Mastering Embedded Systems online diploma

Learn in depth

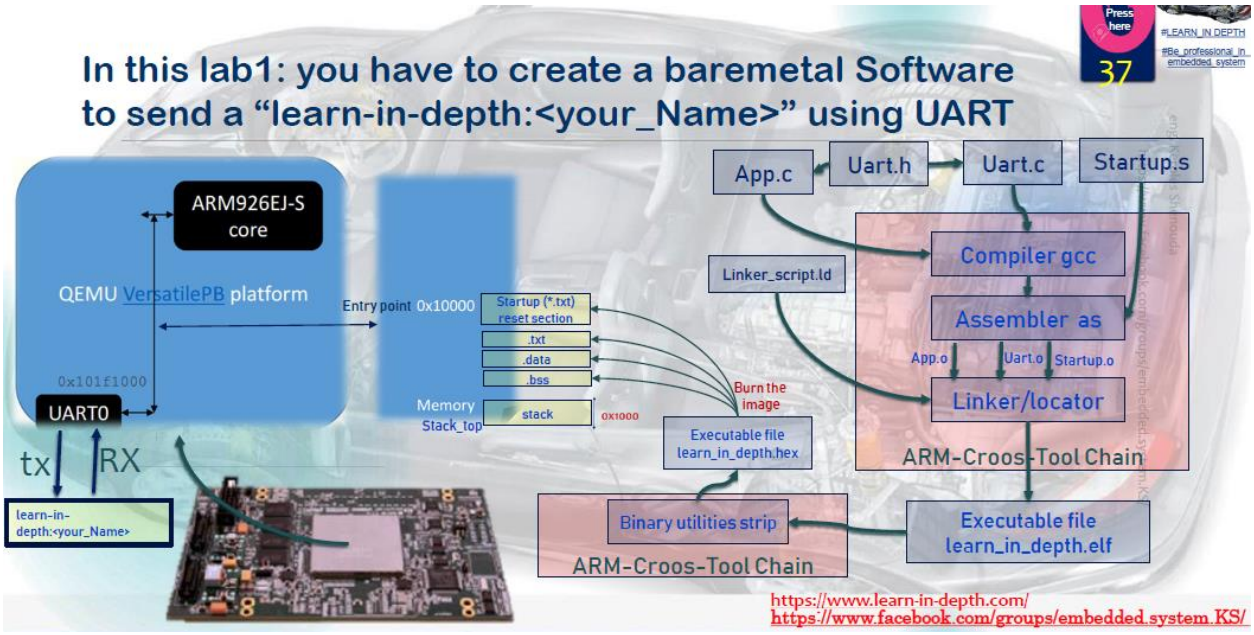
Under supervisor of Eng Keroles Shenouda

Unit 3 Lesson 2

Lab 1

Name: Kyrillos Phelopos Sawiris

Undergraduate computer and systems engineering Ain Shams university


In this lab1: you have to create a baremetal Software to send a "learn-in-depth:<your_Name>" using UART
Press here
37
#LEARN_IN DEPTH
#Be_professional_in_embedded_system
ARM926EJ-S core
QEMU VersatilePB platform
0x101f1000
UART0
tx
RX
learn-in-depth:<your_Name>
Entry point 0x10000
Startup (*.txt) reset section
.txt
.data
.bss
Memory Stack_top
stack
0x1000
Burn the image
Executable file learn_in_depth.hex
App.c
Uart.h
Uart.c
Startup.s
Linker_script.ld
Compiler gcc
Assembler as
App.o
Uart.o
Startup.o
Linker/locator
ARM-Croos-Tool Chain
Binary utilities strip
ARM-Croos-Tool Chain
Executable file learn_in_depth.elf
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/


