Embedded Systems Hands-On 1: Design and Implementation of Hardware/Software Systems

Task 1: Embedded Linux for the Cortex-A53

Carsten Heinz

May 8, 2020

TECHNISCHE UNIVERSITÄT DARMSTADT

This task deals with the Raspberry [Pi] and embedded Linux. It is scheduled for two weeks.

Summary

- Operating the Raspberry [Pi]
- Starting up an embedded Linux
- [OpenOCD] communication with the microcontroller
- Compiling the Linux kernel with a TFT driver
- Device tree and kernel modules

Approach

This and all following exercises rely heavily on online resources. This approach is chosen deliberately to give you an idea about the general approach when getting into touch with new platforms. Furthermore, we require you to document every step you took to reach your solution. Commit your solution and the documentation in your [Git] repository (`ESH01/Group<i>`). *Your documentation has to be conclusive. Make sure that someone else can follow your progress solely by reading your documentation.*

1 Raspberry Pi

The first subtask deals with setting up of the Raspberry [Pi]. To get started, you need an operating system that is loaded during boot up from the SD card attached to the Raspberry [Pi].

Summary

- Download and boot the current [Raspbian] version on the Raspberry [Pi].

[Raspbian] is a Debian based Linux distribution focused on the Raspberry [Pi]. Also the Raspberry [Pi] provides an HDMI output to display a graphical interface, we won't use that capability. Instead you will interact with the Raspberry [Pi] using an SSH connection over Ethernet. Therefore, it is recommended to use [raspbian-ua-netinst].

Install [Raspbian] on your Raspberry [Pi]. Connect your Raspberry [Pi] to a router or the second network port of our lab working stations. Test the SSH and Internet connection of the Raspberry [Pi]. When using the lab work stations, you can share the internet connection with the Raspberry [Pi] using NetworkManager.

2 OpenOCD

This subtask deals with the installation and configuration of [OpenOCD], used to communicate with the Cortex-M0 over SWD. The actual implementation details of the SWD protocol will be analyzed later.

Summary

- Compile [OpenOCD]
- Test the SWD connection to the Cortex-M0

[OpenOCD] is a widely used application to communicate with a variety of processors via JTAG and SWD. As we need the newest version of [OpenOCD], you have to compile the tool from source. Compile and install [OpenOCD] *on the Raspberry [Pi]* as indicated in the `README` located in the [OpenOCD-source] repository.

[OpenOCD] requires two configuration files. The first defines the *interface*, which is used at the host of the communication. In this lab, GPIO pins of the Cortex-A53 are used to drive the SWD protocol. The second part of the configuration describes the *target*, i.e., the Cortex-M0 in our case. [OpenOCD] allows to combine the interface and the target configuration into a single file. The default installation itself provides many frequently used configuration files, which you can find in `/usr/local/share/openocd/scripts`. The following configuration can be used to select the GPIO pins of the Cortex-A53 as the interface and the Cortex-M0 as the SWD target:

`stm32f0raspberry.cfg`

```
1  interface sysfsgpio
2
3  sysfsgpio_swdio_num     SWDIO_PIN_HERE
4  sysfsgpio_swclk_num     SWCLK_PIN_HERE
5
6  transport select swd
7
8  source [find target/stm32f0x.cfg]
9
10 init
11 targets
```

This configuration file targets the [STM32F0] processor, which is the Cortex-M0 mounted on the extension board.

The pin numbers used for the interface configuration are determined by the [STM32F0] (which requires the SWDIO and SWCLK signals to be applied at certain physical pins), as well as the extension board and the Raspberry [Pi] (which define the signal mapping between the Cortex-A53 and the Cortex-M0 pins). Refer to the corresponding datasheets to figure out all pins involved in the SWD communication.

Save the configuration file as `stm32f0raspberry.cfg` and run it through [OpenOCD] with `openocd -f stm32f0raspberry.cfg`. If everything is properly configured and connected, a processor ID should be displayed and [OpenOCD] starts a [GDB] server that can be used for debugging.

Minimum Expected Documentation

- Pins on the Cortex-A53, the Raspberry [Pi], the extension board, and the Cortex-M0 involved in the SWD communication
- Output of [OpenOCD] when running the configuration file
- Any error experienced during compilation and execution

3 Compile a Linux Kernel

Summary

- Compile a current Linux kernel with FBTFT as module.

The simplest way to compile a kernel for the Raspberry [Pi] is to compile it on the Raspberry [Pi] itself. This might take a long time, as the computational power of the Cortex-A53 is rather limited. We will thus cross-compile the kernel on our much more powerful work stations. A Cross-compiler runs on one architecture (e.g., the X86-64 of our PCs) but generates code that can be run on a different target architecture (e.g., the ARMv8 of the Cortex-A53).

Get the latest version of the Raspberry [Pi] optimized [Kernel]. The cloning of the large repository might take a while.

In the meantime, the [Cross-Compiler] toolchain has to be installed. Add `tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin` to your PATH to make it usable.

After cloning the [Kernel] and installing the [Cross-Compiler], a default configuration is loaded by running the following commands inside the repository:

```
/path/to/kernel
KERNEL=kernel7
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
```

The parameter `ARCH` defines the target architecture and `CROSS_COMPILE` the compiler to use.

