Bare-Metal Development
M2M Lectures
Grenoble University

Your Names Here

January 12, 2021

1 Preface by Pr. Olivier Gruber

This document is your work log for the first step in the M2M course, master-level, at the University of Grenoble, France. You will have such a document for each step of our course together.

This document has two parts. One part is about diverse sections, each with a bunch of questions that you have to answers. The other part is really a laboratory log, keeping track of what you do, as you do it.

The questions provide a guideline for your learning. They are not about getting a good grade if you answer them correctly, they are about giving your pointers on what to learn about.

The goal of the questions is therefore not to be answered in three lines of text and be forgotten about. The questions must be researched and thoroughly understood. Ask questions around you if things are unclear, to your fellow students and to me, your professor.

Writing down the answers to the questions is a tool for helping your learn and remember. Also, it keeps track of what you know, the URLs you visited, the open questions that you are trouble with, etc. The tools you used. It is intended to be a living document, written as you go.

Ultimately, the goal of the document is to be kept for your personal records. If ever you will work on embedded systems, trust me, you will be glad to have a written trace about all this.

REMEMBER: plaggia is a crime that can get you evicted forever from french universities... The solution is simple, write using your own words or quote, giving the source of the quoted text. Also, remember that you do not learn through cut&paste. You also do not learn much by watching somebody else doing.

2 QEMU

What is QEMU? Why is it necessary here?

Read the README-QEMU-ARM.
Try it... with "make run"

3 GNU Debugger

Here, try single-stepping the code via the GNU debugger (GDB).
Read the README-GDB.
Try it, in one shell, launch QEMU in debug mode and in another shell launch gdb.

4 Makefile

You need to read and fully understand the provided makefile. Please find a few questions below highlighting important points of that makefile. These questions are there only to guide your reading of the makefile. Make sure they are addressed in your overall writing about the makefile and the corresponding challenge of building bare-metal software.

1. What is the TOOLCHAIN?
2. What are VersatilePB and VersatileAB?
3. What is a linker script? Look at the linker option "-T"
4. Read and understand the linker script that we use
5. Why do we translate the "kernel.elf" into a "kernel.bin" via "objcopy"
6. What ensures that we can debug?
7. What is the meaning of the "-nostdlib" option? Why is it necessary?
8. Try MEMORY=32K, it fails, why? Look at the linker script.
9. Could you use printf in the code? Why?

4.1 Linker Script

Detail here your understanding of the linker script that we use.
Why do we translate the "kernel.elf" into a "kernel.bin" via "objcopy"
Why do we link our code to run at the 0x10000?
Why do we make sure the code for the object file "startup.o" is first?

4.2 ELF Format

What is the ELF format?

Why is it used as an object file format and an executable file format.

How does the ELF executable contain debug information? Which option must be given to the compiler and linker? Why to both?

Confirm what ELF object files and the final ELF executable are with the shell command "file".

Look at the ELF object files and the final ELF executable with the tool: arm-none-eabi-objdump.

5 Startup Code

Read and understand the startup code in the file "startup.s".

Explain here what it does.

6 Main Code

Read and understand the main code in the file "main.c".

Explain here what it does. In particular, explains how the characters you type in the terminal window actually appear on the terminal window.

With a regular shell, the shell echoes the character as you type them. It only sends the characters once you hit the return key, as a complete line.

It is the behavior you notice here?

1. What is an UART and a serial line?
2. What is the purpose of a serial line here?
3. What is the relationship between this serial line and the Terminal window running a shell on your laptop?
4. What is the special testing of the value 13 as a special character and why do we send back '\r' and '\n'?
5. Why can we say this program polls the serial line? Although it works, why is it not a good idea?
6. How could using hardware interrupts be a better solution?
7. Could we say that the function uart_send may block? why?
8. Could we say that the function uart_receive is non-blocking? why?
9. Explain why uart_send is blocking and uart_receive is non-blocking.

7 Test Code – TODO

7.1 Blocking Uart-Receive

Change the code so that the function uart_receive is blocking.

Why does it work in this particular test code?

Why would it be an interesting change in this particular setting?

7.2 Adding Printing

We provided you with the code of a kernel-version of printf, the function called "kprintf" in the file "kprintf.c".

Add it to the makefile so that it is compiled and linked in.

Look at the function "kprintf" and "putchar" in the file "kprintf.c". Why is the function "putchar" calling the function "uart_send"?

Use the function kprintf to actually print the code of the characters you type and not the characters themselves.

Hit the following special keys:

- left and right arrow.
- backspace and delete key.

Explain what you see.

7.3 Line editing

The idea is now to allow the editing of the current line:

- Using the left and right arrows
- Using the "backspace" and "delete" keys

First, experiment using the left/right arrows... and the backspace/delete keys...

- Explain what you see
- Explain what is happening?

Now that you understand, write the code to allow for line editing.

8 Laboratory Log