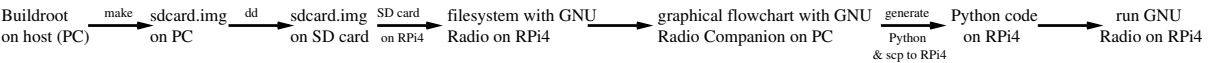
Outline: generic embedded GNU/Linux development methods

Buildroot, as OpenEmbedded/Yocto, supports multiple embedded boards running GNU/Linux: see `https://github.com/buildroot/buildroot/tree/master/board`

1. General presentation of the project: `http://jmfriedt.free.fr/presentation_projet_M1.pdf`
2. First week: getting the Buildroot framework (kernel + library + userspace application + toolchain) functional on the host computer (PC)
3. Second week: getting GNU Radio running on the target system (Raspberry Pi4) – demonstration with FM broadcast radio demodulation and sound transfer to the host used as sound card [1].
4. Design (schematic and routing) of a dedicated radiofrequency source board
5. Later: using these tools to characterize a SAW resonator



[1] G. Goavec-Merou, J.-M Friedt, *"On ne compile jamais sur la cible embarquée" : Buildroot propose GNURadio sur Raspberry Pi (et autres)*, Hackable (2021), at `http://jmfriedt.free.fr/hackable_buildroot.pdf`

Getting started

Embedded systems development is about optimizing resources (lower power consumption for maximum computational power)

- get a functional working environment: a functional GNU/Linux distribution (packages to solve dependencies between libraries and userspace applications): Debian or Ubuntu
- native install on your computer, dual boot or at worst VirtualBox/VMWare virtual machine
- ability to work with the Windows Subsystem for Linux (WSL2 [2] [3] ?): Microsoft is adding POSIX system calls to MS-Windows, worth trying but not tested
- whether native install or virtual machine: start with `netinst` [4] [5] for a minimal setup and add necessary functionalities

[2] `https://devblogs.microsoft.com/commandline/announcing-wsl-2/`
[3] T. Colombo, *WSL2 : cheval de Troie ou cadeau empoisonné ?*, GNU/Linux Magazine France **241** (2020)
[4] `https://www.debian.org/CD/netinst/` for Debian
[5] `https://cdimage.ubuntu.com/netboot/18.04/` for Ubuntu

Linux basics

- package management under Debian/Ubuntu: **apt**
- Ubuntu promotes temporary super user commands prefixed with **sudo**, Debian supports **sudo** if installed, switch to root with **su** - otherwise
- add developer packages: **apt install build-essential**
- if a command is not known: **man command** provides the manual
- basic unix command and tree structure: already addressed at the bachelor level at `http://jmfriedt.free.fr/TP_cmd_unix.pdf`

never, ever, work as root if not performing administration tasks

Embedded system development

Once a functional GNU/Linux (*host* = Intel x86) environment is available:

- develop for the *target* ARM board by cross-compiling: need for a consistent toolchain (compiler and binary handling utilities), kernel (Linux), libraries and userspace applications
- several frameworks provide such consistent functionaliy (Yocto, OpenEmbedded, Buildroot) – the latter being arguably the easiest to grasp and requiring fewer resources (*8 GB hard disk space*)
- fetch the latest stable release of Buildroot:
  wget https://buildroot.org/downloads/buildroot-2022.08.1.tar.gz
  (or check `https://buildroot.org/download.html`)
- uncompress (**gunzip**) and unarchive (**tar xvf**) on a storage medium with at least 8 GB available, possibly external mobile storage medium: **tar zxvf buildroot-2022.08.1.tar.gz**
- *do not attempt* moving the Buildroot directory to some different location after configuring: some hard-coded directory structure will be broken

Embedded system development

First initial compilation of Buildroot

- **tar zxvf buildroot-2022.08.1.tar.gz** to uncompress/unarchive the downloaded file
- **cd buildroot-2022.08.1/** to enter the directory
- **ls configs/raspberrypi*** to check available configurations and that **raspberrypi4_64_defconfig** is supported
- **make raspberrypi4_64_defconfig** to configure with the default configuration
- **make** to compile Buildroot: many archives will be downloaded (requires fast internet connection) and the resulting tree structure requires about 8 GB
- Buildroot should be self-contained and independent of the host operating system assuming basic developer functions are available (**gcc**, **g++**, **make**, **git**, **cmake** ...)
- at the end: **output/images/sdcard.img** is the image to be transferred to the SD card
- bitwise copy from a file to a storage medium: **dd** (Disk Dump)