In this lab you will learn
FOLLOW US
Press here
36
#LEARN_IN DEPTH
#Be_professional_in_embedded_system
You will need
Startup.s
Cross Toolchain
Linker Script
Makefile
C Code files
You will Learn through this lab. Startup, linker, location counter, linker script symbols, Makefile, GDB commands
Binary utilities: Objdump, strip, addr2line, size, readelf
test.c
test.ld
startup.s
Exectuable File Test.bin
Lab1 on ARM VersatilePB
https://www.learn-in-depth.com/
https://www.facebook.com/groups/embedded.system.KS/

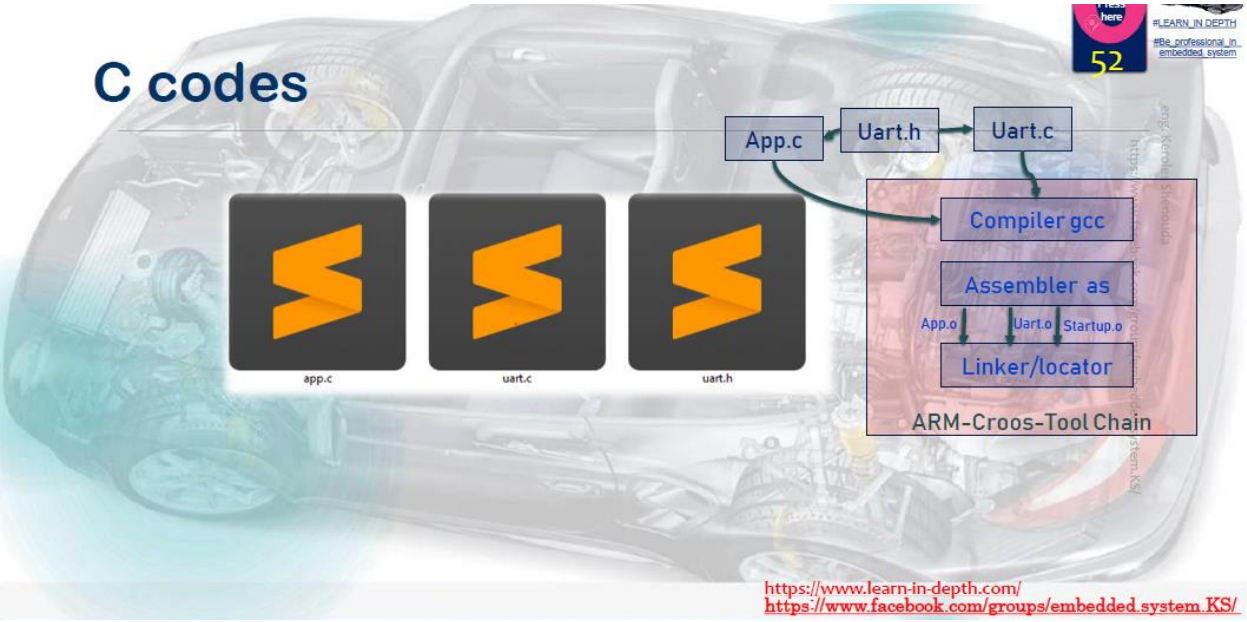
1) Writing C code Files

We will work with (arm926ej-s) core which is supported by QEMU

Its Entry point at 0x 10000

Our project is to send “Learn-in-depth:Kyrillos”

Through UART0 which mapped at 0x101f1000



1.1) Uart.h

uart.h

```
#ifndef _UART0_H
#define _UART0_H

void UART0_Send_String(unsigned char * P_TX_Str);

#endif
```

1.2) Uart.c

uart.h uart.c

```
#include "uart.h"

#define UART0DR *((volatile unsigned int *)((unsigned int *)0x101f1000))

void UART0_Send_String(unsigned char * P_TX_Str)
{
while (* P_TX_Str != '\0')
{
UART0DR =  (unsigned int) * P_TX_Str;
P_TX_Str++;
}
}
```

1.3) App.c

uart.h uart.c app.c

```
#include "uart.h"

unsigned char string_buff [100] = "learn-in-depth:kyrillos";

void main(void)
{

    UART0_Send_String(string_buff);
}
```

2) Let us generate (app/uart).o objects files

Using GNU ARM-Cross-toolchain“arm-none-eabi-gcc.exe”

Commands

```
arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o
```

```
arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s uart.c -o uart.o
```

```
arm-none-eabi-gcc.exe: error: arm926ej-s: No such file or directory

F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 2\Lab 1>arm-none-eabi-gcc.exe
 -c -g -I . -mcpu=arm926ej-s app.c -o app.o

F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 2\Lab 1>arm-none-eabi-gcc.exe
 -c -g -I . -mcpu=arm926ej-s uart.c -o uart.o

F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 2\Lab 1>ls *.o
app.o  uart.o

F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 2\Lab 1>
```

3) Navigate the .objfiles (relocatableimages)

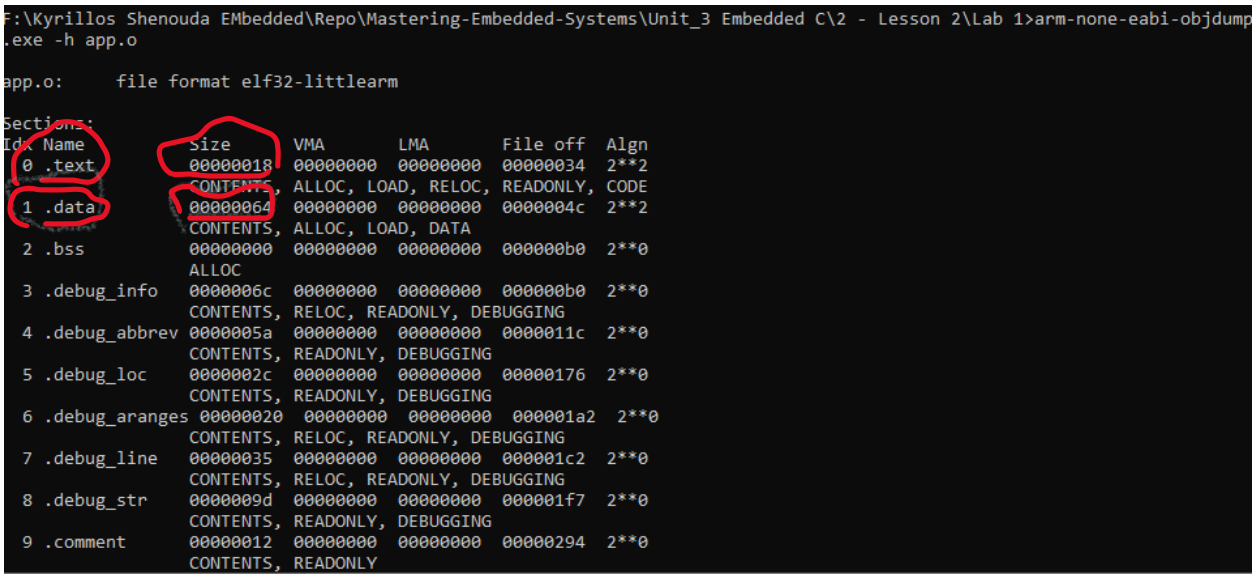
```
F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 2\Lab 1>arm-none-eabi-objdump
.exe --help
Usage: arm-none-eabi-objdump.exe <option(s)> <file(s)>
 Display information from object <file(s)>.
 At least one of the following switches must be given:
  -a, --archive-headers    Display archive header information
  -f, --file-headers       Display the contents of the overall file header
  -p, --private-headers    Display object format specific file header contents
  -P, --private=OPT,OPT... Display object format specific contents
  -h, --[section-]headers  Display the contents of the section headers
  -x, --all-headers        Display the contents of all headers
  -d, --disassemble        Display assembler contents of executable sections
  -D, --disassemble-all    Display assembler contents of all sections
  -S, --source             Intermix source code with disassembly
  -s, --full-contents      Display the full contents of all sections requested
  -g, --debugging          Display debug information in object file
  -e, --debugging-tags     Display debug information using ctags style
  -G, --stabs              Display (in raw form) any STABS info in the file
```

Use -h to show content of section headers

-D to display assembly contents of all sections

command

arm-none-eabi-objdump.exe -h app.o



```
F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 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    0000009d  00000000  00000000  000001f7  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  00000294  2**0
                  CONTENTS, READONLY
```

.text size (size of code) is 0x18

24 in decimal

24/4 = 6 instructions

.data size = 0x64

100 in decimal which is the size of the array we defined

generate the disassembly file from the bin

arm-none-eabi-objdump.exe -D app.o> app.s

app.s

```
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 <UART0_Send_String>
  10:   e8bd8800    pop {fp, pc}
  14:   00000000    andeq   r0, r0, r0

Disassembly of section .data:

00000000 <string_buff>:
   0:   7261656c    rsbvc   r6, r1, #108, 10    ; 0x1b000000
```

R programming language | length : 7,482 lines : 212 | Ln : 1 Col : 1 Pos : 1 | Windows (CR LF) | UTF-8

Uart.s

uart.h | uart.c | app.c | app.s | uart.s

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


Disassembly of section .text:

00000000 <UART0_Send_String>:
   0:   e52db004    push    {fp}        ; (str fp, [sp, #-4]!)
   4:   e28db000    add fp, sp, #0
   8:   e24dd00c    sub sp, sp, #12
   c:   e50b0008    str r0, [fp, #-8]
  10:   ea000006    b   30 <UART0_Send_String+0x30>
  14:   e59f3030    ldr r3, [pc, #48]   ; 4c <UART0_Send_String+0x4c>
  18:   e51b2008    ldr r2, [fp, #-8]
  1c:   e5d22000    ldrb    r2, [r2]
  20:   e5832000    str r2, [r3]
  24:   e51b3008    ldr r3, [fp, #-8]
  28:   e2833001    add r3, r3, #1
```

R programming language | length : 7,570 lines : 211 | Ln : 11 Col : 38 Pos : 261 | Windows (CR LF) | UTF-8

4) writing startup code file

1.Disable all INT

2.Define Interrupt vectors Section

3.InitMemory & Hardware

4.Copy Data from ROM RAM

5.Initialize Data Area

6.Initialize StacK

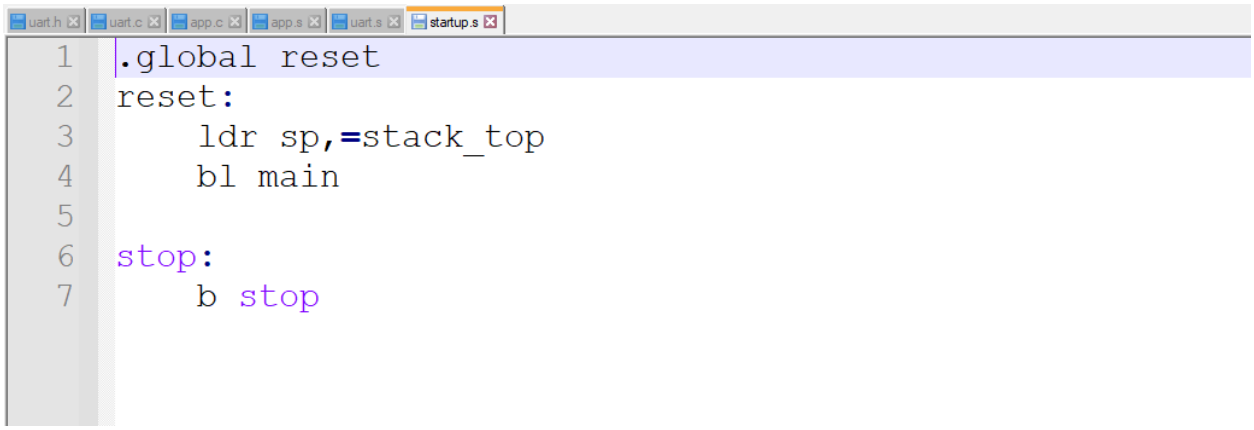
7.Enable interrupts.

8.Create a reset section and Call main( ).

In Lab1: We will write a simple startup:

1.Create a reset section and Call main( ).

2.Initialize StacK



Compile it

