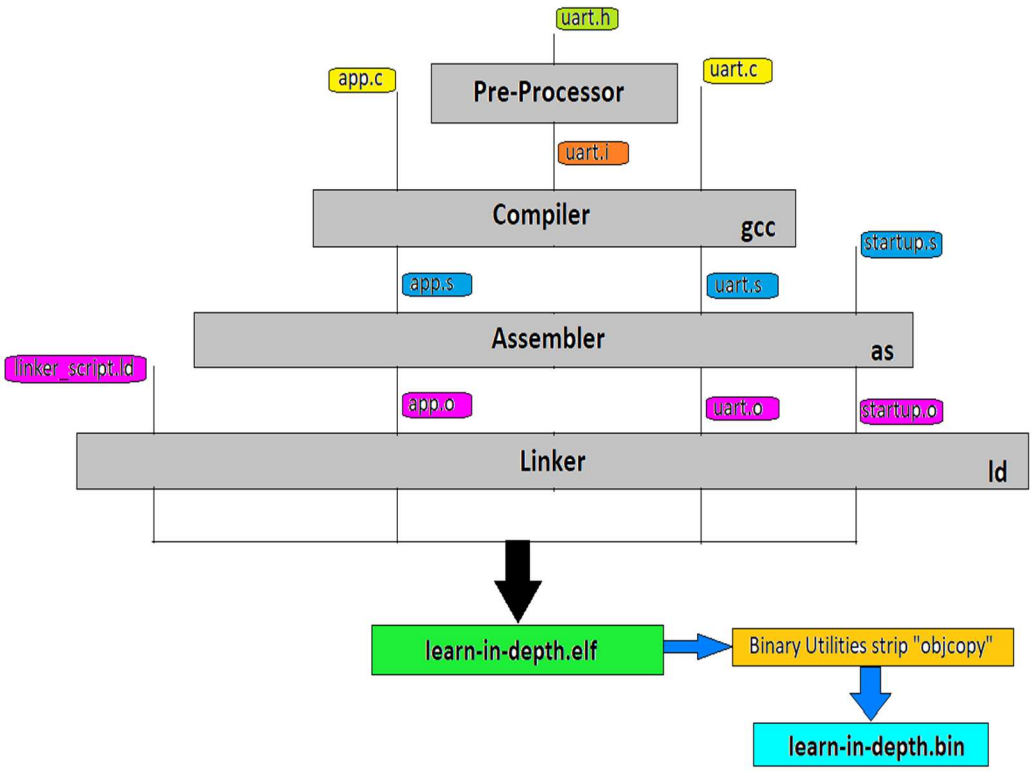
Mostafa Mohamed Edrees

Supervisor:

Eng. Keroles Shenouda

Steps:

- Create C code files. >> app.c , uart.c , uart.h
- Write Startup code file. >> startup.s
- Write Linker Script file. >> linker_script.ld
- Write Makefile to automate build.
- Build the program by Makefile.
- Run the program in the QEMU Simulator.
- Debug the program using gdb.
- Debug the program using eclipse.



Create C code files:

Uart.c

```c
/**
 ******************************************************************************
 * @file           : uart.c
 * @author         : Mostafa Edrees
 * @brief          : lab1 in lesson2 in Embedded C
 * @date           : 17/4/2023
 * @board          : versatilePB physical board
 ******************************************************************************
**/

#include "uart.h"
#include "Platform_types.h"

// Base Address of UART0:       0x101f1000
// Offset of Data Register(DR): 0x0
#define UART0DR  *((vusint32_t * const)((usint32_t *)0x101f1000))

//P_tx_String >> Pointer to transmiting string
void UART0_Send_String(usint8_t * P_tx_String)
{
    while(*P_tx_String != '\0') //loop to print all characters of the string
    {
        UART0DR = *P_tx_String; //send string to UART0 byte by byte
        P_tx_String++; //next character
    }
}
```

