Documentation of Task 01

Further files of this task: See `Task_1/`.

Task 1.1 Raspberry Pi

Hardware preparations:

- Install Raspherry Pi Imager using `sudo apt install rpi-imager` (on Debian-based distros)
- Take sd card with reader and insert in pc
- Open "Imager" and select sd card and Raspherry Pi OS Lite 32 bit
- Press `Ctrl` + `Shift` + `X`:
- Activate Hostname
- Activate SSH - Password
- Username: pi-esho
- Password: RasPi
- Press Write
- insert the SD card in the Raspberry Pi and connect power and ethernet
- we can now run `ssh pi-esho@raspberrypi` to connect to the Raspberry Pi

Software preparations

- update the software with:

```
sudo apt update
sudo apt upgrade -y
```

Task 1.2 OpenOCD

- We follow the instructions on the OpenOCD GitHub Repository
- To build OpenOCD from source we first have to install some dependencies

```
sudo apt update
sudo apt upgrade
sudo apt install git build-essential libtool libtool-bin pkg-config autoconf automake
texinfo
```

- after that we can clone the repository, configure openocd to enable the `bcm2835gpio` interface, build and install openocd

```
# clone the openocd repository
git clone https://github.com/openocd-org/openocd.git
# navigate into the folder
cd openocd
# checkout the latest release v0.12.0
git checkout 9ea7f3d647c8ecf6b0f1424002dfc3f4504a162
# needed when building from the git repository
./bootstrap
# generates the Makefiles required to build. We enable the bcm2835gpio interface
./configure --enable-bcm2835gpio
# build openocd
make
# install it
sudo make install
```

- Compilation succeded without any errors noticed.

- The following pins are involved in the SWD communication:

	SWDIO	**SWCLK**	**(Source)**
Pins on the Cortex-M0	PA13	PA14	STM32F0 Reference Manual, page 922
Pin numbers of the STM32F0 on the extension board	34	37	ESA Board Documentation, PCB Schematic (U11)
Wires on the extension board	JTAG_SWD	JTAG_SWCLK	ESA Board Documentation, PCB Schematic (U11)
Pin numbers on the J1 header on the extension board / on the 40-pin header on the Raspberry Pi	31	29	ESA Board Documentation, PCB Schematic (J1)
GPIO pins on the Cortex-A53	GPIO 6	GPIO 5	Raspberry Pi hardware - Raspberry Pi Documentation

- Thus, GPIO 5 will be used for SWDLK and GPIO 6 for SWDIO. Our configuration file `stm32f0raspberry.cfg` looks like this:

```
adapter driver bcm2835gpio

adapter gpio swdio 6
adapter gpio swclk 5

transport select swd

source [find target/stm32f0x.cfg]

bindto 0.0.0.0

init
targets
```

- With openocd installed, we can run it with our config file. Output after running `openocd -f stm32f0raspberry.cfg`:

```
Open On-Chip Debugger 0.12.0-01004-g9ea7f3d64 (2024-05-16-10:11)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
Info : BCM2835 GPIO JTAG/SWD bitbang driver
Info : clock speed 1006 kHz
Info : SWD DPIDR 0x0bb11477
Info : [stm32f0x.cpu] Cortex-M0 r0p0 processor detected
Info : [stm32f0x.cpu] target has 4 breakpoints, 2 watchpoints
Info : starting gdb server for stm32f0x.cpu on 3333
Info : Listening on port 3333 for gdb connections
    TargetName         Type       Endian TapName            State
--  ------------------ ---------- ------ ------------------ ------------
 0* stm32f0x.cpu       cortex_m   little stm32f0x.cpu       running

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

Task 1.3 Compile a Linux Kernel

Cross-Compile on a x86 machine

- To cross-compile the Linux kernel on a x86 machine, we mainly follow the steps descriped in the Raspberry Pi Linux Kernel Documentation.

- First, we ensure all necessary dependencies and the cross-compiler are installed on our system:

```
sudo apt install git bc bison flex libssl-dev make libc6-dev libncurses5-dev
libncursesw5-dev
sudo apt install crossbuild-essential-armhf # this installs the 32-bit cross-compiler
```

- In the meantime, we clone the Raspberry Pi Linux kernel repository with

```
git clone --depth=1 https://github.com/raspberrypi/linux
cd linux
```

- To compile the kernel:

```
# set shell variable with the target kernel version
KERNEL=kernel7
# create config
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- bcm2709_defconfig
# show config and close it afterwards
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
# compile the kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j36 zImage modules dtbs
```

- The config file is attached as `raspberry-pi-linux-kernel-config_cross-compiling`
- the final kernel image is then located at `./arch/arm/boot/Image`
- To install the kernel, we insert the Pi's SD card back into our computer and mount both partitions to a subfolder of our local repository for convenient access:

  ```
  mkdir mnt
  mkdir mnt/{fat32,ext4}
  sudo lsblk # => sdcard is sdf on my system
  sudo umount /dev/sdf{1,2} # sdcard was already mounted automatically elsewhere by the OS
  sudo mount /dev/sdf1 mnt/fat32/
  sudo mount /dev/sdf2 mnt/ext4/
  ```

- We can then remove all the kernel images currently stored on the SD card and install our self-compiled kernel:

  ```
  sudo rm mnt/fat32/kernel*.img

  sudo env PATH=$PATH make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
  INSTALL_MOD_PATH=mnt/ext4 modules_install # install kernel modules

  # copy kernel and device tree blobs onto the sdcard
  sudo cp arch/arm/boot/zImage mnt/fat32/$KERNEL.img
  sudo cp arch/arm/boot/dts/broadcom/*.dtb mnt/fat32/
  sudo cp arch/arm/boot/dts/overlays/*.dtb* mnt/fat32/overlays/
  sudo cp arch/arm/boot/dts/overlays/README mnt/fat32/overlays/

  sudo umount /dev/sdf{1,2}
  ```

- After all that, we insert the SD card back into our Raspberry Pi and it powers on normally. `uname -a` returns

  ```
  Linux esa-pi 6.6.30-v7+ #1 SMP Thu May 16 16:26:36 CEST 2024 armv7l GNU/Linux
  ```

  so we are now indeed using our self-compiled kernel.

Cross-Compilation on a fedora Linux

- Not everyone in our group uses a Debian-based Linux. On fedora, the build depencencies have to be installed via `dnf`.
- On fedora, we use the `gcc-arm-linux-gnu` cross-compiler. The cross-compiler package can be installed with

  ```
  sudo dnf install gcc-arm-linux-gnu
  ```

and in every `make` command, we have to use `CROSS_COMPILE=arm-linux-gnu-` instead of `CROSS_COMPILE=arm-linux-gnueabihf-`.

Compile Time

- On a system with an Intel Core i7-13700k (where the CPU is power-limited to 155 watts), compilation took around 2.5 mins. `time make ARCH=arm ...` returned:

```
real    2m28,310s
user    52m23,139s
sys     4m46,420s
```

- In contrast, compiling with 12 threads on a Laptop took about 28 min and 12 sec.

Compile on the Raspberry pi

- To compile and install the kernel directly on the Raspberry Pi, we again mainly follow the steps described in the Raspberry Pi Linux Kernel Documentation. We execute the following steps:

```
# install git and build dependencies
sudo apt install git bc bison flex libssl-dev make libncurses5-dev libncursesw5-dev

# clone the repository
git clone --depth=1 https://github.com/raspberrypi/linux

# set shell variable with the target kernel version
KERNEL=kernel7
# create config
make bcm2709_defconfig
# show config and close it afterwards
make menuconfig

# compile the kernel
make -j4 zImage modules dtbs

# install the kernel
sudo make modules_install
sudo cp arch/arm/boot/dts/broadcom/*.dtb /boot/firmware/
sudo cp arch/arm/boot/dts/overlays/*.dtb* /boot/firmware/overlays/
sudo cp arch/arm/boot/dts/overlays/README /boot/firmware/overlays/
sudo rm /boot/firmware/*.img
sudo cp arch/arm/boot/zImage /boot/firmware/$KERNEL.img
```

- The config file is attached as `raspberry-pi-linux-kernel-config_local`.

- After a reboot, `uname -a` returns

```
Linux esa-pi 6.6.30-v7+ #1 SMP Thu May 16 22:35:38 CEST 2024 armv7l GNU/Linux
```

so compilation succeeded.

Compile Time

- On our Raspberry Pi 3s, compilation took almost 3 hours. During compilation, the Pi operated at its thermal limit of 85°C so the Cortex-A53 most likely got throttled heavily. `time make -j4 ...` returned:

  ```
  real    167m29.066s
  user    616m37.755s
  sys     33m2.246s
  ```