Embedded systems 1/7

J.-M Friedt

FEMTO-ST/time & frequency department

jmfriedt@femto-st.fr

slides at `jmfriedt.free.fr`

September 5, 2021

Background

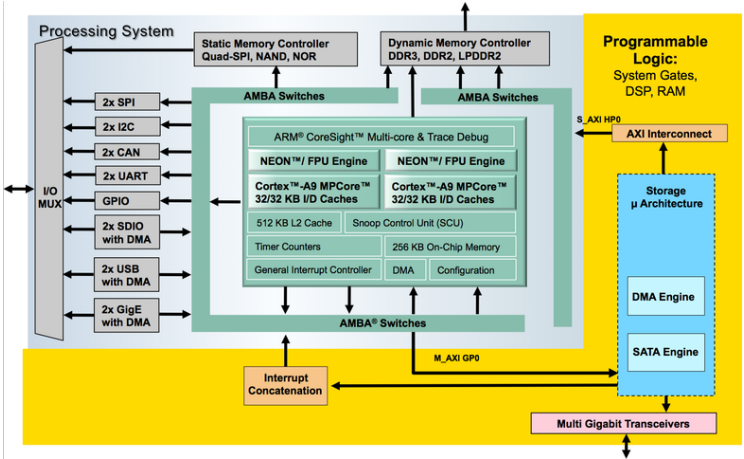
- Working on microcontrollers: no process, a single C program converted to assembly language is executed on the processor
- Only risk for breaking the sequential execution of the program: interrupts
- No abstraction, direct access (read/write) to registers through their address to configure hardware
- No resource sharing mechanism or memory protection from other tasks
- $\Rightarrow$ executive environments (FreeRTOS, RTEMS, NuttX) providing an implementation of such mechanisms on a static monolithic binary
- Operating system (Linux):
  - portable environment with standardized interfaces (POSIX)
  - hardware abstraction mechanisms (drivers, communicating with user space through `/dev` or `/sys/bus`)
  - concept of process, with each task being allocated its own memory space
  - memory space abstraction: MMU (Memory Management Unit)

Outline

1. Transition M1–M2: basics
   Why use GNU/Linux on embedded systems ?
   Cross-compilation and `buildroot` architecture. IP network configuration. Hardware control from userspace (skipping the kernel abstraction layer), link between virtual and hardware memory
2. Kernel-space programming – dynamically loaded kernel module. Communicating through the `/dev` system calls + `ioctl()`, kernel functions (timer)
3. Other user-kernel communication interfaces: `/dev` → `/sys/bus` plateforms, IIO (Industrial Input Output)
4. Hardware description through a *devicetree*, accessing such information from a kernel module, more functions provided by the kernel (semaphores, scheduler tasks)
5. Accessing FPGA resources (Processing Logic – PL) from the general purpose processor (Processing System – PS)
6. source of latencies, real time applications, real time extension of the Linux kernel: Xenomai
7. hardware interrupt handling

The Redpitaya platform

- Xilinx Zynq [1]: hardware architecture providing a general purpose processing system (PS) and a programmable logic FPGA (PL)
- → make the most out of the two architectures (user interface, networking, complex algorithms in the PS, high bandwidth and true parallel computing in the PL)
- AXI bus[2]
- 2 ADC 125 MS/s
- 2 DAC 125 MS/s
- slow ADCs, GPIO, USART, USB, Ethernet ...



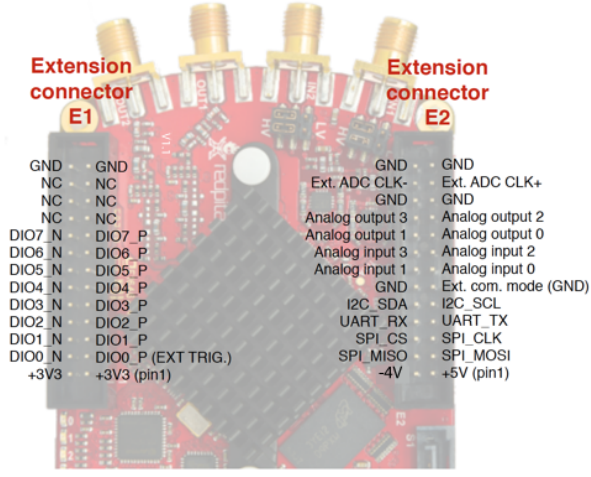
[1] www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, competitor to Intel/Altera SOC www.altera.com/products/soc/overview.html or Microsemi/Actel SmartFusion combining FPGA and Cortex-M3 on a single chip www.microsemi.com/products/fpga-soc/soc-fpgas

