**Lab2 report**

Inhalt

[Lab description 1](#_Toc59282208)

[Step 1: define project files 1](#_Toc59282209)

[Step 2 : C Startup code 2](#_Toc59282210)

[Startup Summury: 2](#_Toc59282211)

[Startup.s lab1 2](#_Toc59282212)

[Startup functions: 2](#_Toc59282213)

[Code: 3](#_Toc59282214)

[Step 3: Linker script 3](#_Toc59282215)

[Step 4: CHECK ENTRY POINT Validation 3](#_Toc59282216)

[Step 5: QEMU Simulator 4](#_Toc59282217)

Lab description

Board: Arm Versatile

Emulator: QEMU

Goal: compile using GCC arm-none-eabi, Symbols solving visualization

and GCC compilation process illustration

Project description: enabling UARTDR register to send and display string message from app layer

Step 1: define project files

Files to be created: uart.c, uart.h and app.c

Used commands in cmd:

* arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o >>> generate obj for app
* arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s uart.c -o uart.o >>> generate obj for uart
* arm-none-eabi-objdump.exe -h app.o >>> display genrated sections for app.o (.text, .data, .bss, .rodate, debug info )
* arm-none-eabi-objdump.exe -D app.o > app.s >>> generate disassembly file from the bin
* arm-none-eabi-objdump.exe -s app.o >>> Display full content all sections

Step 2 : C Startup code

***Startup code for C programs usually consists of the following series of actions:***

Disable all interrupts.

Create a vector table for your microcontroller. Vector table are MCU specific

Copy any initialized data from ROM to RAM.

Zero the uninitialized data area.

Allocate space for and initialize the stack.

Initialize the processor’s stack pointer.

Create and initialize the heap

Enable interrupts.

Callmain

Important forr the Stack:

* Stack **(r13” SP”)**
* Global variables
* Initialized **.data**
* Uninitialized. bss
* Read-only data **.rodata**
* Then force the PC register to jump on the main functions

Startup Summury:

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 ().

Startup.s lab1

Startup functions:

Startup.s should include the following:

1. Create a reset section and Call main( ).
2. Initialize Stack

Code:

. global reset, interrupt\_vectors

interrupt\_vectors:

b reset

b UNDEF\_Handler

b SWI\_Handler

b PABT\_Handler

b DABT\_Handler

b NULL\_Handler

reset:

ldr sp, = 0x00011000

bl main

stop b stop

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

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

Step 3: Linker script

Linker script commands:

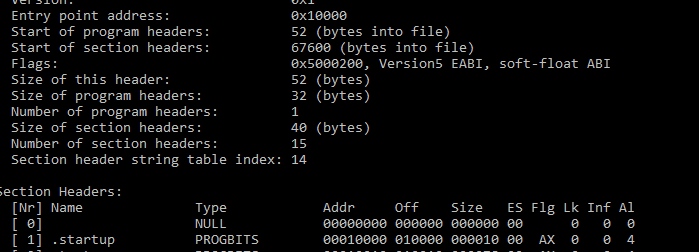
* ENTRY
* MEMORY: (ORGIN = , LENGTH= ) gilt für (vect, rom, ram, sram )
* SECTIONS:
* Location counter
* Section: {….}>(vma) AT>(lma)
* Symbols
* ALIGN
* KEEP
* INPUT
* OUTPUT

To read the symbols: arm-none-eabi-nm.exe app.o

Step 4: CHECK ENTRY POINT Validation

CHECK ENTRY POINT + START UP = arm-none-eabi-readelf.exe -a learn-in-depth.elf

Set start up address and entry point given in linker script = 0x10000



Step 5: QEMU Simulator

To run the program in the QEMU Simulator: qemu-system-arm-M versatilepb-m 128M-nographic-kernel test.bin