Embedded system development

- ⚠ **WARNING:** the following command will **definitely delete** all data on the target medium. Make sure how the SD-card is called. It is usually **/dev/sdb** but in case a mobile hard disk/USB stick is inserted, it could be that the SD-card is called something else. Check many times before running **dd**
- identify the block name [6] using **dmesg** | **tail** after inserting the SD card reader, or **lsblk**

```
[514523.735373] scsi 6:0:0:0: Direct-Access     Mass     Storage Device   1.00 PQ: 0 ANSI: 0 CCS
[514523.735669] sd 6:0:0:0: Attached scsi generic sg1 type 0
[514523.994885] sd 6:0:0:0: [sdb] 31422464 512-byte logical blocks: (16.1 GB/15.0 GiB)
[514523.995006] sd 6:0:0:0: [sdb] Write Protect is off
[514523.995008] sd 6:0:0:0: [sdb] Mode Sense: 03 00 00 00
[514523.995129] sd 6:0:0:0: [sdb] No Caching mode page found
[514523.995133] sd 6:0:0:0: [sdb] Assuming drive cache: write through
[514524.024807]  sdb: sdb1 sdb2
[514524.025712] sd 6:0:0:0: [sdb] Attached SCSI removable disk
```

- **sudo dd if=output/images/sdcard.img of=/dev/sdd bs=8M**
  (replace **sdd** with the appropriate medium provided by **dmesg**)
- This procedure will have to be repeated every time a modification is brought to Buildroot.

[6] also make sure a file manager has not automagically mounted the filesystems stored on the SD: if **mount** refers to some automounted filesystem in **/media**, umount them

Network configuration

We need to connect the Raspberry Pi4 to the host computer through an Ethernet link [7]:

- point to point Ethernet connection is most easily established when both computers are on the same sub-network
- on the host computer (personal computer):
  ifconfig -a [8] to identify the name of the network interface
  sudo ifconfig eth0 192.168.2.1 to set [9] the IP (Internet Protocol) address of interface Ethernet **eth0** to **192.168.2.1** [10]
- network configuration is an administrator task: in room 215B, prefix commands with `sudo`

[7] if using a Virtual Box with a GNU/Linux guest on a Microsoft Windows host, configure network as a *Bridged Adapter* (not the default NAT) and check the Windows firewall settings

[8] **ifconfig** is now superseded with **ip**: if **ifconfig** is not available, try **ip addr**

[9] assuming no interference from a network manager

[10] with **ip**: **ip a add 192.168.2.1 dev eth0**

Network configuration

On the SD-card (still inserted in the USB-SD adapter on the host computer)

- we need to set the IP address of the Raspberry Pi4 on the same subnet 192.168.2.X
- network configuration is handled by **/etc/network/interfaces**
- mount the second partition of the SD-card (**mount /dev/sdb2 /mnt** if the SD-card is **sdb**)
- edit the **/mnt/etc/network/interfaces** file to be read by the Raspberry Pi4 (not to be confused with **/etc/network/interfaces** on the host)
- replace the **dhcp** entry (dynamic IP allocation) with

```
iface eth0 inet static
  address 192.168.2.2
  netmask 255.255.255.0
```

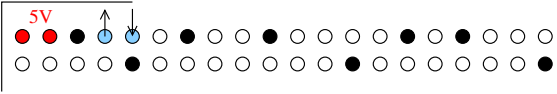
  This will select the default IP address 192.168.2.2 for the Raspberry Pi4
- remove the SD-card: **umount /mnt**
- insert the SD-card in the Raspberry Pi4, connect the Ethernet cable, wait for the Raspberry Pi4 to boot, and **ping 192.168.2.2** from the host

