Digital electronics

J.-M Friedt

FEMTO-ST/time & frequency department

jmfriedt@femto-st.fr

slides at `jmfriedt.free.fr`

February 20, 2024

Plan

7 lessons/lab sessions (4-hour long schedules):

1. Executive environments: principles and introduction, getting started with FreeRTOS
2. FreeRTOS, RTEMS, Nuttx ... multitasking and associated methods to make sure shared data and resources are kept in known states (mutex & semaphore)
3. Using the scheduler, mutex and semaphores to solve the "philosopher problem"
4. Embedded systems with GNU/Linux – POSIX compatible operating system
   Architecture of an operating system, kernel v.s userspace
   Internet connectivity and networking
5. Accessing hardware resources from userspace – memory translation from physical to virtual address space (Memory Management Unit) – `/dev/mem`
6. Accessing hardware resources from a web server – internet connected instrument
7. From userspace to kernel space: character device (*char device*) for communicating between users and the kernel

Accessing hardware resources from userspace

- On a microcontroller: `*(type*)address=value;`
- Problem of accessing hardware resources through `/sys`: excessively slow due to all the abstraction layers of the kernel (140 μs to access `/sys/class/gpio` !)
- Direct memory access in C: faster ...
- but requires translating the hardware (real) address to a virtual address + looses consistency management by the kernel
- *Memory Management Unit*: memory access supervisor, virtualization of memory addresses (all processes use the same virtual address space)
- memory accessible from userspace (`root`) through `/dev/mem`
- Fastest: kernel module, but same challenge of memory translation for MMU handling.

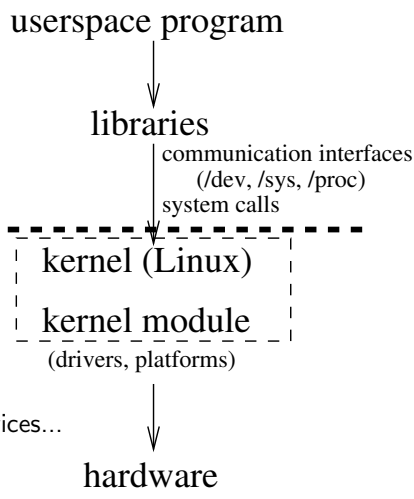
Here userspace access will be used for fast prototyping (shell, C)

Buildroot

Consistent environment providing

- a cross-compilation toolchain
- a Linux kernel
- userspace applications
- a bootloader (CPU initialization + loading the kernel)

for Redpitaya (Zynq) or Raspberry Pi: 6–8 GB disk space resulting in a 200+ MB image in `output/images/sdcard.img`



- `/dev` holds all the communication interfaces between userspace and the kernel (**c**haracter devices and **b**lock devices...

... matching the Unix philosophy of "Everything is a file" [1]

[1] E.S. Raymond, *The Art of Unix Programming* (2003) and most significantly section 3 "The Elements of Operating-System Style"

Accessing hardware from userspace: /dev

Example of serial ports:

```
jmfriedt@dhcp-221:~$ ls -l /dev/ttyS*
crw-rw---- 1 root dialout 4, 64 Oct  8 18:43 /dev/ttyS0
crw-rw---- 1 root dialout 4, 65 Oct  8 18:43 /dev/ttyS1
crw-rw---- 1 root dialout 4, 66 Oct  8 18:43 /dev/ttyS2
crw-rw---- 1 root dialout 4, 67 Oct  8 18:43 /dev/ttyS3
```

Accessing serial ports (*terminal* handling):

```
int fd;
struct termios oldtio,newtio;
fd=open("/dev/ttyS0", O_RDWR | O_NOCTTY );
tcgetattr(fd,&oldtio); /* save current serial port settings */
newtio.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;  /* _no_ CRTSCTS */
tcsetattr(fd,TCSANOW,&newtio);
```

then

```
unsigned char cmd;
read(fd,&cmd,1);
printf"(%x) ",(cmd&0xff));fflush(stdout);
```

Command line tool for transfering data through the serial port:

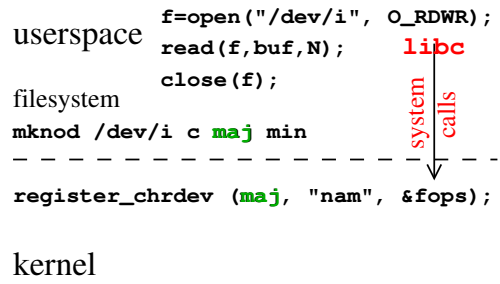
`stty -F /dev/ttyS0 9600 && cat < /dev/ttyS0` ou `minicom`