[2] 32 bits @ 150 MHz = 1.2 GB/s

The Redpitaya platform

E2 connector (pin 1 is on the board side. next to the SATA connector) [3]

Pin	Description
1	+5V
2	-3.4V (50mA)1
3	SPI(MOSI) PS_MIO10
4	SPI(MISO) PS_MIO11
5	SPI(SCK) PS_MIO12
6	SPI(CS#) PS_MIO13
7	UART(TX) PS_MIO08
8	UART(RX) PS_MIO09
9	I2C(SCL) PS_MIO50
10	I2C(SDA) PS_MIO51
11	Ext com.mode GND (default)
12	GND
13	Analog Input 0 0-3.5V
14	Analog Input 1 0-3.5V
15	Analog Input 2 0-3.5V
16	Analog Input 3 0-3.5V
17	Analog Output 0 0-1.8V
18	Analog Output 1 0-1.8V
19	Analog Output 2 0-1.8V
20	Analog Output 3 0-1.8V
21	GND
22	GND
23	Ext Adc CLK+
24	Ext Adc CLK-
25	GND
26	GND



MIO connected to PS, EMIO to PL

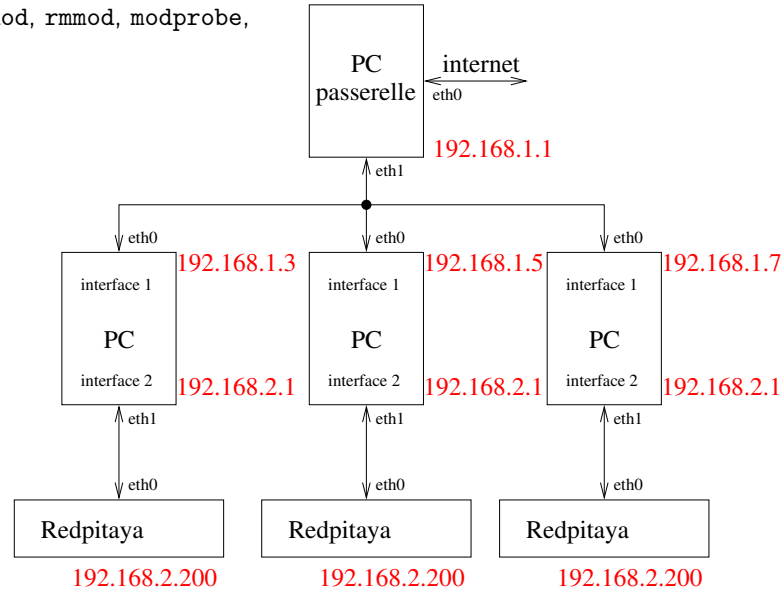
[3] redpitaya.readthedocs.io/en/latest/developerGuide/125-14/extent.html

Objective of the course

- assess various means of accessing hardware (FPGA GPIO to LEDs) resources from the Linux kernel
- need for
  - cross-compiling,
  - control hardware resources from Linux (we will begin with the LEDs connected to the PS),
  - configure the FPGA from GNU/Linux and inform the kernel of the associated resources needed
  - understand the various interfaces between Linux kernel and (GNU) userspace
- You will reproduce the whole process on the slow analog to digital converters (XADC)

Basic commands for network configuration

- Kernel modules: `lsmod`, `insmod`, `rmmod`, `modprobe`, modules in `/lib/modules`
- system status: `dmesg`, `uname`
- network [4]: `ifconfig`, `route`, DNS in `/etc/resolv.conf`, NFS in `/etc/exports`
  ⇒ `mount IP:source dest`



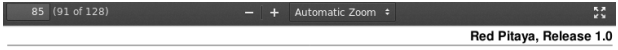
Hardware access from userspace

Through `/sys/class`

- sharing information formated as ASCII sentences
- configuration through appropriate pseudo-files (no `ioctl`)
- well suited for user interaction (*shell* programing)

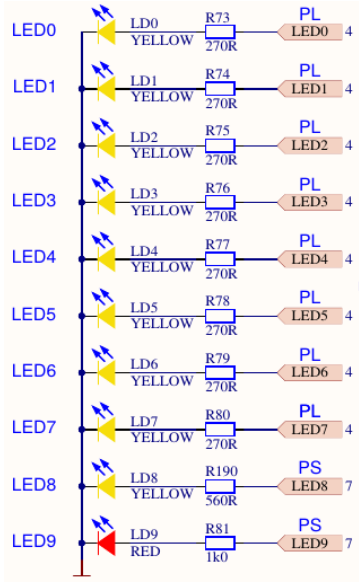
```
# echo "1" > /sys/class/gpio/gpio906/value
```

Warning with C: `fseek` **at offset 0 by** `SEEK_SET` **for multiple accesses without** `fopen` **and** `fclose`



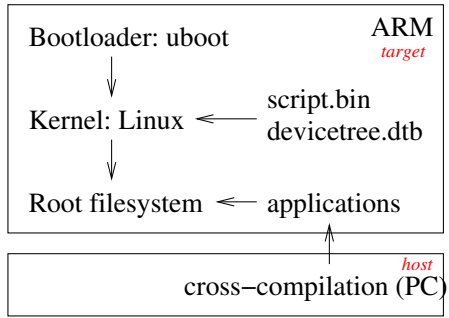
The default pin assignment for GPIO is described in the next table.