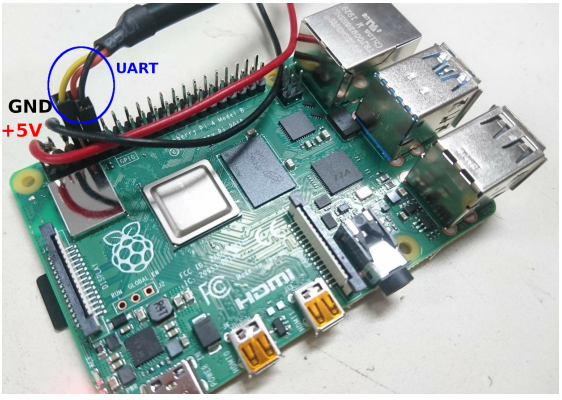
Network configuration

- if all goes well, we get a reply, meaning the Raspberry Pi4 has booted and the network configuration is correct
- We need to run a server to connect to the Raspberry Pi4 from the host computer: Secure SHell (**ssh**) is provided by **dropbear**
- in the Buildroot directory on the host computer: run
  make menuconfig
  to start configuring Buildroot with new packages
- search ("/") the keywork **dropbear** and select this package
- the ssh server requires a root password: `System Configuration` → `Enable root login with password` → provide a password you will remember
- **make** to generate a new **sdcard.img** archive and **dd** to the SD card
- **ssh root@192.168.2.2** to log into the Raspberry Pi4

No Ethernet ? serial-USB cable

In case no Ethernet port is available on the host computer, option 1 is to use a *serial to USB converting* cable: the console displays a login shell





On the PC:

```
minicom -D /dev/ttyUSB0

Welcome to Buildroot
buildroot login:
```

At the login prompt, enter the administrator identifier `root`
since it is the only account available.

No Ethernet ? virtual Ethernet over USB-C (1/2)

In case no Ethernet port is available on the host computer, option 2 is to use the virtual Ethernet over USB-C[11] (might require powering from a USB3 port)

- in the first SD card partition, edit **config.txt** and add a line with
  `dtoverlay=dwc2`
  to load the USB OTG functionality from **overlays/dwc2.dtbo**
- add a file in **/etc/init.d/** of the SD card second partition (the embedded GNU/Linux system) named **S01-module** with

```
modprobe dwc2
modprobe g_ether
```

- make the script on the SD card executable:
  `chmod 755 etc/init.d/S01-module`
- after booting the Raspberry Pi 4, on the host computer, **lsusb** will show

```
Bus 001 Device 051: ID 0525:a4a2 Netchip Technology, Inc. Linux-USB Ethernet/RNDIS Gadget
```

- a new network interface named **usb0** will be available on both the embedded board – see **ifconfig -a** – and the host computer (assuming **g_ether** was **modprobe** on the host computer)

[11] `https://dev.webonomic.nl/4-ways-to-connect-your-raspberry-pi-4-to-the-internet`

No Ethernet ? virtual Ethernet over USB-C (2/2)

1. As described earlier, modify `network/interfaces` of the Raspberry Pi 4 with (here with IP 192.168.3.2)

   ```
   iface usb0 inet static
     address 192.168.3.2
   ```

2. check the name of the new interface on the host computer: it could be that it was renamed from `usb0` to something like `enp0s20u2`: defined its IP address on the host computer in the same subnet (here 192.168.3.1):

   ifconfig usb0 192.168.3.1

   or

   ip a add 192.168.3.1 dev usb0

3. check the routing table of the host computer: **/sbin/route -n**. If there is no entry associated with `usb0` (or its replacement name), add a routing condition: **sudo route add 192.168.3.2 usb0** to route packets through the interface (`usb0`) associated with the RPi4 target address 192.168.3.2

4. check the connection with **ping 192.168.3.2** from the host computer and **ping 192.168.3.1** from the Raspberry Pi 4. In both cases a reply with

   ```
   PING 192.168.3.1 (192.168.1.1) 56(84) bytes of data.
   64 bytes from 192.168.3.1: icmp_seq=1 ttl=64 time=0.181 ms
   64 bytes from 192.168.3.1: icmp_seq=2 ttl=64 time=0.170 ms
   ```

   should be displayed if communication is successful

5. continue with the `dropbear` install for ssh connection (slide 9)

Adding audio support

Audio is not active in the default Buildroot configuration.

To activate audio, add in the `config.txt` of the first partition of the SD card:
`dtparam=audio=on`

After booting, load the sound card driver (`modprobe snd-bcm2835.ko`) so that **dmesg** displays

```
[    X.XXXXXX] bcm2835_audio bcm2835_audio: card created with 8 channels
```

Add the ALSA[12] utilities using `make menuconfig` in Buildroot and select the `speaker-test` function so that the sound can be tested using
`# speaker-test -t sine -f 440`

[12] Advanced Linux Sound Architecture

Further reading

- P. Ficheux & É. Bénard, *Linux Embarqué 4ème édition*, Eyrolles (2012)
- P. Ficheux, *Linux Embarqué – Mise en place et développement*, Eyrolles (2018)
- K. Yaghmour, J. Masters, G. Ben-Yossef, P. Gerum, *Building Embedded Linux Systems, 2nd Ed.*, O'Reilly (2008)
- J. Madieu, *Linux Device Drivers Development*, Packt (2017)
- C. Hallinan, *Embedded Linux Primer: A Practical, Real-World Approach, 2nd Edition*, Prentice Hall (2010)

No hardware? Emulator (RPi3)

qemu as provided in Debian/Ubuntu's `qemu-system-arm` package emulates the Raspberry Pi 3:

```
$ qemu-system-aarch64 -M help | grep raspi
...
raspi3b              Raspberry Pi 3B (revision 1.2)
```

and allows for networking. From the Buildroot `output/images` directory:

```
qemu-system-aarch64 -kernel Image -dtb ./bcm2710-rpi-3-b.dtb          \
-drive file=./sdcard.img,format=raw,if=sd,id=hd-root                  \
-append "rw earlycon=pl011,0x3f201000 console=ttyAMA0 loglevel=8      \
root=/dev/mmcblk0p2 fsck.repair=yes net.ifnames=0 rootwait memtest=1" \
-M raspi3b -m 1024 -serial mon:stdio -no-reboot -nographic            \
-device usb-net,netdev=net0 -netdev user,id=net0,hostfwd=tcp::5555-:22
```

For copying files through `ssh`, root access must be enabled: edit `/etc/ssh/sshd_config` in the emulator and modify `PermitRootLogin yes`.
Restart the server service `/etc/init.d/S50sshd restart` and from the host (PC): `ssh -p 5555 root@localhost` will connect to the emulator.

No hardware? Emulator (RPi4)

After activating the VirtIO drivers in the Linux kernel (`make linux-menuconfig` in the Buildroot directory), e.g. by enabling (Y) first the VirtIO PCI support:

```
CONFIG_VIRTIO_BLK=y
CONFIG_VIRTIO_BLK_SCSI=y
CONFIG_SCSI_VIRTIO=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_MMIO=y
CONFIG_FUSE_FS=y
CONFIG_VIRTIO_FS=y
```

execute [13]

```
qemu-system-aarch64  -M virt -cpu cortex-a72 -nographic -smp 1 -kernel Image \
-append "rootwait root=/dev/vda console=ttyAMA0"  -netdev user,id=eth0       \
-device virtio-net-device,netdev=eth0 \
-drive file=rootfs.ext4,if=none,format=raw,id=hd0 -device virtio-blk-device,drive=hd0
```

to run the Raspberry Pi 4 image on QEMU.

[13] https://raduzaharia.medium.com/system-emulation-using-qemu-raspberry-pi-4-and-efi-87652ff203b7

This week

Demonstrate your ability to

1. setup a functional Buildroot cross-development framework
2. configure the embedded Linux system (IP address)
3. run GNU/Linux on the Raspberry Pi and connecting through the network
4. cross-compile a C program, transfer to the Raspberry Pi and execute

Resources:

- *Never compile on the target ! GNU Radio on embedded systems using Buildroot*, FOSDEM 2021 at `https://archive.fosdem.org/2021/schedule/event/fsr_gnu_radio_on_embedded_using_buildroot/`
- *GNURadio running on embedded boards: porting to buildroot*, European GNU Radio Days 2018 at `https://pubs.gnuradio.org/index.php/grcon/article/view/86`
- *How To Build QEMU Images With Buildroot* at `https://www.youtube.com/watch?v=O9RHMKJqVTg`
- G. Goavec-Merou, J.-M Friedt, *"On ne compile jamais sur la cible embarquée" : Buildroot propose GNURadio sur Raspberry Pi (et autres)*, Hackable (2021), at `http://jmfriedt.free.fr/hackable_buildroot.pdf` [in French]
- Raspberry Pi 4 datasheet at `https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf`

Next week

GNU Radio on Raspberry Pi 4

1. making sure GNU Radio is properly installed: accessing GNU Radio blocks and playing a sound
2. first demonstration with RTL-SDR dongle: FM receiver
3. from RPi4 to PC used as sound card: Zero-MQ publish/subscribe
4. from PC to RPi4: TCP/IP server running as a Python thread

$\rightarrow$ all the tools needed to develop an embedded instrument (data from instrument to PC and control commands from PC to instrument)