Accessing hardware from userspace

Through `/dev`

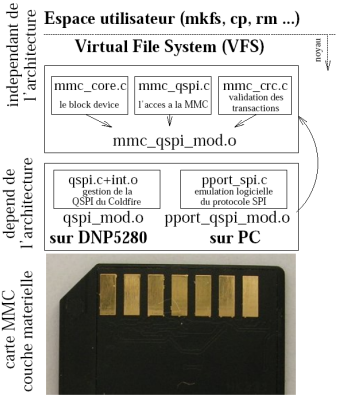
- system calls: opening, closing, writing, reading to a file but control (`ioctl`)
- sends the request to kernel module implementing each method
- each peripheral (file in userspace) is represented by a class (*major number*) and index (*minor number*) at the kernel level

```
brw-rw----  1 root     disk      8,   0 Feb 28 06:21 sda
brw-rw----  1 root     disk      8,   1 Feb 28 06:21 sda1
brw-rw----  1 root     disk      8,   2 Feb 28 06:21 sda2
brw-rw----  1 root     disk      8,   3 Feb 28 06:21 sda3
[...]
crw-rw----  1 root     dialout   4,  64 Feb 28 07:21 ttyS0
crw-rw----  1 root     dialout   4,  65 Feb 28 07:21 ttyS1
crw-rw----  1 root     dialout   4,  66 Feb 28 07:21 ttyS2
crw-rw----  1 root     dialout   4,  67 Feb 28 07:21 ttyS3
```



Lacking dynamic peripheral creation (`udev`) in `/dev`? manually create entry with `mknod`

Accessing hardware from userspace [2] [3]



- `block` (b) device ("files") behaves *from a user perspective* as `char` (c) device (*pipe*) [a]
- data transferts with hardware are handled with data blocks (buffer, cache) rather than at the byte level
- replace `file_operation` with `block_operation`
- transfers are handled through disk "geometry" defined in a `gendisk` structure.

`http://www.makelinux.net/ldd3/chp-16-sect-1`

[2] P. Ficheux, *Programmation noyau sous Linux : les pilotes en mode bloc*, GNU/Linux Magazine France, 109, pp.4-10 (Octobre 2008)

[3] S. Guinot, J.-M Friedt, *Stockage de masse non-volatile : un block device pour MultiMediaCard*, GNU/Linux Magazine France, Hors Série 25 (Avril 2006)

Practical demonstration

Raspberry Pi4: two LEDs (green and red), accessed by

1. `/sys/class/leds`
2. `/sys/class/gpio`
3. `devmem`
4. dedicated C program

Where are the LEDs? Check the Linux *devicetree* describing hardware in
`.../linux-custom/arch/arm/boot/dts/bcm2711-rpi-4-b.dts`

```
&leds {
        act_led: act {
                label = "led0";
                linux,default-trigger = "mmc0";
                gpios = <&gpio 42 GPIO_ACTIVE_HIGH>;
        };
        pwr_led: pwr {
                label = "led1";
                linux,default-trigger = "default-on";
                gpios = <&expgpio 2 GPIO_ACTIVE_LOW>;
        };
};
```

Accessing hardware from userspace

> **Caveat**: the Raspberry Pi4 is used for illustration for its affordability. It is arguably the **worst teaching platform** I have met !

Through `/sys/class`

- sharing commands as ASCII sentences
- configuring through the appropriate file entry (no `ioctl`)
- most appropriate for interacting with the user through `shell` or high abstraction languages (e.g Python)

```
$ cat /sys/class/rtc/rtc0/time
07:37:02
# cat /sys/class/backlight/intel_backlight/max_brightness
4500
# echo "1000" >  /sys/class/backlight/intel_backlight/brightness
```

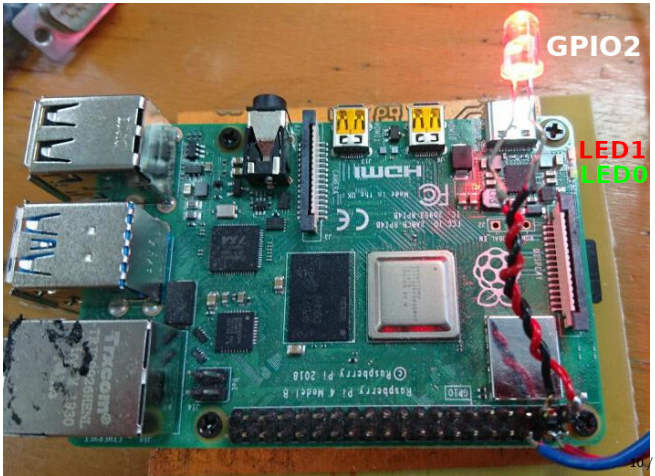
C access to `/sys/class`: make sure to `fseek` at offset 0 with `SEEK_SET` for multiple accesses without having to `fopen` and `fclose`

Accessing hardware: /sys/class/leds

`cat /sys/class/leds/led0/trigger`: default behaviour is to toggle the green LED when accessing the `mmc0` mass storage medium

```
echo "none" >  /sys/class/leds/led0/trigger
echo "1" > /sys/class/leds/led0/brightness
```