To view the default configuration that was set through the previous command, you can run

```
/path/to/kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Make sure that
`Device Drivers -> Staging drivers -> Support for small TFT LCD display modules`
is installed as a module which is shown by a "(m)". If the driver is not selected, you can change the installation status using the space key. Additionally, make sure that

- `FB driver for the ILI9340 LCD Controller` and
- `Module for adding FBTFT devices`

will be installed as modules. Close `menuconfig` and save the configuration file. Build the kernel using

```
/path/to/kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j4 zImage modules
```

You can use `-jn` to run make in parallel with `n` threads.

Install the finished kernel as detailed in [Kernel-Building]. You might want to use SSH and SCP instead of switching the SD card to your PC.

Minimum Expected Documentation

- The `.config` of the [Kernel] build
- Compile time of the [Kernel] on a X86-64 machine *and* on the Raspberry [Pi]

4 Loading Linux Modules

In this subtask, the driver installed in Subtask 3 is used to control the [ILI9340] display.

Summary

- Load the FBTFT driver with the module `fbtft_devices`.

Make sure you understand how the display and the Raspberry [Pi] are connected. Refer to the [Display-Pinout] and the Raspberry [Pi-Pinout] for any details.

The `ftbft_device` module is used to configure the driver. You can load the module with `modprobe fbtft_device`. Look at the output of the loaded module in `dmesg`. Check the [FBTFT-Wiki] to load the module with the correct parameters for the [ILI9340] chipset. A successfully loaded driver will generate a framebuffer device at `/dev/fb1` to control the display. Test the display with a suitable tool such as `con2fbmap`.

Minimum Expected Documentation

- Pin configuration of the display and the Raspberry [Pi]
- Configuration of the driver
- Any problems you faced getting the display to work
- Frames per second achieved by the display

5 Compile FBTFT into the Kernel

One downside of using modules to drive the display is the late stage at which it can be loaded during the boot process. Early boot messages can not be displayed. To load the driver as early as possible, it can be compiled into the kernel and loaded via a device tree.

Summary

- Compile a current Linux kernel with FBTFT included
- Use a device tree to load the driver

Compile and install the Linux kernel again, but this time with the display driver as a component indicated by a (*).

Device trees can be used to indicate which drivers for which devices should be loaded during boot up. The device tree lists configuration of peripheral components hierarchically (e.g., this SPI device is used to access this display using these settings), and loads the correct drivers with the indicated settings. Refer to the [Device-Tree-Example] and create a device tree using the parameters found in Subtask 4. Compile and include your device tree as shown in [Device-Tree-Integration].

A correct configuration will result in console output on the display during the boot process.

Minimum Expected Documentation

- The `.config` of the [Kernel] build
- Device tree for the [ILI9340] display.

Bibliography

Cross-Compiler Github. URL: https://github.com/raspberrypi/tools (visited on 2018-03-08).

Device-Tree-Example Github. *hy28a-overlay.dts*. URL: https://github.com/notro/fbtft/blob/master/dts/overlays/rpi/hy28a-overlay.dts (visited on 2018-03-08).

Device-Tree-Integration Github. *FBTFT RPI overlays*. URL: https://github.com/notro/fbtft/wiki/FBTFT-RPI-overlays (visited on 2018-03-08).

Display-Pinout Github. *Signalmapping of the Display Header*. URL: https://github.com/notro/fbtft/wiki/LCD-Modules#adafruit-22 (visited on 2018-03-08).

FBTFT-Wiki Github. URL: https://github.com/notro/fbtft/wiki/fbtft_device (visited on 2018-03-08).

GDB GNU. *The GNU Project Debugger*. URL: https://www.gnu.org/software/gdb/ (visited on 2018-03-08).

Git URL: https://git-scm.com/ (visited on 2018-03-08).

ILI9340 ILITEK. *a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color*. 2010. URL: https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf (visited on 2018-03-08).

Kernel Github. *Kernel source tree for Raspberry Pi Foundation-provided kernel builds*. URL: https://github.com/raspberrypi/linux.git (visited on 2018-03-08).

Kernel-Building Raspberry Pi Foundation. URL: https://www.raspberrypi.org/documentation/linux/kernel/building.md (visited on 2018-03-08).

OpenOCD Dominic Rath. *Open On-Chip Debugger*. URL: http://openocd.org (visited on 2018-03-08).

OpenOCD-source *Open On-Chip Debugger*. URL: https://sourceforge.net/p/openocd/code/ci/v0.10.0/tree/ (visited on 2018-03-08).

Pi Raspberry Pi Foundation. *Raspberry Pi 3 Model B*. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b (visited on 2018-03-08).

Pi-Pinout Github. *Signal mapping of the Raspberry Pi 2 Header*. URL: http://i.stack.imgur.com/vYpJn.png (visited on 2018-03-08).

Raspbian URL: https://www.raspbian.org (visited on 2018-03-08).

raspbian-ua-netinst Github. *Raspbian (minimal) unattended netinstaller*. URL: https://github.com/debian-pi/raspbian-ua-netinst (visited on 2018-03-08).

STM32F0 STMicroelectronics. *STM32F0x1/STM32F0x2/STM32F0x8 advanced ARM-based 32-bit MCUs*. URL: http://www.st.com/resource/en/reference_manual/dm00031936.pdf (visited on 2018-03-08).

Acronyms

FBTFT Framebuffer TFT
GPIO General Purpose IO
HDMI High Definition Multimedia Interface
JTAG Joint Test Action Group
SCP Secure Copy
SD Secure Digital
SPI Serial Peripheral Interface
SSH Secure Shell
SWD Serial Wire Debugging
TFT Thin-Film Transistor