Uart.h

```c
/**
 ******************************************************************************
 * @file           : uart.h
 * @author         : Mostafa Edrees
 * @brief          : lab1 in lesson2 in Embedded C
 * @date           : 17/4/2023
 * @board          : versatilePB physical board
 ******************************************************************************
**/

#ifndef _UART_H_
#define _UART_H_

#include "Platform_types.h"


//UART0 API
void UART0_Send_String(usint8_t * P_tx_String);


#endif
```

App.c

```c
/**
 ******************************************************************************
 * @file           : app.c
 * @author         : Mostafa Edrees
 * @brief          : lab1 in lesson2 in Embedded C
 * @date           : 17/4/2023
 * @board          : versatilePB physical board
 ******************************************************************************
**/

#include "uart.h"
#include "Platform_types.h"

//String that will send to UART0
usint8_t String_Buffur[100] = "learn-in-depth:<Mostafa Mohamed Edrees>";

void main(void)
{
    UART0_Send_String(String_Buffur);

}
```

Write Startup code file:

startup.s:

```
/**
 ******************************************************************************
 * @file            : startup.s
 * @author          : Mostafa Edrees
 * @brief           : lab1 in lesson2 in Embedded C
 * @date            : 17/4/2023
 * @board           : versatilePB physical board
 ******************************************************************************
**/

.globl reset

reset:
    ldr sp, =0x00011000      //before linker
    bl main

stop:
    b stop
```

```
   4:   4728203a            ; <UNDEFINED> instruction: 0x4728203a
   8:   2029554e    eorcs   r5, r9, lr, asr #10
   c:   2e372e34    mrccs   14, 1, r2, cr7, cr4, {1}
  10:   Address 0x00000010 is out of bounds.


Disassembly of section .ARM.attributes:

00000000 <.ARM.attributes>:
```

Write Linker Script file:

Linker_script.ld

```
/**
 ******************************************************************************
 * @file            : linker_script.ld
 * @author          : Mostafa Edrees
 * @brief           : lab1 in lesson2 in Embedded C
 * @date            : 17/4/2023
 * @board           : versatilePB physical board
 ******************************************************************************
**/

ENTRY(reset)

MEMORY
{
    Mem (rwx) : ORIGIN = 0x00000000 , LENGTH = 64M
}

SECTIONS
{
    . = 0x10000;

    .startup . :
    {
        startup.o(.text)
    }> Mem
    .text :
    {
        *(.text)
    }> Mem
    .date
    {
        *(.date)
    }> Mem
    .bss
    {
        *(.bss)
    }> Mem

    . = . + 0x1000; /* 1000 >> 4KB for stack */
    stack_top = .;
}
```

After linker_script the addresses will be physical with SOC

We put address of stack_top at startup.s to put this address in stack pointer register (PC).

Startup.s

```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 2/Lab1 (master)
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop
```

```
.globl reset

reset:
    ldr sp, =stack_top  //after linker
    bl main

stop:
    b stop
```

Write Makefile to automate build:

```
CC = arm-none-eabi-
CFLAGS = -g -mcpu=$(Arm_Processor)
INCS = -I .
LIBS =
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
AsOBJ = $(As:.s=.o)


Project_Name = learn-in-depth
Copyrights = Mostafa Edrees
date = 23/4/2023
Board = VersatilePB
Arm_Processor = arm926ej-s


all: learn-in-depth.bin
    @echo -e "\n*******************************"
    @echo -e "\tBuild is Done"
    @echo -e "Project Name:" $(Project_Name)
    @echo -e "Copyrights:" $(Copyrights)
    @echo -e "date:" $(date)
    @echo -e "Board:" $(Board)
    @echo -e "Arm Processor:" $(Arm_Processor)
    @echo -e "*******************************\n"


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 $(OBJ) $(AsOBJ) -o $@


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



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

clean:
    rm *.elf .bin
```

Build the program by Makefile:

```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab1 (master)
$ mingw32-make.exe all
arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . app.c -o app.o
arm-none-eabi-gcc.exe -c -g -mcpu=arm926ej-s -I . uart.c -o uart.o
arm-none-eabi-as.exe -g -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
arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o startup.o -o learn-in-dept
h.elf
arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin

********************************
        Build is Done
Project Name: learn-in-depth
Copyrights: Mostafa Edrees
date: 23/4/2023
Board: VersatilePB
Arm Processor: arm926ej-s
********************************
```

Run the program in the QEMU Simulator:

```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab1 (master)
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn-in-depth.bin
learn-in-depth<Mostafa Edrees>
```

Debug the program using gdb:

When we open the debug.

```
MINGW32:/d/Mastering Embedded System/GitHub_Repo/Mastering_Embedded_System_Online_Diploma/Embedded C/Lesson 3/L...
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab1 (master)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
```

```
MINGW32:/d/Mastering Embedded System/GitHub_Repo/Mastering_Embedded_System_Online_Diploma/Embedded C/Lesson 3/...
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab1 (master)
$ arm-none-eabi-gdb.exe learn-in-depth.elf
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:\Mastering Embedded System\GitHub_Repo\Mastering_Embedded_System_
Online_Diploma\Embedded C\Lesson 3\Lab1\learn-in-depth.elf...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:15
15              ldr sp, =stack_top
(gdb)
```

using “l” to print the code of file that we in it:

```
(gdb) l
10
11
12          .globl reset
13
14          reset:
15                  ldr sp, =stack_top
16                  bl main
17
18          stop:
19                  b stop(gdb)
```

using display/3i $pc

```
19              b stop(gdb) display/3i $pc
1: x/3i $pc
=> 0x10000 <reset>:     ldr     sp, [pc, #4]    ; 0x1000c <stop+4>
   0x10004 <reset+4>:   bl      0x10010 <main>
   0x10008 <stop>:      b       0x10008 <stop>
(gdb)
```

breakpoint at “`UART0DR = *P_tx_String;`”

```
(gdb) s
22                      UART0DR = *P_tx_String; //send string character by character
1: x/3i $pc
=> 0x10044 <UART0_Send_String+20>:
    ldr r3, [pc, #48]    ; 0x1007c <UART0_Send_String+76>
   0x10048 <UART0_Send_String+24>:      ldr     r2, [r11, #-8]
   0x1004c <UART0_Send_String+28>:      ldrb    r2, [r2]
(gdb) b uart.c:22
Breakpoint 1 at 0x10044: file uart.c, line 22.
```

The string will printed character by character and we use “c” to pause the breakpoint.

```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_
Embedded_System_Online_Diploma/Embedded C/Lesson 3/Lab1 (master)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
le
```

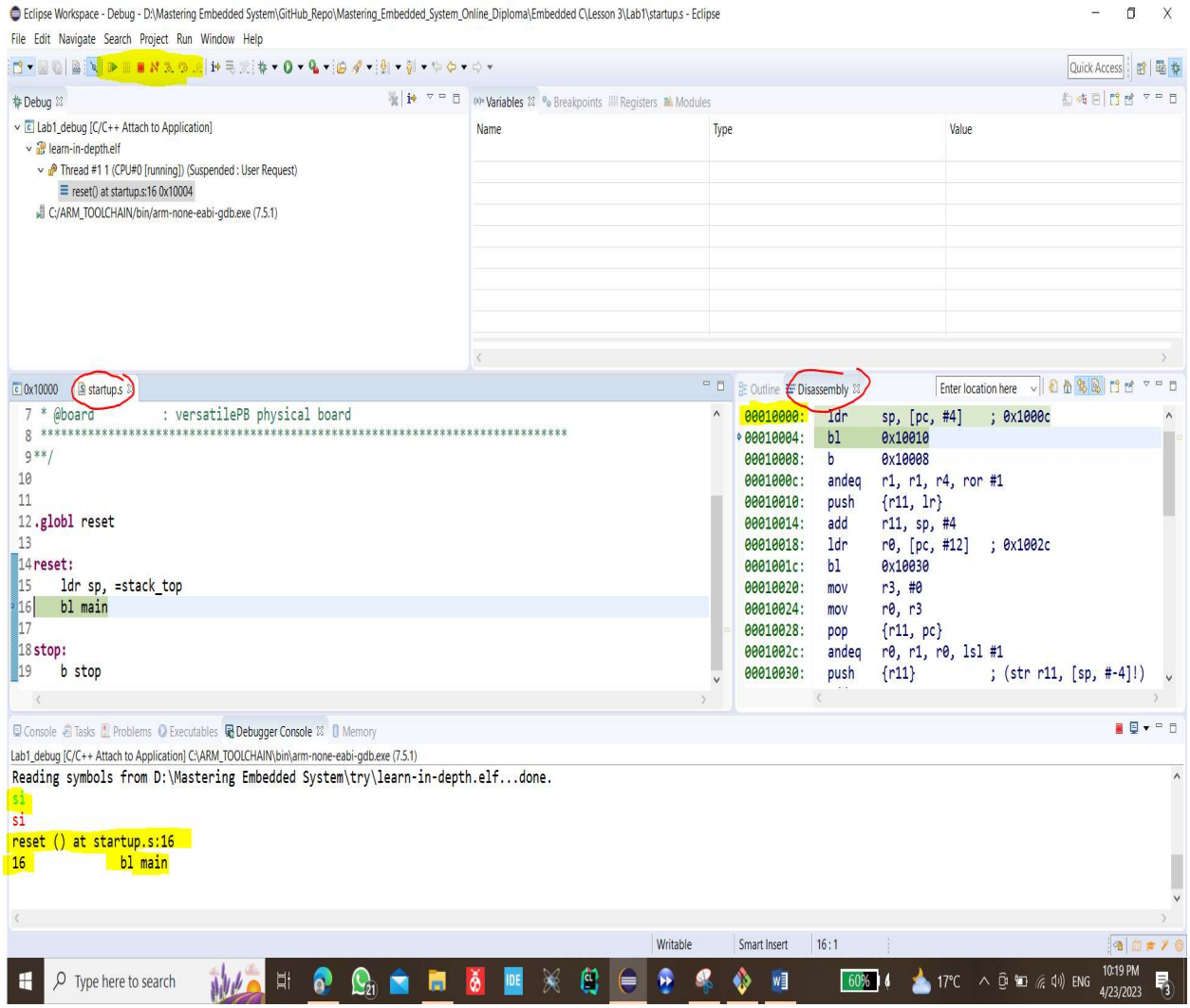
```
MINGW32:/d/Mastering Embedded System/GitHub_Repo/Mastering_Embedded_System_Online_Diploma/Embedded C/Lesson 3/L...
(gdb) c
Continuing.

Breakpoint 1, UART0_Send_String (
    P_tx_String=0x10081 <String_Buffer+1> "earn-in-depth<Mostafa Edrees>")
    at uart.c:22
22                      UART0DR = *P_tx_String; //send string character by character
1: x/3i $pc
=> 0x10044 <UART0_Send_String+20>:
    ldr r3, [pc, #48]    ; 0x1007c <UART0_Send_String+76>
   0x10048 <UART0_Send_String+24>:      ldr     r2, [r11, #-8]
   0x1004c <UART0_Send_String+28>:      ldrb    r2, [r2]
(gdb) c
Continuing.

Breakpoint 1, UART0_Send_String (
    P_tx_String=0x10082 <String_Buffer+2> "arn-in-depth<Mostafa Edrees>")
    at uart.c:22
22                      UART0DR = *P_tx_String; //send string character by character
1: x/3i $pc
=> 0x10044 <UART0_Send_String+20>:
    ldr r3, [pc, #48]    ; 0x1007c <UART0_Send_String+76>
   0x10048 <UART0_Send_String+24>:      ldr     r2, [r11, #-8]
   0x1004c <UART0_Send_String+28>:      ldrb    r2, [r2]
```

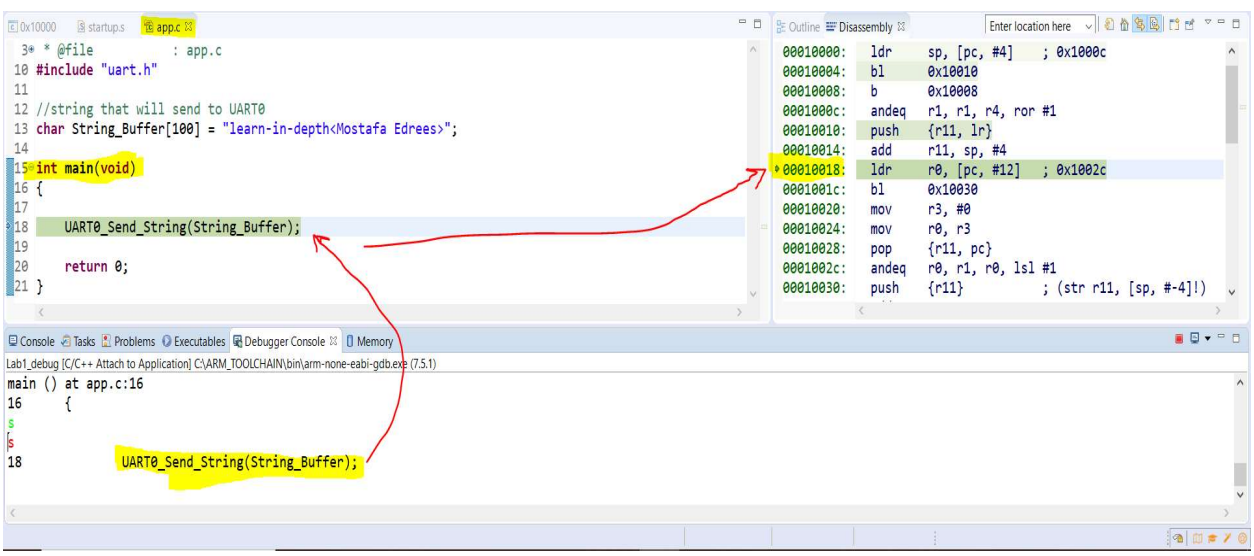
And we complete to print all characters of the string.

There are anthor commands like:

si , s , c , watch , print , where , info breakpoints , …….

Debug the program using eclipse:





```
//Data Register >> offset = 0x0
#define UART0DR *((volatile unsigned int * const)((unsigned int *)0x101f1000))


//P_tx_String >> pointer to transimited string
void UART0_Send_String(char *P_tx_String)
{
    while(*P_tx_String != '\0') //loop to print all characters of the string
    {
        UART0DR = *P_tx_String; //send string character by character
        P_tx_String++; //next character
    }
}
```

```
00010030:   push    {r11}           ; (str r11, [sp, #-4]!)
00010034:   add     r11, sp, #0
00010038:   sub     sp, sp, #12
0001003c:   str     r0, [r11, #-8]
00010040:   b       0x10060
00010044:   ldr     r3, [pc, #48]   ; 0x1007c
00010048:   ldr     r2, [r11, #-8]
0001004c:   ldrb    r2, [r2]
00010050:   str     r2, [r3]
00010054:   ldr     r3, [r11, #-8]
00010058:   add     r3, r3, #1
0001005c:   str     r3, [r11, #-8]
00010060:   ldr     r3, [r11, #-8]
```

```
Lab1_debug [C/C++ Attach to Application] C:\ARM_TOOLCHAIN\bin\arm-none-eabi-gdb.exe (7.5.1)
s
s
22                  UART0DR = *P_tx_String; //send string character by character
b uart.c:22
Breakpoint 1 at 0x10044: file uart.c, line 22.
```

```
lenovo@MostafaEdrees MINGW32 /d/Mastering Embedded System/GitHub_Repo/Mastering_Embedded_
System_Online_Diploma/Embedded C/Lesson 3/Lab1 (master)
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.elf
le
```