```
F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 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
```

5) writing linker script

The Linker Script is a text file made up of a series of Linker directives which tell the Linker where the available memory is and how it should be used.

Linker script code

uart.h | uart.c | app.c | app.s | uart.s | startup.s | linker_script.ld

```
2  ENTRY(reset)
3
4  MEMORY
5  {
6  mem(rwx): ORIGIN = 0x00000000 , LENGTH = 64M
7  }
8
9  SECTIONS
10 {
11     . =0x10000;
12     .startup . :
13     {
14     startup.o(.text)
15     }>mem
16
17     .text :
18     {
19     *(.text)
20     }>mem
21
22     .data :
23     {
24     *(data)
25     }>mem
26
27     .bss :
28     {
29     *(.bss)
30     }>mem
31
32     . = . + 0x1000;
33     stack_top = .;
34 }
```

- **To link .o files together in .elf file use the following command**

```
arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o startup.o -o learn-in-depth.elf
```

- **to show the details of learn-in-depth.elf file use**

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 .ARM.attributes 0000002e  00000000  00000000  000080dc  2**0
                  CONTENTS, READONLY
  3 .comment      00000011  00000000  00000000  0000810a  2**0
                  CONTENTS, READONLY
  4 .data         00000064  00010078  00010078  00008078  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .debug_info   000000c8  00000000  00000000  0000811b  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_abbrev 000000ab  00000000  00000000  000081e3  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_loc    00000058  00000000  00000000  0000828e  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_aranges 00000040  00000000  00000000  000082e6  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .debug_line   00000072  00000000  00000000  00008326  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .debug_str    000000bf  00000000  00000000  00008398  2**0
                  CONTENTS, READONLY, DEBUGGING
 11 .debug_frame  00000054  00000000  00000000  00008458  2**2
                  CONTENTS, READONLY, DEBUGGING
```

- **To read the symbols**

arm-none-eabi-nm.exe learn-in-depth.elf

```
F:\Kyrillos Shenouda EMbedded\Repo\Mastering-Embedded-Systems\Unit_3 Embedded C\2 - Lesson 2\Lab 1>arm-none-eabi-nm.exe
learn-in-depth.elf
00010010 T main
00010000 T reset
000110dc T stack_top
00010008 t stop
00010078 D string_buff
00010028 T UART0_Send_String
```

Let us now to linking all the objects and generate the map file

arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o startup.o -o learn-in-depth.elf -Map=map_file.map

The .mapfile gives a complete listing of all code and data addresses for the final software image.

6) Compilation process to generate binary file

Command

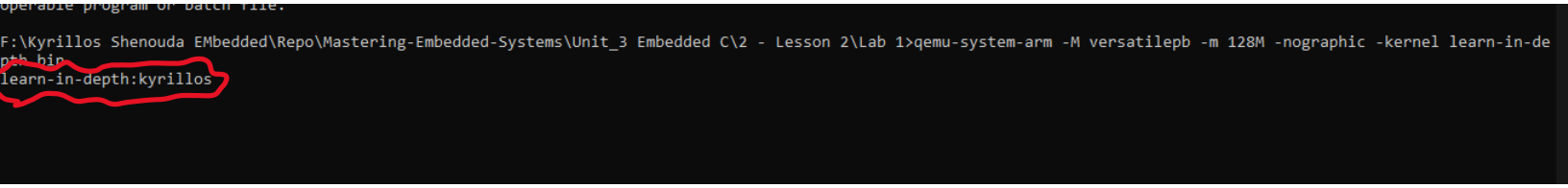
```
arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin
```

7) run the program in the QEMU Simulator

command

qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn-in-depth.bin

8) Output



Finished Successfully (^_^)