Printing (Learn-in-Depth:Hassan Attia) Using UART

1-app.c,uart.c,uart.h:

app.c

```c
#include "uart.h"

unsigned char string_buffer1 [100] = "Learn-in-Depth: Hassan Attia";
const unsigned char string_buffer2 [100] =  "Learn-in-Depth: Hassan Attia";



void main (void){


uart_send_string (string_buffer1);


}
```

C source file | length : 239 lines : 14 | Ln : 14 Col : 2 Pos : 240 | Windows (CR LF) | UTF-8 | INS

[redacted]\Downloads\Embedded\labs\LAB_1_BM_SW_ARM_versatile_pb\uart.c - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

uart.c | startup.s | linker-script.ld

```c
#include "uart.h" // including function prototype
#define UART0DR *((volatile unsigned int* const)((unsigned int*) 0x101f1000)) // uart data register base adress and port 0 adress

void uart_send_string(unsigned char* p_tx_string){

    while(*p_tx_string != 0){

        UART0DR = (unsigned int)*p_tx_string; //printing character
        p_tx_string++; // pointing to the next character



    }


}
```

C source file | length : 407 lines : 16 | Ln : 13 Col : 6 Pos : 401 | Windows (CR LF) | UTF-8 | INS

[redacted]\Downloads\Embedded\labs\LAB_1_BM_SW_ARM_versatile_pb\uart.h - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

uart.h

```c
//Header Protection
#ifdef _UART_H_
#define _UART_H_

void uart_send_string(unsigned char* p_tx_string);

#endif
```

2-Startup:

startup.s

```
.global reset

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

3-Linker Script:

linker-script.ld

```
ENTRY(reset)

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

SECTIONS
{
    . = 0x10000;
    .startup . :
    {
       startup.o(.text)
    }> Mem


    .text :
    {
      *(.text)  *(.rodata)
    }> Mem

    .data :
    {

      *(.data)

    }> Mem

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

    . = . + 0x1000;
    stack_top = .;

    }
```

4- Compiling (app.c,uart.c,startup.s) files including debug section:

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-gcc.exe  -c -g -mcpu=arm926ej-s app.c -o app.o

[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-gcc.exe  -c -g -mcpu=arm926ej-s uart.c -o uart.o

[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ 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
```

5- debug sections for (app.c,uart.c,startup.s):

-startup:

```
$ 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
  4 .debug_line   0000003a  00000000  00000000  00000066  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_info   00000076  00000000  00000000  000000a0  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  6 .debug_abbrev 00000014  00000000  00000000  00000116  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  00000130  2**3
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

-app.o

```
$ 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 .debug_info   0000009e  00000000  00000000  00000114  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_abbrev 00000084  00000000  00000000  000001b2  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_loc    0000002c  00000000  00000000  00000236  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  00000262  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   00000035  00000000  00000000  00000282  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .debug_str    000000a1  00000000  00000000  000002b7  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .comment      00000012  00000000  00000000  00000358  2**0
                  CONTENTS, READONLY
 11 .ARM.attributes 00000032  00000000  00000000  0000036a  2**0
                  CONTENTS, READONLY
 12 .debug_frame  0000002c  00000000  00000000  0000039c  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

-uart.o

```
$ 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 .debug_info   0000005c  00000000  00000000  00000084  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 00000051  00000000  00000000  000000e0  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000002c  00000000  00000000  00000131  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  0000015d  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000003f  00000000  00000000  0000017d  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000082  00000000  00000000  000001bc  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  0000023e  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000032  00000000  00000000  00000250  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000028  00000000  00000000  00000284  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

6- linker symbols before resloving:

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D string_buffer1
00000000 R string_buffer2
         U uart_send_string

[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-nm.exe uart.o
00000000 T uart_send_string

[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-nm.exe startup.o
         U main
00000000 T reset
         U stack_top
00000008 t stop
```

7- .elf file with debug info sections and symbols after resovling:

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-ld.exe -T linker-script.ld app.o uart.o startup.o -o learn-in-depth.elf -Map=Map_file.txt
```

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-nm.exe learn-in-depth.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_send_string
```

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ 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         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
  5 .debug_line   000000ae  00000000  00000000  0000817f  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_info   00000170  00000000  00000000  0000822d  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_abbrev 000000e9  00000000  00000000  0000839d  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_aranges 00000060  00000000  00000000  00008488  2**3
                  CONTENTS, READONLY, DEBUGGING
  9 .debug_loc    00000058  00000000  00000000  000084e8  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .debug_str    000000b4  00000000  00000000  00008540  2**0
                  CONTENTS, READONLY, DEBUGGING
 11 .debug_frame  00000054  00000000  00000000  000085f4  2**2
                  CONTENTS, READONLY, DEBUGGING
```

8-Extracting binaries from .elf file

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf  learn-in-depth.bin
```

9-Using QEMU simulator to run the baremetal sw:

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn-in-depth.bin
Learn-in-Depth: Hassan Attia
```

10-using GDB commands to debug the baremetal sw on arm-versatilepb:

```
[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ qemu-system-arm -M versatilepb -m 128M -nographic -s -S -kernel learn-in-depth.bin
```

```
[redacted]/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb

[redacted] MINGW32 ~/Downloads/Embedded/labs/LAB_1_BM_SW_ARM_versatile_pb
$ 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 C:\Users\hassa\Downloads\Embedded\labs\LAB_1_BM_SW_ARM_versatile_pb
\learn-in-depth.elf...done.
(gdb)
```

11-Build a connection between the host machine and the target machine(arm-versatilepb) via TCP(ethernet) connection:

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
reset () at startup.s:4
4                       ldr sp, =stack_top
(gdb)
```

12-Displaying instructions and which instruction has program counter to perform :

```
display/4i $pc
1: x/4i $pc
=> 0x10000 <reset>:     ldr     sp, [pc, #4]    ; 0x1000c <stop+4>
   0x10004 <reset+4>:   bl      0x10010 <main>
   0x10008 <stop>:      b       0x10008 <stop>
   0x1000c <stop+4>:    andeq   r1, r1, r0, asr #2
(gdb)
```

13-Print a value of variable:

```
print string_buffer1[1]
$1 = 101 'e'
```

14-Step instruction:

```
=> 0x10000 <reset>:     ldr     sp, [pc, #4]    ; 0x1000c <stop+4>
   0x10004 <reset+4>:   bl      0x10010 <main>
   0x10008 <stop>:      b       0x10008 <stop>
(gdb) si
reset () at startup.s:5
5                       bl main
1: x/3i $pc
=> 0x10004 <reset+4>:   bl      0x10010 <main>
   0x10008 <stop>:      b       0x10008 <stop>
   0x1000c <stop+4>:    andeq   r1, r1, r0, asr #2
```

15-Putting some breakpoints:

```
(gdb) b main
Breakpoint 2 at 0x10018: file app.c, line 11.
(gdb) b reset
Breakpoint 3 at 0x10004: file startup.s, line 5.
(gdb) b uart_send_string
Breakpoint 4 at 0x10038: file uart.c, line 6.
(gdb) info breakpoint
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x00010010 in main at app.c:8
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00010018 in main at app.c:11
3       breakpoint     keep y   0x00010004 startup.s:5
4       breakpoint     keep y   0x00010038 in uart_send_string at uart.c:6
```

MAKEFILE

1-Incremental Build:

```
#Author: Hassan Attia
all: learn-in-depth.bin
    @echo "*******Build is Done********"
learn-in-depth.bin: learn-in-depth.elf

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

learn-in-depth.elf: app.o uart.o startup.o
    arm-none-eabi-ld.exe -T linker-script.ld -Map=Map_file.txt app.o uart.o startup.o -o learn-in-depth.elf


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

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

startup.o: startup.s
    arm-none-eabi-as.exe -mcpu=arm926ej-s -g  startup.s -o startup.o

clean:
    rm *.o

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

2-Some simplifications using (build process/dependencies/target)

```
#Author: Hassan Attia

CC=arm-none-eabi-
CFLAGS=-mcpu=arm926ej-s -g
INCS=
LIBS=


all: learn-in-depth.bin
    @echo "*******Build is Done********"
learn-in-depth.bin: learn-in-depth.elf

    $(CC)objcopy.exe -O binary $< $@

learn-in-depth.elf: app.o uart.o startup.o
    $(CC)ld.exe -T linker-script.ld $(LIBS) -Map=Map_file.txt app.o uart.o startup.o -o learn-in-depth.elf


app.o: app.c
    $(CC)gcc.exe $(FLAGS) -c $(INCS) $< -o $@

uart.o: uart.c
    $(CC)gcc.exe $(FLAGS) -c $(INCS) $< -o $@

startup.o: startup.s
    $(CC)as.exe $(FLAGS)  $< -o $@

clean:
    rm *.o

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

3-Using wildcard to simplify more:

```
#Author: Hassan Attia

CC=arm-none-eabi-
CFLAGS=-mcpu=arm926ej-s -g
INCS=
LIBS=
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
AsOBJ = $(As:.s=.o)
Project_Name=learn-in-depth

all: $(Project_Name).bin
    @echo "*******Build is Done********"

$(Project_Name).bin: $(Project_Name).elf

    $(CC)objcopy.exe -O binary $< $@

$(Project_Name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker-script.ld $(LIBS) -Map=Map_file.txt $(OBJ) $(AsOBJ) -o $@


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


startup.o: startup.s
    $(CC)as.exe $(FLAGS)  $< -o $@

clean:
    rm *.o

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