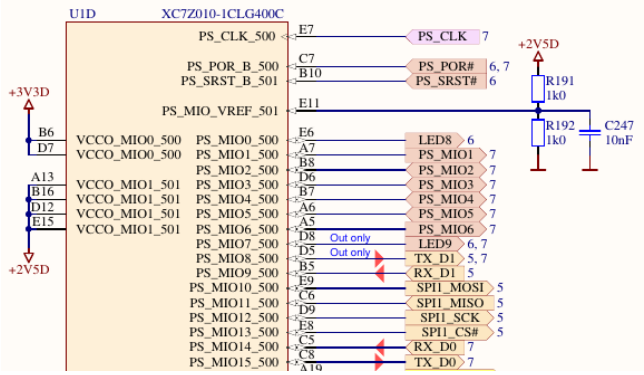
for manually controlling the LED
Same for LED1 and the red (power)
LED (default: "default-on" mode in dts).

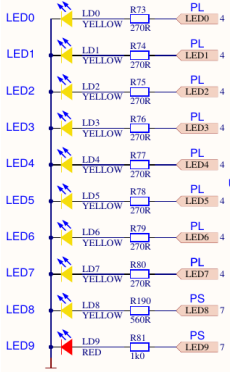


Accessing hardware: /sys/class/gpio on the Redpitaya

Through /sys/class/gpio

- check that the GPIO driver is loaded: gpio_zynq
- Zynq GPIO is defined with its index 906+MIO
- Request GPIO handling authorization: export (will create gpiox subdirectory)
- Set the GPIO as input or output (direction: "in" or "out")
- Set the GPIO state (brightness: 0 or 1)
- Accessing from a web server: *check access authorizations*





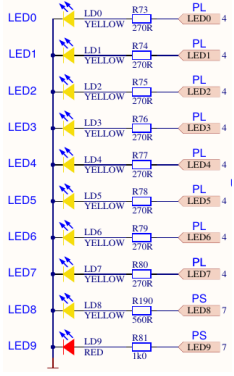
Accessing hardware: /sys/class/gpio on the Redpitaya

Through /sys/class/gpio

- check that the GPIO driver is loaded: gpio_zynq
- Zynq GPIO is defined with its index 906+MIO
- Request GPIO handling authorization: export (will create gpiox subdirectory)
- Set the GPIO as input or output (direction: "in" or "out")
- Set the GPIO state (brightness: 0 or 1)
- Accessing from a web server: *check access authorizations*

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

FPGA con-nector	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)



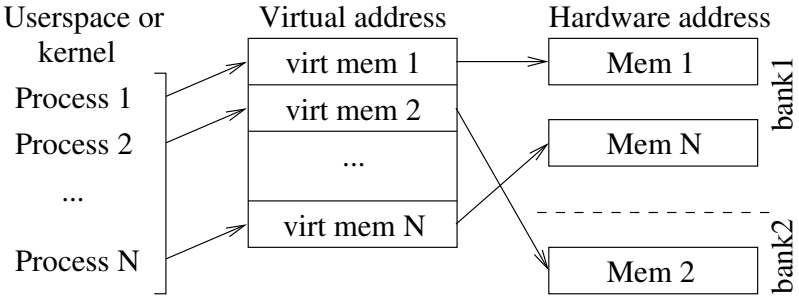
Virtual memory/hardware memory address

Hardware address space

- hardware address: value on the address bus to identify a peripheral
- each peripheral decodes the address bus to check if the message is targeted to this interface
- only one peripheral associated with each hardware address (otherwise, conflict)

Virtual memory

- each process has the same address space
- memory organization independent of hardware requirements
- MMU (Memory Management Unit): translation between virtual and hardware memory



devmem: access hw memory @ (Raspberry Pi 4)

```
cd /sys/class/gpio/
echo 2 > export
cd gpio2/
echo out > direction
devmem 0xfe200000 32 0x044
echo in > direction
devmem 0xfe200000 32 0x004
```

to check that accessing the `direction` method indeed changes the associated register

GPIO 2 is next to power supply, expansion port

```
devmem 0xfe200000 32 0x44 $ output
devmem 0xfe200028 32 0x04 # set
devmem 0xfe20001c 32 0x04 # clear
```

and for the green LED (GPIO 42)

```
echo none > /sys/class/leds/led0/trigger
devmem 0xfe200010 32 # SEL4, 42=6..8
# 0x00000064
devmem 0xfe200020 32 0x400 # SET1=20
devmem 0xfe20002c 32 0x400 # CLR1=2C
```

5.2. Register View

The GPIO has the following registers. All accesses are assumed to be 32-bit. The GPIO register base address is `0x7E21 5000`.

