1/5/2022

Embedded C

Lab (1)

Reham Nady Abd Elmotaal

1-Introduction:

Write a baremetal software to send string “Learn-in-depth: Reham Nady” using UART protocol in VersatilePB micro-controller chip which is based on arm926ej-s micro-processor . I will build everything from scratch including startup ,linker script and source codes ,and compile them using arm cross tool chain ,and using qemu tool to execute this simple software on virtual board.

2-Source codes:

2.1:app.c

This is main application code,defined an array and then pass it to uart_send function which defined in another file.

app.c

```c
#include"uart.h"

unsigned char buffer[100]="learn-in-depth:Reham Nady";
 int main() {
  uart_send(buffer);
  return 0;
}

```

2.2: uart.c:

this function received char by char and then transmit it using UARTDR register.

uart.c

```c

#include"uart.h"

void uart_send(unsigned char*p_tx_str){
  while(*p_tx_str ){
    UART0DR = (unsigned int )*p_tx_str;
    p_tx_str++;
  }
}

```

2.3:uart.h:

I will send my message via UART port 0 which is mapped in address 0x101f1000. this register is placed at offset `0x0`, so you need to read and write at the beginning of the memory allocated for the UART0, there is a macro will be used to write data at the memory address of UART0DR to be transmitted to UART0.

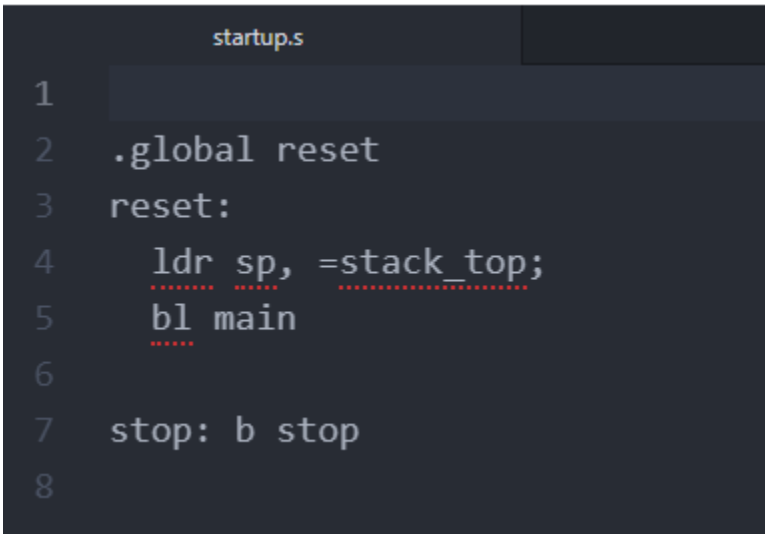
uart.h

```c
#ifndef _UART_H_
#define _UART_H_

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

void uart_send(unsigned char*p_tx_str);

#endif
```

3-Startup:

This startup code is written in assembly code to initialize the stack.

In this code we just make a reset section .in this section I just initialized the stack pointer with “stack_top” which is a symbol will be resolved while linking process,then brunch to main function.



4-Linker script:

In this linker script, we define the entry point of my application which is the reset section, then defined memory boundaries ,in this app we have just one memory . the last section in linker used to divide my code in all file and organize it to burn it on the micro controller.

Linker_scrip.ld

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

}
SECTIONS{
  . = 0x10000;
  .startup . :{
    startup.o(.text)
    }>Mem
  .text :{
    *(.text)
  }>Mem
  .data :{
    *(.data)
  }>Mem
  .bss :{
  *(.bss)
  }>Mem
   . = . + 0x1000;
  stack_top = .;
}
```

5-Symbols:

5.1:symbol of app.o:

1- buffer : which in data section

2- main: which in text section

3- uart_send: unresolved symbol and will be resolved during Linking process.

```
00000000 D buffer
00000000 T main
         U uart_send
```

5.2:symbol of uart.o:

1-uart_send: which in text section

```
00000000 T uart_send
```

5.3:symbol of startup.o:

1- main: unresolved symbol and will be resolved during Linking process.

2- reset: which in text section.

3-stack_top: unresolved symbol and will be resolved during Linking process.

4- stop: which in text section.

```
         U main
00000000 T reset
         U stack_top
00000008 t stop
```

5.4:elf image sympols:

1- buffer: which in data section.

2- main: which in text section.

3-reset: which in text section.

4- stack_top: which in data section.

5- stop: which in text section.

6- uart_send: : which in text section.

```
00010084 D buffer
00010010 T main
00010000 T reset
000110e8 D stack_top
00010008 t stop
00010030 T uart_send
```

6-Sections Headers:

6.1: app.o sections headers

```
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         00000064  00000000  00000000  00000054  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000b8  2**0
                  ALLOC
  3 .comment      0000007f  00000000  00000000  000000b8  2**0
                  CONTENTS, READONLY
  4 .ARM.attributes 00000032  00000000  00000000  00000137  2**0
                  CONTENTS, READONLY
```

1-text section: size of instruction code =0x20.

2-data section: size of initialized global array = 0x64 =100 byte.

3-bss section: size of uninitialized global =0x0.

6.2: uart.o sections headers

```
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
```

1-text section: size of instruction code =0x54.

2-data section: size of initialized global = 0x0.

3-bss section: size of uninitialized global =0x0.

6.3: startup.o sections headers

```
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
```

Only text section which size of 0x10.

6.3: elf image sections headers

```
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         00000064  00010084  00010084  00010084  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  3 .ARM.attributes 0000002e  00000000  00000000  000100e8  2**0
                  CONTENTS, READONLY
  4 .comment      0000007e  00000000  00000000  00010116  2**0
                  CONTENTS, READONLY
```

1-startup section: size of startup code =0x10,and located at entry point.

2-text section: size of instruction code = 0x74.

3-data section: size of initialized global =0x64.

7-executing app:

```
reham@MYCOMPUTER MINGW64 /e/Program Files (x86)/qemu
$ qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn.bin
learn-in-depth:Reham Nady
```