Embedded systems 5/7

G. Goavec-Mérou, J.-M Friedt

FEMTO-ST/time & frequency department

jmfriedt@femto-st.fr

slides at `jmfriedt.free.fr`

October 20, 2024

Device

A device only defines the resources needed by a peripheral but will not create the communication entries (taken care of by the *driver*)

```
static struct resource jmf_resources1[] = {
        {.start  = 0x41200000,
         .end    = 0x412000ff,
         .flags  = IORESOURCE_MEM, // could be IORESOURCE_IO: see include/linux/ioport.h
         .name   = "io-led1"       // x86=PortIO ; ARM=MMIO (Memory Mapped IO)
        },
        {.start  = 0x41210000,
         .end    = 0x412100ff,
         .flags  = 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");
```

$\Rightarrow$ otherwise handled by the devicetree

Device

```
[...]
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");
```

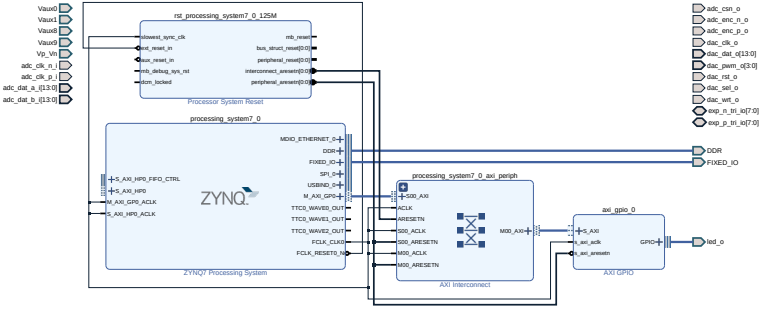
```
redpitaya> insmod dr_comm.ko
redpitaya> insmod pl_resource2.ko
jmf hello 0
jmf hello 1
redpitaya> ls /sys/bus/platform/devices/jmf.*
/sys/bus/platform/devices/jmf.0:
driver           modalias         subsystem        valuer
driver_override  power            uevent           valuew

/sys/bus/platform/devices/jmf.1:
driver           modalias         subsystem        valuer
driver_override  power            uevent           valuew
```

Driver

```
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* my_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));  // end-start information recovery
 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");
```

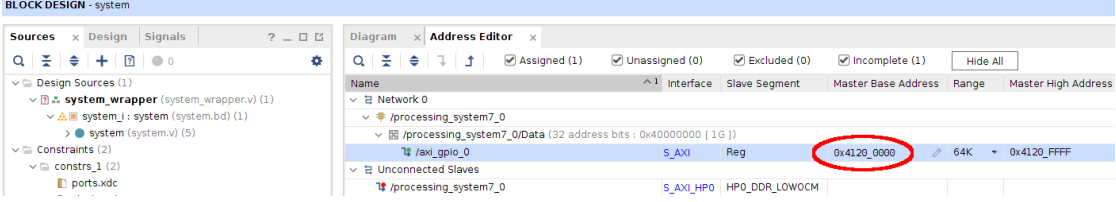
FPGA

Xilinx Vivado to configure the FPGA (WebPack: 37 GB, including SDK)



PS (Zynq xc7z010clg400-1)+bus AXI+GPIO for accessing EMIO (10 min sythesis first time, 5 min afterwards)

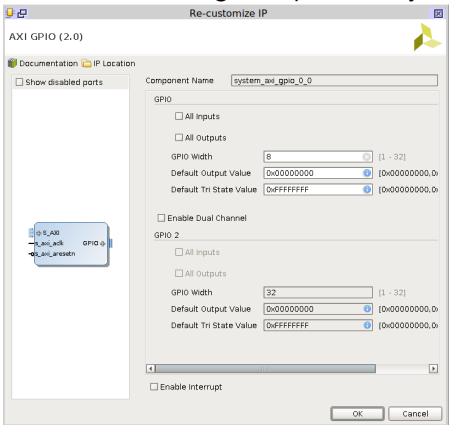
Vivado provides mandatory address ranges for each peripheral on the AXI bus



FPGA: EMIO

EMIO: GPIO connected to the PL ($\neq$ MIO connected to the PS)

Functions of the registers provided by the GPIO block: [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.

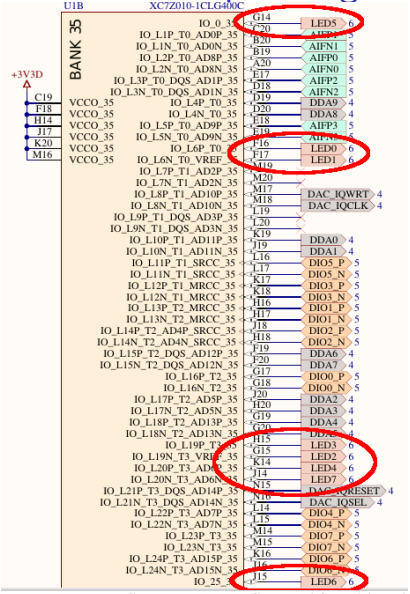
Table 2-4: **Registers**

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.

See https://digilent.com/reference/programmable-logic/guides/vivado-add-gpio

[1] https://docs.amd.com/v/u/en-US/pg144-axi-gpio

FPGA: EMIO pin assignment



File: `ports.xdc`

```
[...]
### LED

set_property IOSTANDARD LVCMOS33 [get_ports {led_o_tri_io[*]}]
set_property SLEW SLOW [get_ports {led_o_tri_io[*]}]
set_property DRIVE 8 [get_ports {led_o_tri_io[*]}]

set_property PACKAGE_PIN F16 [get_ports {led_o_tri_io[0]}]
set_property PACKAGE_PIN F17 [get_ports {led_o_tri_io[1]}]
set_property PACKAGE_PIN G15 [get_ports {led_o_tri_io[2]}]
set_property PACKAGE_PIN H15 [get_ports {led_o_tri_io[3]}]
set_property PACKAGE_PIN K14 [get_ports {led_o_tri_io[4]}]
set_property PACKAGE_PIN G14 [get_ports {led_o_tri_io[5]}]
set_property PACKAGE_PIN J15 [get_ports {led_o_tri_io[6]}]
set_property PACKAGE_PIN J14 [get_ports {led_o_tri_io[7]}]
```

Red_Pitaya_Schematics_STEM_125-10_V1.0.pdf

Configuring the FPGA from GNU/Linux: fpga_manager

Unified, vendor independent access to the PL part of the SoC:

- convert the `.bit` synthesized by Vivado to a `bit.bin`: provide a configuration file `.bif`

```
all: {bitstream_name.bit}
```

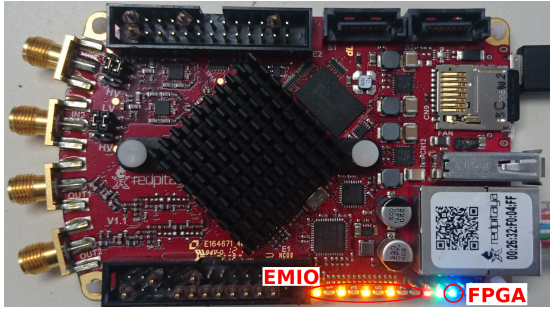
- `bit` → `bit.bin`: `$VIVADO_SDK/bin/bootgen -image file.bif -arch zynq -process_bitstream bin`
- configure the FPGA using `fpga_manager` (`mkdir /lib/firmware` if needed)

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

- Use the functionalities provided by the new FPGA bitstream

```
devmem 0x41200004 8 0x0
devmem 0x41200000 8 0xf
```

- WARNING: accessing the FPGA not configured by a bitstream will yield a crash of Linux (AXI transaction not acknowledged)



GPIO driver

Driver for controlling the 8 LEDs from the probe method (see slide 2 for device definition)

```
#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");
```

Controlling the LEDs from a writable file

Private structure[2] dupplicating the resources used by the driver[3]

```
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* my_struct=dev_get_drvdata(dev);          // LOCAL copy
 long int res;
 if (!kstrtol(buf,10,&res)) writel(res,my_struct->mem_base); // all as output
 printk("%s = %ld",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* my_struct=platform_get_drvdata(pdev);
 kfree(my_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 *my_struct;
 my_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);
 my_struct->mem_base=ioremap(r->start, resource_size(r)); // on recupere end-start
 platform_set_drvdata(pdev,my_struct);
 return 0;
}
```

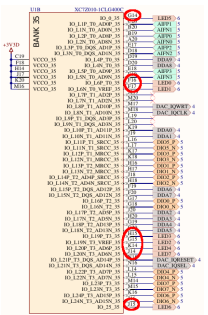
[2] see for example data_dma_direct_core in https://github.com/oscimp/linux_driver/

[3] DO NOT create a global variable which would be overwritten by the latest copy of the driver being loaded.

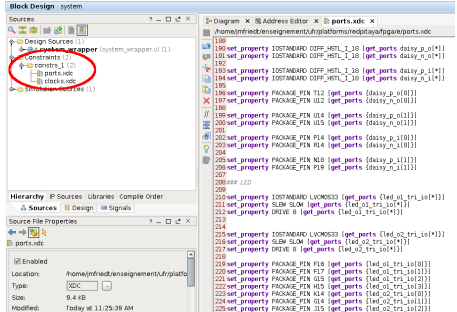
Demonstration using multiple copies of the driver

Application justifying the two resource configurations:

- split the single 8-bit GPIO latch in two 4-bit subsets
- *autoroute*
- constraint file after re-naming `gpio_rtl` to `ledo_2`

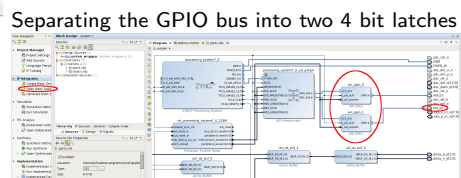


EMIO pins connected to LEDs

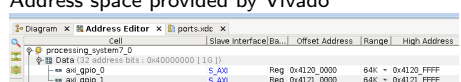


Type and wiring of the pins

Separating the GPIO bus into two 4 bit latches



Address space provided by Vivado



Exercise: update the driver to fetch the two possible configurations, depending whether gpio0 or gpio1 is being controlled

Demonstrate passing parameters (pattern on the LEDs) through `/sys`

Official Xilinx GPIO driver

- Xilinx is providing a GPIO driver with resource definition as devicetree nodes: `$LINUX/Documentation/devicetree/bindings/gpio/gpio-xilinx.txt` (PL) and `$LINUX/Documentation/devicetree/bindings/gpio/gpio-zynq.yaml` (PS)
- source code in `$LINUX/drivers/gpio/gpio-xilinx.c` (PL[4]) and `$LINUX/drivers/gpio/gpio-zynq.c` (PS[5])

Device tree overlay representing the new GPIO:

```
/dts-v1/;  // $BR/dtc -@ -O dtb gpio_overlay.dts -o t.dtbo
/plugin/;
/ {
  fragment@0 {
    target = <&fpga_full>;
    #address-cells = <1>;
    #size-cells = <1>;
    __overlay__ {
      #address-cells = <1>;
      #size-cells = <1>;
      firmware-name = "design_1_wrapper.bit.bin";
      axi_gpio_0: gpio@42100000 {
        #gpio-cells = <2>;
        compatible = "xlnx,xps-gpio-1.00.a";
        gpio-controller ;
        reg = <0x42100000 0x10000>;
        xlnx,gpio-width = <0x8>;
       };
     };
  };
};
```

will create
/sys/class/devices/soc0/fpga-full/42100000.gpio,
/sys/class/firmware/devicetree/base/fpga-full/gpio@42100000,
/sys/bus/platform/devices/42100000.gpio and
/sys/bus/platform/drivers/gpio-xilinx/42100000.gpio

Initially a single GPIO controller (PS):

```
> ls /sys/class/gpio/
export       gpiochip906  unexport
> ls /dev/gpiochip0
/dev/gpiochip0
```

becomes two controllers:

```
> mkdir /sys/kernel/config/device-tree/overlays/myoverlay
> cat t.dtbo  > /sys/kernel/config/device-tree/overlays/myoverlay/dtbo
fpga_manager fpga0: writing design_1_wrapper.bit.bin to Xilinx Zynq →
      ↪FPGA Manager
OF: overlay: WARNING: memory leak will occur if overlay removed...
OF: overlay: WARNING: memory leak will occur if overlay removed...

redpitaya> ls /sys/class/gpio/
export       gpiochip874  gpiochip906  unexport
redpitaya> ls /dev/gpiochip*
/dev/gpiochip0  /dev/gpiochip1

redpitaya> ls /sys/class/gpio/
export       gpiochip874  gpiochip906  unexport
redpitaya> cd /sys/class/gpio/
redpitaya> echo 875 > export
redpitaya> echo "out" > gpio875/direction
redpitaya> echo "1" > gpio875/value
```

[4] xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841645/Solution+Zynq+PL+Programming+With+FPGA+Manager

[5] https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842398/Linux+GPIO+Driver

Official Xilinx GPIO driver

- Xilinx is providing a GPIO driver with resource definition as devicetree nodes: `$LINUX/Documentation/devicetree/bindings/gpio/gpio-xilinx.txt` (PL) and `$LINUX/Documentation/devicetree/bindings/gpio/gpio-zynq.yaml` (PS)
- source code in `$LINUX/drivers/gpio/gpio-xilinx.c` (PL[6]) and `$LINUX/drivers/gpio/gpio-zynq.c` (PS[7])

Device tree overlay representing the new GPIO:

```
/dts-v1/;  // $BR/dtc -@ -O dtb gpio_overlay.dts -o t.dtbo
/plugin/;
/ {
  fragment@0 {
    target = <&fpga_full>;
    #address-cells = <1>;
    #size-cells = <1>;
    __overlay__ {
      #address-cells = <1>;
      #size-cells = <1>;
      firmware-name = "design_1_wrapper.bit.bin";
      axi_gpio_0: gpio@42100000 {
        #gpio-cells = <2>;
        compatible = "xlnx,xps-gpio-1.00.a";
        gpio-controller ;
        reg = <0x42100000 0x10000>;
        xlnx,gpio-width = <0x8>;
       };
     };
  };
};
```

will create
/sys/class/devices/soc0/fpga-full/42100000.gpio,
/sys/class/firmware/devicetree/base/fpga-full/gpio@42100000,
/sys/bus/platform/devices/42100000.gpio and
/sys/bus/platform/drivers/gpio-xilinx/42100000.gpio

Only a single GPIO state can be handled by `/sys/class/gpio` ⇒ see `/dev/gpio*` and use `libgpiod` found in Buidroot at `Hardware handling` → `libgpiod and tools`

```
> gpiodetect
gpiochip0 [zynq_gpio] (118 lines)
gpiochip1 [42100000.gpio] (32 lines)
> gpioset  gpiochip1 1=1 2=1 3=0 4=1
```

if `xlnx,gpio-width = <0x8>;` was omitted in the .dts (default 32 lines) or when adding this entry:

```
> gpiodetect
gpiochip0 [zynq_gpio] (118 lines)
gpiochip1 [42100000.gpio] (8 lines)
```

[6] `xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841645/Solution+Zynq+PL+Programming+With+FPGA+Manager`
[7] `https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842398/Linux+GPIO+Driver`

Exercise

1. add the `XADC` block to the Vivado design already including the GPIO
2. synthesize and configure the FPGA using this bitstream: what pins are associated with the analog inputs? what is the voltage range?
3. find the documentation of the XADC block and identify the registers relevant for using the analog to digital converters
4. write a driver able to read an ADC value (what is the base-address of the registers?)
5. demontrate periodic acquisition of the XADC samples by a driver