Address Offset	Register Name	Description	Size
0x00	GPFSEL0	GPIO Function Select 0	32
0x04	GPFSEL1	GPIO Function Select 1	32
0x08	GPFSEL2	GPIO Function Select 2	32
0x0C	GPFSEL3	GPIO Function Select 3	32
0x10	GPFSEL4	GPIO Function Select 4	32
0x14	GPFSEL5	GPIO Function Select 5	32
0x18	-	Reserved	-
0x1C	GPSET0	GPIO Pin Output Set 0	32
0x20	GPSET1	GPIO Pin Output Set 1	32
0x24	-	Reserved	-
0x28	GPCLR0	GPIO Pin Output Clear 0	32
0x2C	GPCLR1	GPIO Pin Output Clear 1	32
0x30	-	Reserved	-
0x34	GPLEV0	GPIO Pin Level 0	32
0x38	GPLEV1	GPIO Pin Level 1	32

17:15	FSEL5	FSEL5 - Function Select 5	RW	0
14:12	FSEL4	FSEL4 - Function Select 4	RW	0
11:9	FSEL3	FSEL3 - Function Select 3	RW	0
8:6	FSEL2	FSEL2 - Function Select 2	RW	0
5:3	FSEL1	FSEL1 - Function Select 1	RW	0
2:0	FSEL0	FSEL0 - Function Select 0	RW	0

Table 62. GPIO Alternate function select register 0

GPFSEL1 Register

Bit(s)	Field Name	Description	Type	Reset
31:30		Reserved - Write as 0, read as don't care		
29:27	FSEL19	FSEL19 - Function Select 19 000 = GPIO Pin 19 is an input 001 = GPIO Pin 19 is an output 100 = GPIO Pin 19 takes alternate function 0 101 = GPIO Pin 19 takes alternate function 1 110 = GPIO Pin 19 takes alternate function 2 111 = GPIO Pin 19 takes alternate function 3 011 = GPIO Pin 19 takes alternate function 4 010 = GPIO Pin 19 takes alternate function 5	RW	0
26:24	FSEL18	FSEL18 - Function Select 18	RW	0
23:21	FSEL17	FSEL17 - Function Select 17	RW	0

devmem: access hw memory @ (Raspberry Pi 4)

```
cd /sys/class/gpio/
echo 2 > export
cd gpio2/
echo out > direction
devmem 0xfe200000 32 0x044
echo in > direction
devmem 0xfe200000 32 0x004
```

to check that accessing the `direction` method indeed changes the associated register

GPIO 2 is next to power supply, expansion port

```
devmem 0xfe200000 32 0x44 $ output
devmem 0xfe200028 32 0x04 # set
devmem 0xfe20001c 32 0x04 # clear
```

and for the green LED (GPIO 42)

```
echo none > /sys/class/leds/led0/trigger
devmem 0xfe200010 32 # SEL4, 42=6..8
# 0x00000064
devmem 0xfe200020 32 0x400 # SET1=20
devmem 0xfe20002c 32 0x400 # CLR1=2C
```

8:6	FSEL32	FSEL32 - Function Select 32	RW	0
5:3	FSEL31	FSEL31 - Function Select 31	RW	0
2:0	FSEL30	FSEL30 - Function Select 30	RW	0

Table 65. GPIO Alternate function select register 3

GPFSEL4 Register

Bit(s)	Field Name	Description	Type	Reset
31:30		Reserved - Write as 0, read as don't care		
29:27	FSEL49	FSEL49 - Function Select 49 000 = GPIO Pin 49 is an input 001 = GPIO Pin 49 is an output 100 = GPIO Pin 49 takes alternate function 0 101 = GPIO Pin 49 takes alternate function 1 110 = GPIO Pin 49 takes alternate function 2 111 = GPIO Pin 49 takes alternate function 3 011 = GPIO Pin 49 takes alternate function 4 010 = GPIO Pin 49 takes alternate function 5	RW	0
26:24	FSEL48	FSEL48 - Function Select 48	RW	0
23:21	FSEL47	FSEL47 - Function Select 47	RW	0
20:18	FSEL46	FSEL46 - Function Select 46	RW	0
17:15	FSEL45	FSEL45 - Function Select 45	RW	0

31:0	SETn (*n*=0..31)	0 = No effect 1 = Set GPIO pin *n*	WO	0

Table 68. GPIO Output Set Register 0

GPSET1 Register

Bit(s)	Field Name	Description	Type	Reset
31:26		Reserved - Write as 0, read as don't care		
25:0	SETn (*n*=32..57)	0 = No effect 1 = Set GPIO pin *n*.	WO	0

Table 69. GPIO Output Set Register 1

GPCLR0 Register

Synopsis

The output clear registers are used to clear a GPIO pin. The CLR*n* field defines the respective GPIO pin to clear, writing a "0" to the field has no effect. If the GPIO pin is being used as an input (by default) then the value in the CLR*n* field is ignored. However, if the pin is subsequently defined as an output then the bit will be set according to the last set/clear operation. Separating the set and clear functions removes the need for read-modify-write operations.

Bit(s)	Field Name	Description	Type	Reset
31:0	CLRn (*n*=0..31)	0 = No effect 1 = Clear GPIO pin *n*	WO	0

Hw interaction through `/dev/mem`, the MMU interface (see `busybox-1.27.1/miscutils/devmem.c`)

- benefit: no interaction with the kernel (fast)
- drawback: no interaction with the kernel (no consistent management of resource access)

```c
#include <fcntl.h>
#include <sys/mman.h>
#define MAP_SIZE 4096UL
#define MAP_MASK (MAP_SIZE - 1)

int main(int argc, char **argv) {
 int fd; void *map_base, *virt_addr; unsigned long read_result, writeval;
 off_t target;
 int access_type = 'w'; // Args : { address } [ type [ data ] ]
 target = strtoul(argv[1], 0, 0);
 if(argc > 2) access_type = tolower(argv[2][0]);
 fd = open("/dev/mem", O_RDWR | O_SYNC);
 map_base = mmap(0, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, target & ~MAP_MASK);
 virt_addr = map_base + (target & MAP_MASK);
 switch(access_type) { case 'b': read_result = *((unsigned char *) virt_addr); break;
                       case 'h': read_result = *((unsigned short *)virt_addr); break;
                       case 'w': read_result = *((unsigned long *) virt_addr); break;
 }
 printf("Value at address 0x%X (%p): 0x%X\n", target, virt_addr, read_result);
 if(argc > 3) {
  writeval = strtoul(argv[3], 0, 0);
  switch(access_type) { case 'b': *((unsigned char *) virt_addr) = writeval; break;
                        case 'h': *((unsigned short *)virt_addr) = writeval; break;
                        case 'w': *((unsigned long *) virt_addr) = writeval; break;
              }       }
 munmap(map_base, MAP_SIZE); close(fd); return 0;
}
```

Register access in C

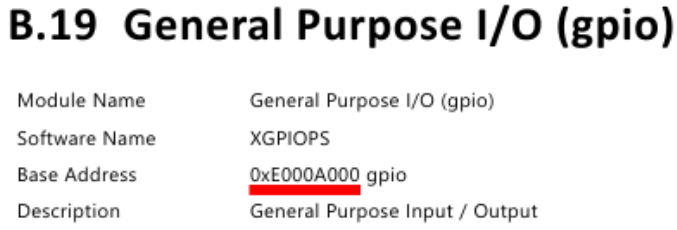
Case of RPi: from https://www.raspberrypi.org/forums/viewtopic.php?t=244031

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <unistd.h>

#define BCM2711_PERI_BASE        0xFE000000
#define GPIO_BASE                (BCM2711_PERI_BASE + 0x200000) /* GPIO controller */
#define PAGE_SIZE (4*1024)
#define BLOCK_SIZE (4*1024)
#define INP_GPIO(g) *(gpio+((g)/10)) &= ~(7<<(((g)%10)*3))
#define OUT_GPIO(g) *(gpio+((g)/10)) |=  (1<<(((g)%10)*3))

#define GPIO_SET *(gpio+7)  // sets   bits which are 1 ignores bits which are 0
#define GPIO_CLR *(gpio+10) // clears bits which are 1 ignores bits which are 0

int main(int argc, char **argv)
{unsigned int *gpio;
 int g,rep;
 int mem_fd = open("/dev/mem", O_RDWR|O_SYNC);
 gpio = mmap(
      NULL,             //Any adddress in our space will do
      PAGE_SIZE,        //Map length
      PROT_READ|PROT_WRITE,// Enable reading & writing to mapped memory
      MAP_SHARED,       //Shared with other processes
      mem_fd,           //File to map
      GPIO_BASE         //Offset to GPIO peripheral
   );
  INP_GPIO(2); // must use INP_GPIO before we can use OUT_GPIO
  OUT_GPIO(2);
  GPIO_SET = 1<<2;
  return 0;
}
```

Register description

Zynq 70xx datasheet [4] chapitre 14 (GPIO)

- 4 GPIO banks (32 bits)
- GPIO control register offset with respect to base address 0xEA00A000
- GPIO activation procedure detailed in section 14.3 of *Programming Guide*
- each register is described in appendix B



B.19 General Purpose I/O (gpio)

Module Name	General Purpose I/O (gpio)
Software Name	XGPIOPS
Base Address	0xE000A000 gpio
Description	General Purpose Input / Output

- as found on microcontrollers: enable, output, data

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

Register description

Beware: the **clock** of the GPIO bank must be activated

- global configurations start at 0xF8000000 (*System Level Control Registers*)

14.3.1 Start-up Sequence

Main Example: Start-up Sequence

1. **Resets:** The reset options are described in section 14.4.2 Resets.
2. **Clocks:** The clocks are described in section 14.4.1 Clocks.
3. **GPIO Pin Configurations:** Configure pin as input/output is described in section 14.3.2 GPIO Pin Configurations.
4. **Write Data to GPIO Output pin:** Refer to example in section 14.3.3 Writing Data to GPIO Output Pins.
5. **Read Data from GPIO Input pin:** Refer to example in section 14.3.4 Reading Data from GPIO Input Pins .
6. **Set GPIO pin as wake-up event**: Refer to example in section GPIO as Wake-up Event.

14.3.2 GPIO Pin Configurations

Each individual GPIO pin can be configured as input/output. However, bank0 [8:7] pins must be configured as outputs. Refer to section 14.2.3 Bank0, Bits[8:7] are Outputs for further details.

Example: Configure MIO pin 10 as an output

1. **Set the direction as output:** Write `0x0000_0400` to the gpio.DIRM_0 register.

Register (slcr) APER_CLK_CTRL

Name	APER_CLK_CTRL
Relative Address	0x0000012C
Absolute Address	0xF800012C
Width	32 bits
Access Type	rw
Reset Value	0x01FFCCCD
Description	AMBA Peripheral Clock Control

Register APER_CLK_CTRL Details

Please note that these clocks must be enabled if you want to read from the peripheral register space.

Field Name	Bits	Type	Reset Value	Description
reserved	31:25	rw	0x0	Reserved. Writes are ignored, read data is zero.
SMC_CPU_1XCLKACT	24	rw	0x1	SMC AMBA Clock control 0: disable, 1: enable
LQSPI_CPU_1XCLKACT	23	rw	0x1	Quad SPI AMBA Clock control 0: disable, 1: enable
GPIO_CPU_1XCLKACT	22	rw	0x1	GPIO AMBA Clock control 0: disable, 1: enable

- in case of failure, check the register configuration lock

Register description

Beware: the **clock** of the GPIO bank must be activated

- global configurations start at 0xF8000000 (*System Level Control Registers*)

Register SLCR_LOCK Details

Field Name	Bits	Type	Reset Value	Description
reserved	31:16	wo	0x0	Reserved. Writes are ignored, read data is zero.
LOCK_KEY	15:0	wo	0x0	Write the lock key, 0x767B, to write protect the slcr registers: all slcr registers, 0xF800_0000 to 0xF800_0B74, are write protected until the unlock key is written to the SLCR_UNLOCK register. A read of this register returns zero.

Register (slcr) SLCR_UNLOCK

Name	SLCR_UNLOCK
Relative Address	0x00000008
Absolute Address	0xF8000008
Width	32 bits
Access Type	wo
Reset Value	0x00000000
Description	SLCR Write Protection Unlock

Register SLCR_UNLOCK Details

Field Name	Bits	Type	Reset Value	Description
reserved	31:16	wo	0x0	Reserved. Writes are ignored, read data is zero.
UNLOCK_KEY	15:0	wo	0x0	Write the unlock key, 0xDF0D, to enable writes to the slcr registers. All slcr registers, 0xF800_0000 to 0xF800_0B74, are writeable until locked using the SLCR_LOCK register. A read of this register returns zero.

Register (slcr) APER_CLK_CTRL

Name	APER_CLK_CTRL
Relative Address	0x0000012C
Absolute Address	0xF800012C
Width	32 bits
Access Type	rw
Reset Value	0x01FFCCCD
Description	AMBA Peripheral Clock Control

Register APER_CLK_CTRL Details

Please note that these clocks must be enabled if you want to read from the peripheral register space.

Field Name	Bits	Type	Reset Value	Description
reserved	31:25	rw	0x0	Reserved. Writes are ignored, read data is zero.
SMC_CPU_1XCLKACT	24	rw	0x1	SMC AMBA Clock control 0: disable, 1: enable
LQSPI_CPU_1XCLKACT	23	rw	0x1	Quad SPI AMBA Clock control 0: disable, 1: enable
GPIO_CPU_1XCLKACT	22	rw	0x1	GPIO AMBA Clock control 0: disable, 1: enable

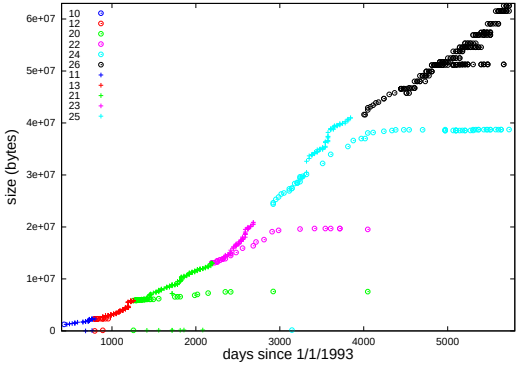
- in case of failure, check the register configuration lock

Kernel & kernel modules

- Developper $\Rightarrow$ focus on the software-hardware interface
- Only the kernel can access all resources of the processor (e.g. interrupts, DMA)
- The kernel shares resources and make sure hardware access is consistent (kernel = supervisor)
- Adding new functionalities to the kernel: modules ("plugins")

Porting GNU/Linux to a new architecture

- a (cross-)compilation toolchain
- define the memory map of the target (`-T` option of `ld`)
- bootloader development (peripheral initialization and loading the kernel to memory)[5]
- developing kernel modules for handling hardware dedicated to the new architecture [6].

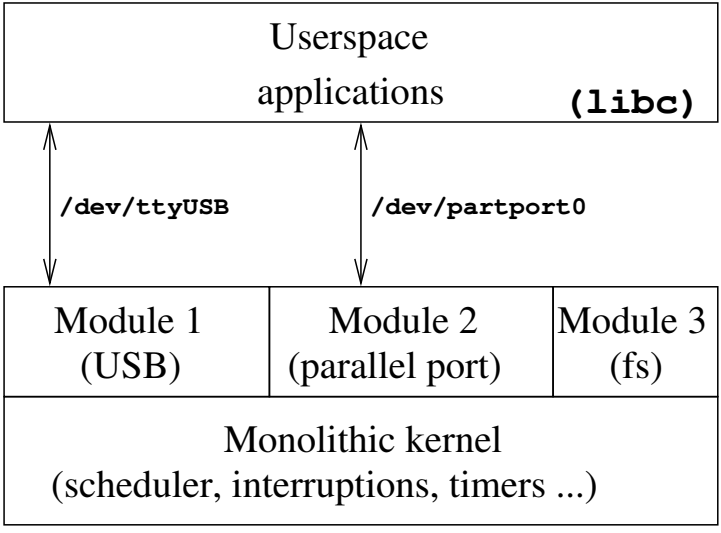


Evolution of the Linux kernel `.tar.gz` archive size (`www.kernel.org`)

[5] S. Guinot, J.-M Friedt, *GNU/Linux sur Playstation Portable*, GNU/Linux Magazine France **114**, Mars 2009

[6] J. Corbet, A. Rubini & G. Kroah-Hartman, *Linux Device Drivers, 3rd Edition*, O'Reilly (2005), available at à `http://lwn.net/Kernel/LDD3/`

Operating System (OS) architecture



system functions, network, scheduling and memory handling, vfs

The /dev directory

- Pseudo-files linking userspace and kernelspace through system calls implemented in `libc` (`man syscalls`)
- each *device* is defined with a *major number* and, within each class, a *minor number*
- the kernel is not aware of filesystems: on the kernel side, a driver is associated with a given major number and handles each syscall:
  - `open()`
  - `close()`
  - `write()`
  - `read()`
  - `ioctl()`

```
static struct file_operations qadc_fops =
{
        owner:          THIS_MODULE,
        read:           qadc_read,
        open:           qadc_open,
        ioctl:          qadc_ioctl,
        release:        qadc_release,
};

static int __init qadc_init (void)
{...
 register_chrdev (qadc_major, "ppp", &qadc_fops);
 ...}

module_init (qadc_init);
module_exit (qadc_exit);
```

Basic structure

Kernel module ⇒ no `main()` starting function:

- one entry and one exit function:
  `module_init (my_init_function);` and
  `module_exit (my_exit_function);`
- initializing the communication link with userspace: `register_chrdev (90 , "jmf" ,&fops );`
- information displayed through `/var/log/syslog` using
  `printk (KERN_ALERT "formatted message");` (*no coma !*)
- ability to dynamically create dev entries using *misc. devices*

```
int hello_start()  // init_module(void)
{jmfdev.name = "jmf";  // /dev/jmf: major = classe misc
 jmfdev.minor = MISC_DYNAMIC_MINOR;
 jmfdev.fops = &fops;
 misc_register(&jmfdev); // dynamic creation of /dev/jmf
}
```

Module handling commands

Modules (*kernel object*) with `.ko` extension

- are linked to the kernel with `insmod toto.ko`,
- are removed from the kernel with `rmmod toto`.
- list of attached modules with `lsmod`
- Messages logged by modules using `printk` are displayed with `dmesg`
- "Official" modules are located in `/lib/modules`
- "Official" modules with dependencies are loaded with `modprobe` [7]

[7] see `/lib/modules/version/modules.dep`

Directories needed for compiling modules

The sources of the kernel running on the PC are located in `/usr/src`:

- Debian/GNU Linux separates generic headers from parts specific to a given architecture
- `/usr/src/linux-headers-*-common` does *not* includes parts specific to the current architecture
- `/usr/src/linux-headers-*-*arch` is the directory we use to find the specific configuration of the kernel running on the host.

New `Makefile` architecture for compiling modules relying on the kernel `Makefile` by adding the new kernel target to the `obj-m` variable with

```
obj-m +=0mymod.o 1mymod.o 2mymod.o
all:
        make -C /usr/src/linux-headers-4.8.0-1-amd64 M=$(PWD) modules
```

For cross-compiling to the target architecture: `output/build/linux-*` in Buildroot provides the Linux kernel

```
obj-m +=0mymod.o 1mymod.o 2mymod.o
all:
        make ARCH=arm CROSS_COMPILE=arm-linux- -C \
    ../buildroot-2023.08-rc3/output/build/linux-xilinx-v2022.1/ M=$(PWD) modules
```

Real v.s virtual addresses

Same problem as `/dev/mem` and `mmap` in userspace: here too the MMU must provide the virtual block address translated to the physical address:

```
request_mem_region (IO_BASE, 36*9, "GPIO test" );// PG9
jmf_gpio = ( u32 )ioremap(IO_BASE, 36*9);       // 36 bytes/port, ports A-I
writel (0x10, jmf_gpio+36*6+0x04 );             // GPIO PG9 en sortie
```

30.2. Port Register List

Module Name	Base Address
PIO	0x01C20800

Register Name	Offset	Description
Pn_CFG0	n*0x24+0x00	Port n Configure Register 0 (n from 0 to 9)
Pn_CFG1	n*0x24+0x04	Port n Configure Register 1 (n from 0 to 9)
Pn_CFG2	n*0x24+0x08	Port n Configure Register 2 (n from 0 to 9)
Pn_CFG3	n*0x24+0x0C	Port n Configure Register 3 (n from 0 to 9)
Pn_DAT	n*0x24+0x10	Port n Data Register (n from 0 to 9)
Pn_DRV0	n*0x24+0x14	Port n Multi-Driving Register 0 (n from 0 to 9)
Pn_DRV1	n*0x24+0x18	Port n Multi-Driving Register 1 (n from 0 to 9)
Pn_PUL0	n*0x24+0x1C	Port n Pull Register 0 (n from 0 to

Addresses given in the microprocessor datasheet (here Allwinner A10)

Methods provided by the kernel

Many functionalities provided by the kernel for handling tasks (scheduler), timer or communicating with userspace.

- concepts close to microcontroller programming

```
init_timer_on_stack (&exp_timer );
exp_timer.expires = jiffies+delay*HZ ;
exp_timer.data = 0;
exp_timer.function = do_something ;
add_timer (&exp_timer);
```

- *tasklet*: software interrupts

```
char my_tasklet_data[]="my_tasklet_function was called";
void my_tasklet_function( unsigned long data )
{ printk( "%s\n", (char *)data ); return; }

DECLARE_TASKLET( my_tasklet, my_tasklet_function,
   (unsigned long) &my_tasklet_data );

int init_module( void )
{ tasklet_schedule( &my_tasklet ); return 0; }

void cleanup_module( void )
{ tasklet_kill( &my_tasklet ); return; }
```

→ ISR should be **short** and **schedule** a tasklet

Kernel module: interrupt handling

Only a kernel module can handle interrupts (e.g. interrupt 7 is associated with pin 10 of the parallel port on a PC)

```
jmfriedt@(none):~$ cat /proc/interrupts | grep parp
  7:          0    IO-APIC-edge  parport0
```

handled with

```
#define PARALLEL_PORT_INTERRUPT 7
#define BASEPORT                0x378
int s=0;
static int int_handler(void) {printk(">>> PARALLEL PORT INT\n");s=1;}
static int __init jmfpport_init_module (void) {
  ret = request_irq(PARALLEL_PORT_INTERRUPT, &int_handler,
                    SA_INTERRUPT, "parallelport", NULL);
  enable_irq(7);
  outb_p(0x10, BASEPORT + 2);
}
static ssize_t skeleton_read (struct file *file, char *buf,
                size_t count, loff_t *ppos) {
 [...]
 do {} while (sortir==0);    // interruptible_sleep_on(&skeleton_wait);
 err = copy_to_user(buf,string,counter);
 ...
}
```

Select an OS wisely

- an OS must boot: takes time and hence energy
- an OS requires learning new APIs
- an OS requires memory and computational power
- challenge of long term maintenance when relying on someone else's code

$\Rightarrow$ consider the benefits (libraries, external contribution, stable tools, networking) over these drawbacks before selecting an OS.

Practical application

Discover the Zynq 7010 of the Redpitaya[8]

Accessing hardware from userspace ... [9]

- ... from the shell through `/sys/class/gpio` [10] : `Device Drivers` → `GPIO Support`
- ... from the sell through `devmem` to handle processor registers
- compile a C program using the *buildroot* toolchain: PATH must include `$BR/output/host/usr/bin`
- demonstrate accessing registers from a userspace C program accessing `/dev/mem` (mask with address with pagesize of MMU)
- Kernel module for Linux
  - Converting from virtual to real memory in kernel space
  - Dynamic communication entry creation in `/dev`
  - Reading a datasheet (hardware addresses) remains mandatory !
  - GNU License to be allowed to access the functionalities provided by the kernel or other modules (hardware abstraction)

→ become familiar with hardware handling with the GNU/Linux overhead in order to prepare for kernel device driver by convertin the userspace C blinking LED example to a kernel module

[8] `redpitaya.readthedocs.io/en/latest/developerGuide/125-14/shem.html`

[9] `http://jmfriedt.free.fr/redpitaya_eng.pdf`

[10] `https://www.kernel.org/doc/Documentation/gpio/sysfs.txt`