FPGA	connector	GPIO	MIO/EMIO index	sysfs index	color, dedicated meaning
		exp_p_io [7:0]	EMIO[15:8]	906+54+[15:8]=[975:968]	
		exp_n_io [7:0]	EMIO[23:16]	906+54+[23:16]=[983:976]	
		LED [7:0]	EMIO[7:0]	906+54+[7:0]=[967:960]	yellow
		LED " [8]"	MIO[0]	906+ [0] = 906	yellow = CPU heartbeat (user defined)
		LED " [9]"	MIO[7]	906+ [7] = 913	red = SD card access (user defined)



Basic reminders on development environment

- Buildroot = generic (Redpitaya, Raspberry Pi ...), homogeneous environment providing an optimized cross-compilation toolchain, libraries, kernel and userspace binaries
- host=x86 PC, target=ARM Redpitaya
- `output/host/usr/bin` includes tools to work on the PC
- `output/build` includes source codes of the binaries to be executed on the target
- in `output/build`, the directory `linux-xilinx-v2019.1` includes the kernel sources
- `BR2_EXTERNAL` principle: extend the official buildroot archive capability with the Reditaya support
- Redpitaya support described at `jmfriedt.free.fr/redpitaya.pdf` requiring the `github.com/trabucayre/redpitaya.git` archive.



No hardware ?

Buildroot image genereted for Redpitaya will run on qemu:

- Debian GNU/Linux `qemu-system-arm` supports the A9 CPU core but not specific peripherals such as Ethernet inteface:

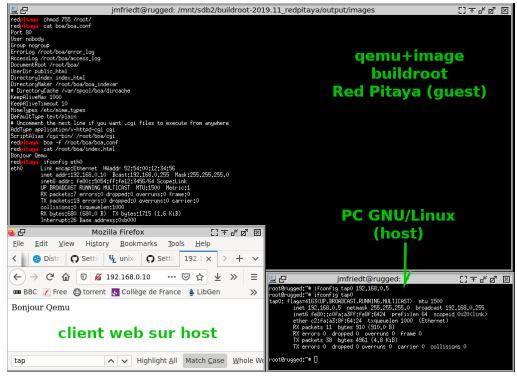
```
qemu-system-arm -append "console=ttyPS0,115200 root=/dev/mmcblk0 rw \
earlyprintk" -M xilinx-zynq-a9 -m 1024  -serial mon:stdio \
-dtb zynq-red_pitaya.dtb -nographic -kernel zImage  -sd rootfs.ext4
```

- custom Xilinx fork [5] [6] of qemu for hardware peripheral support

```
sudo .../aarch64-softmmu/qemu-system-aarch64
-append "console=ttyPS0,115200
root=/dev/mmcblk0 rw earlyprintk"
-M arm-generic-fdt-7series -machine linux=on
-smp 2 -m 1024  -serial mon:stdio
-dtb zynq-red_pitaya.dtb -nographic
-kernel zImage -sd rootfs.ext4 -net nic -net
nic -net nic -net nic -net tap,downscript=no
```

requires administration privileges to create the `tap0` network interface.

- also valid for Raspberry Pi single board computers



[5] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842060/QEMU

[6] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842054/QEMU+-+Zynq-7000

Lab session

- Two LEDs are connected to MIO0 and MIO7: write a shell script to have these two LEDs blink through `/sys/class/gpio`
- Compile a program displaying "Hello World" and execute on the PC and the Redpitaya board following an `ssh` connection and `nfs` remote filesystem mount.
  Buildroot is located at `/home/jmfriedt/buildroot-2021.08_redpit`: export the appropriate `PATH` to have access to the toolchain
- Identify in the datasheet[7] which register, and their location, to configure these GPIOs, and write a C program to have the LEDs blink. What configuration operations must be performed on the GPIOs to achieve sur a result ?
- Switch on and off the LEDs through a web page (`boa` server running on the Redpitaya).
- Integrate the LED control in a TCP/IP server (`socket` → `bind` → `listen` → { `accept` → `recv`/`send` → `close`})

[7] `www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf`