Systèmes embarqués 6

G. Goavec-Mérou, J.-M Friedt

FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

21 septembre 2017

Device

Le device ne fait qu'instancier le matériel mais n'est pas en charge de créer les fichiers de communication (une série de fichiers par *driver*)

```
static struct resource jmf_resources1[] = {
        {.start  = 0x41200000,
         .end    = 0x412000ff,
         .flags  = IORESOURCE_MEM, // PAS IORESOURCE_MEM
         .name   = "io-led1"
        },
        {.start  = 0x41210000,
         .end    = 0x412100ff,
         .flags  = IORESOURCE_MEM, // PAS IORESOURCE_MEM
         .name   = "io-led2"
        },
    };

static struct platform_device *pdev1,*pdev2;

static int __init gpio_simple_init(void)
{pdev1=platform_device_register_simple("jmf",0,jmf_resources1,ARRAY_SIZE(jmf_resources1));
 pdev2=platform_device_register_simple("jmf",1,jmf_resources1,ARRAY_SIZE(jmf_resources1));
 return 0;
}

static void __exit gpio_simple_exit(void)
{platform_device_unregister(pdev1);
 platform_device_unregister(pdev2);
}
module_init(gpio_simple_init)
module_exit(gpio_simple_exit)
MODULE_LICENSE("GPL");
```

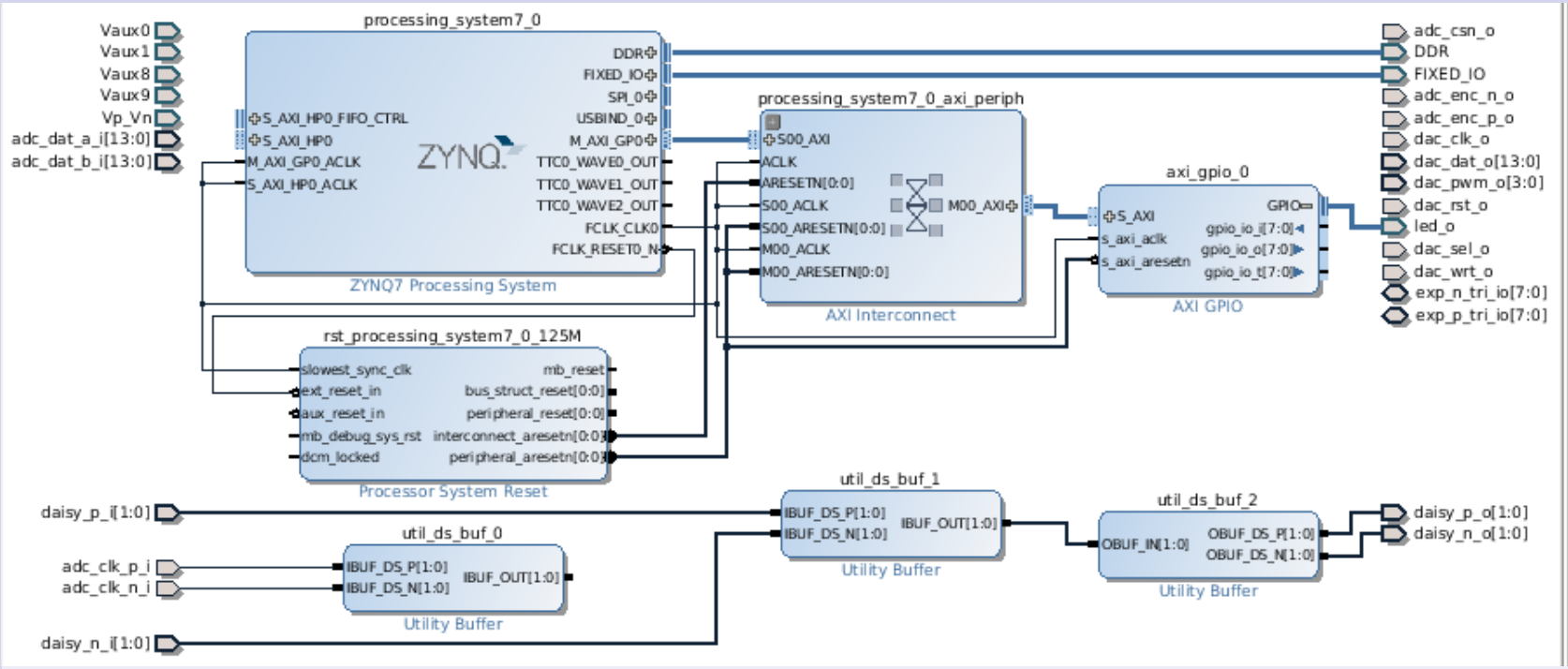
```
redpitaya> insmod dr_comm.ko                          driver           modalias         subsystem        valuer
redpitaya> insmod pl_resource2.ko                     driver_override  power            uevent           valuew
jmf hello 0
jmf hello 1                                           /sys/bus/platform/devices/jmf.1:
redpitaya> ls /sys/bus/platform/devices/jmf.*         driver           modalias         subsystem        valuer
/sys/bus/platform/devices/jmf.0:                      driver_override  power            uevent           valuew
```

Driver

```
static ssize_t gpio_simple_store(struct device *dev, struct device_attribute *attr, const
      ↪char *buf, size_t count)
{struct jmf_struct* ma_struct=dev_get_drvdata(dev);
 long int res;
 if (!kstrtol(buf,10,&res)) printk("%s = %d",buf,res);
 return(count);
}

static DEVICE_ATTR(valuew, 0220, NULL, gpio_simple_store);

static int gpio_simple_remove(struct platform_device *pdev)
{device_remove_file(&pdev->dev, &dev_attr_valuew);
 printk(KERN_ALERT "jmf bye %d\n",pdev->id);return 0;
}

static int gpio_simple_probe(struct platform_device *pdev)
{struct resource *r;
 void *m;
 device_create_file(&pdev->dev, &dev_attr_valuew);
 r= platform_get_resource(pdev, IORESOURCE_MEM, pdev->id);
 printk(KERN_ALERT "jmf hello %d:%x--%x\n",pdev->id,(int)r->start,(int)r->end);
 m=ioremap(r->start, resource_size(r));  // on recupere end-start
 writel(0x0,m+0x04); // all as output (0 = output !)
 writel(0x05,m);     // all as output
 return 0;
}

static struct platform_driver gpio_simple_driver = {
        .probe          = gpio_simple_probe,
        .remove         = gpio_simple_remove,
        .driver = {.name   = "jmf",},
};
module_platform_driver(gpio_simple_driver);
MODULE_LICENSE("GPL");
```

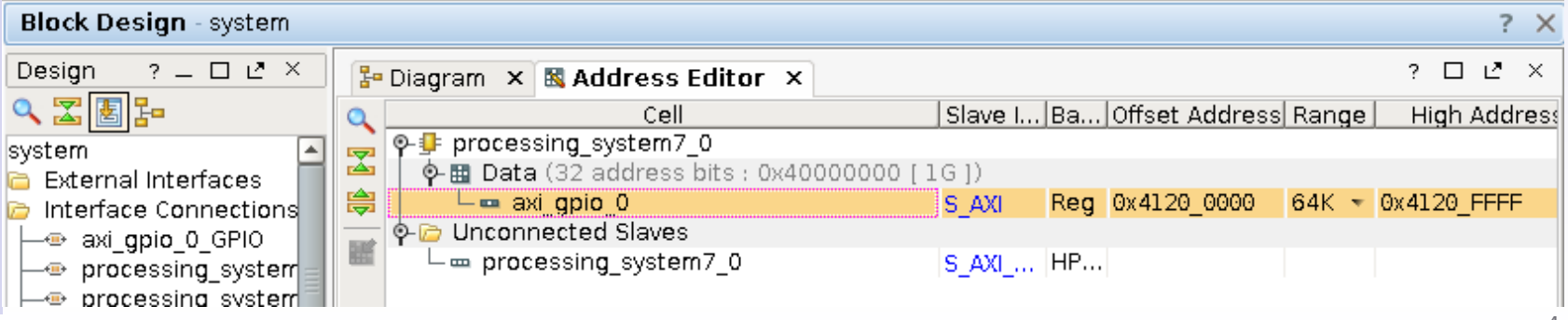
FPGA

Xilinx Vivado pour configurer le FPGA (28 GB, penser à installer SDK)



PS (Zynq)+bus AXI+GPIO donne accès aux EMIO

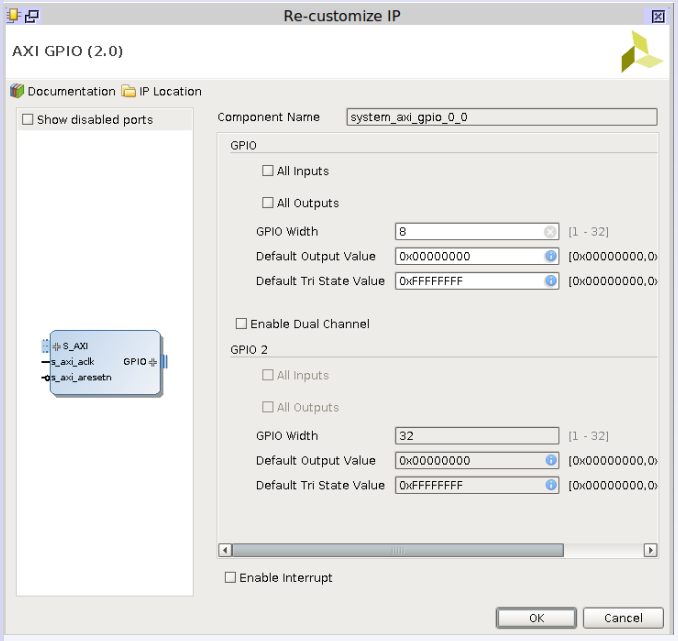
Vivado impose les adresses sur bus AXI des divers périphériques



FPGA : EMIO

EMIO : GPIO connectés au PL ($\neq$ MIO connectés au PS)

Rôle des registres du bloc GPIO : [1] :



Register Space

Note: The AXI4-Lite write access register is updated by the 32-bit AXI Write Data (*_wdata) signal, and is not impacted by the AXI Write Data Strobe (*_wstrb) signal. For a Write, both the AXI Write Address Valid (*_awvalid) and AXI Write Data Valid (*_wvalid) signals should be asserted together. Also see Answer Record 4127.

Table 2-4 shows the AXI GPIO registers and their addresses.

Table 2-4: **Registers**

Address Space Offset(3)	Register Name	Access Type	Default Value	Description
0x0000	GPIO_DATA	R/W	0x0	Channel 1 AXI GPIO Data Register.
0x0004	GPIO_TRI	R/W	0x0	Channel 1 AXI GPIO 3-state Control Register.
0x0008	GPIO2_DATA	R/W	0x0	Channel 2 AXI GPIO Data Register.
0x000C	GPIO2_TRI	R/W	0x0	Channel 2 AXI GPIO 3-state Control.
0x011C	GIER(1)	R/W	0x0	Global Interrupt Enable Register.
0x0128	IP IER(1)	R/W	0x0	IP Interrupt Enable Register (IP IER).
0x0120	IP ISR(1)	R/TOW(2)	0x0	IP Interrupt Status Register.

1. https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

Programmer le FPGA depuis GNU/Linux (méthode unifiée) : fpga_manager

- passer d'un `.bit` à un `.bin` : fournir un fichier de configuration `.bif`

  ```
  all: {nom_du_bitstream.bit}
  ```

- `bit` → `bit.bin`

```
$VIVADO_SDK/bin/bootgen -image file.bif -arch zynq -process_bitstream bin
```

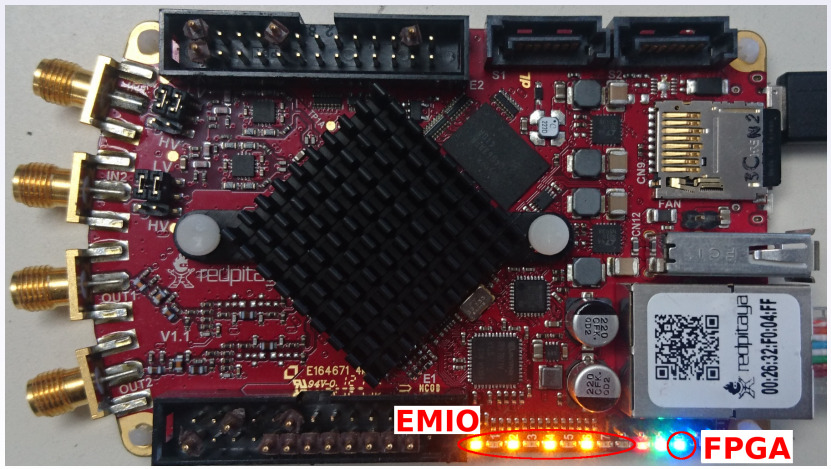
- flasher le FPGA avec `fpga_manager`

  ```
  cp system_wrapper.bit.bin /lib/firmware/system_wrapper.bit.bin
  echo "system_wrapper.bit.bin" > /sys/class/fpga_manager/fpga0/firmware
  ```

- Profiter du nouveau bitstream dans le FPGA

  ```
  devmem 0x41200004 w 0x0
  devmem 0x41200000 w 0xf
  ```

- ATTENTION : accéder au FPGA alors qu'aucun bitstream ne configure le FPGA se traduit par un crash de Linux (transaction AXI non-acquittée)



fpga_manager

Démonstration avec un driver pour commander les 8 LEDs depuis la méthode `init`

```
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/io.h>                   // ioremap

static struct platform_driver gpio_simple_driver = {
        .probe          = gpio_simple_probe,
        .remove         = gpio_simple_remove,
        .driver = {.name   = "jmf",},
};

static int gpio_simple_remove(struct platform_device *pdev)
{printk(KERN_ALERT "jmf bye %d\n",pdev->id);return 0;}

static int gpio_simple_probe(struct platform_device *pdev)
{struct resource *r;
 static void __iomem *jmf_gpio;    //int jmf_gpio;
 r= platform_get_resource(pdev, IORESOURCE_MEM, pdev->id); // MEM et non IO
 printk(KERN_ALERT "jmf hello %d:%x--%x\n",pdev->id,(int)r->start,(int)r->end);
 jmf_gpio=(void *)ioremap(r->start, 0x04);

 writel(0x0,jmf_gpio+0x04); // all as output
 writel(0x05,jmf_gpio); // all as output
 return 0;
}
module_platform_driver(gpio_simple_driver);
MODULE_LICENSE("GPL");
```

Comment commander les LEDs depuis un fichier en écriture

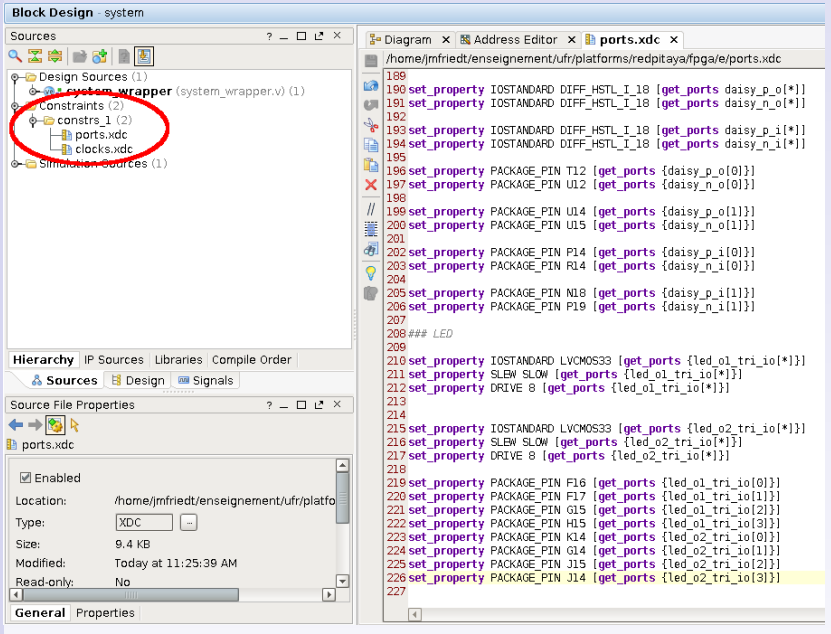
Structure privée duppliquant les ressources occupées par le pilote

```
struct jmf_struct {void* mem_base;};

static ssize_t gpio_simple_store(struct device *dev, struct device_attribute *attr, const →
      ↪char *buf, size_t count)
{struct jmf_struct* ma_struct=dev_get_drvdata(dev);
 long int res;
 if (!kstrtol(buf,10,&res)) writel(res,ma_struct->mem_base); // all as output
 printk("%s = %d",buf,res);
 return(count);
}

static DEVICE_ATTR(valuew, 0220, NULL, gpio_simple_store);

static int gpio_simple_remove(struct platform_device *pdev)
{struct jmf_struct* ma_struct=platform_get_drvdata(pdev);
 kfree(ma_struct);
 device_remove_file(&pdev->dev, &dev_attr_valuew);
 printk(KERN_ALERT "jmf bye %d\n",pdev->id);return 0;
}

static int gpio_simple_probe(struct platform_device *pdev)
{struct resource *r;
 struct jmf_struct *ma_struct;
 ma_struct=(struct jmf_struct *)kzalloc(sizeof(struct jmf_struct),GFP_KERNEL); // z=zero
 device_create_file(&pdev->dev, &dev_attr_valuew);
 r= platform_get_resource(pdev, IORESOURCE_MEM, pdev->id);
 ma_struct->mem_base=ioremap(r->start, resource_size(r)); // on recupere end-start
 platform_set_drvdata(pdev,ma_struct);
 return 0;
}
```

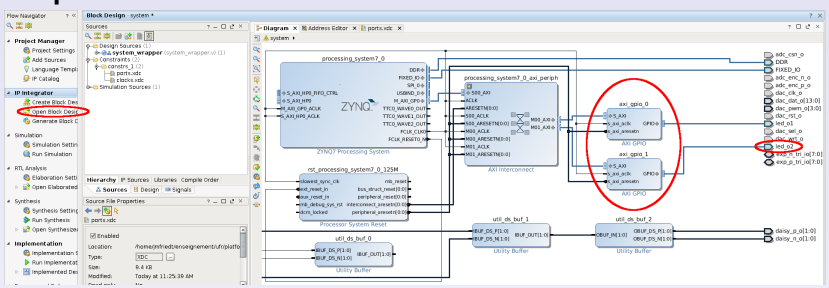
Démonstration sur plusieurs instances du pilote

Application pour justifier de deux configurations des ressources :

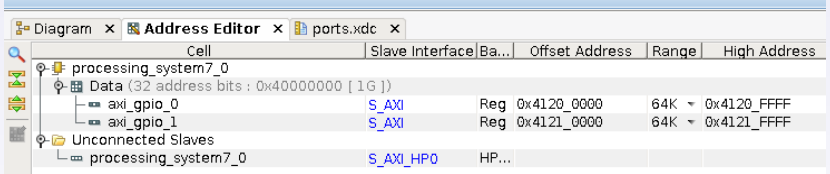
- diviser le latch GPIO en deux sous ensembles de 4 bits
- *autorouter*
- fichier de contrainte après avoir renommé `gpio_rtl` en `ledo_2`



Séparation du GPIO en 2 latchs de 4 bits



Plages d'adresses proposées par Vivado



Nature et assignation des broches

Exercice : re-écrire le pilote pour s'approprier deux configurations possibles, selon que ce soit led0 ou led1

Démonstration du passage de paramètre (motif des LEDs) par `/sys`

Exercice pour le 1er Janvier 2018

1. ajouter le bloc `XADC` au design Vivado contenant le GPIO
2. synthétiser et flasher ce bitstream : quelles broches sont associées aux entrées analogiques ? quelle gamme de tensions ?
3. trouver la documentation du bloc XADC et identifier les registres pertinents pour la mise en œuvre des convertisseurs
4. proposer un *driver* capable de lire une valeur d'un ADC (quelle est l'adresse de base des registres ?)
5. démontrer un échantillonnage périodique du XADC par un *driver*