Embedded systems 7/7

J.-M Friedt

FEMTO-ST/time & frequency department

jmfriedt@femto-st.fr

slides at `jmfriedt.free.fr`

October 3, 2020

IIO

IIO : Industrual Input/Outputs. (2009)

Why yet another communication framework ?

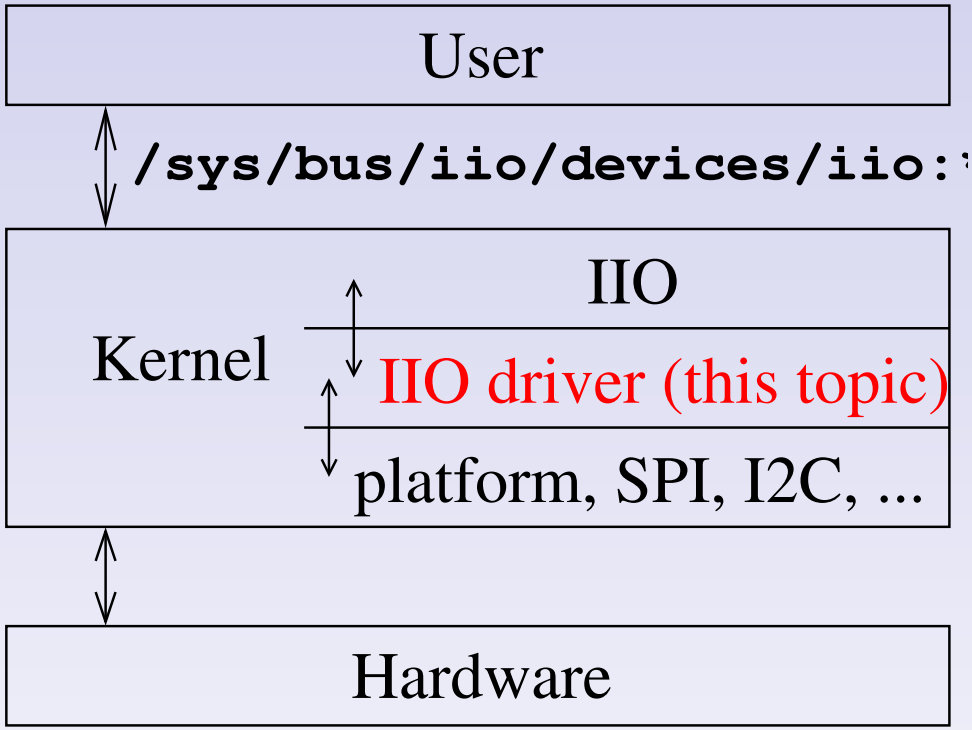
- IIO provides the interfaces that all sensor drivers accessed through the Linux kernel must expose (ADC, DAC, accelerometer, magnetometer ...),
- share information as ASCII sentences rather than binary, easier to use from the shell or interpreted scripting languages (e.g. Python)
- API between the user interface and the drivers accessing the hardware

```
static int ad5624r_spi_write(struct spi_device *spi,
                             u8 cmd, u8 addr, u16 val, u8 len)
{
     u32 data;
     u8 msg[3];

     data = (0 << 22) | (cmd << 19) | (addr << 16) | (val << (16 - len));
     msg[0] = data >> 16;
     msg[1] = data >> 8;
     msg[2] = data;

     return spi_write(spi, msg, 3);
}
```

Interface definition



`modprobe industrialio`
to avoid

```
[ 81.594608] iio_simple_dummy: Unknown symbol iio_device_alloc (err 0)
[ 81.594621] iio_simple_dummy: Unknown symbol iio_device_free (err 0)
[ 81.594634] iio_simple_dummy: Unknown symbol iio_device_unregister (err 0)
[ 81.594644] iio_simple_dummy: Unknown symbol iio_device_register (err 0)
```

Loading the driver ...

- In case auto-detection is not taken care of, description of the hardware architecture through the devicetree
- definition of a driver (which is *not* automatically probed)

```
static const struct platform_device_id ad5624r_id[] = {
        {"ad5624r3", ID_AD5624R3},
        {"ad5644r3", ID_AD5644R3},
        {"ad5664r3", ID_AD5664R3}, // peripheral identifier
...

static struct spi_driver ad5624r_driver = {
        .driver = {
                    .name = "ad5624r",
                    .owner = THIS_MODULE,
                    },
        .probe = ad5624r_probe,
        .remove = ad5624r_remove,
        .id_table = ad5624r_id,  // which components are controlled by
};                               //  ... by this driver
```

... through a device definition

- a platform device requests the driver services upon initialization

```
static struct platform_device plat_gr_device = {
   .name = "ad5624r3",     // device requiring the driver
// SAME NAME than in static struct platform_driver ad5624r_driver = {
```

- the probe method of the driver is called as many times as the peripheral is detected
- the probe method accepts arguments that init could not have received.

```
insmod composant.ko
insmod dummy_platform.ko
ls -l /sys/bus/iio/devices/iio\:device0/
cat /sys/bus/iio/devices/iio\:device0/name
```

- this time

```
ret = iio_device_register(indio_dev);
```

$\Rightarrow$ creates an IIO interface ($\neq$ platform_device_register)

Driver initialization

```
static int composant_probe(struct platform_device *pdev)
{struct composant_state *st;
 struct iio_dev *indio_dev;
 static const char *iio_dummy_part_number = "iio_dummy_part_no";
 indio_dev = devm_iio_device_alloc(&pdev->dev, sizeof(*st));
 st = iio_priv(indio_dev);
 platform_set_drvdata(pdev, indio_dev);
 indio_dev->dev.parent = &pdev->dev;
 indio_dev->name=iio_dummy_part_number;

 indio_dev->info = &composant_info;
 indio_dev->modes = INDIO_DIRECT_MODE;
 indio_dev->channels = st->chip_info->channels;
 indio_dev->num_channels = COMPOSANT_CHANNELS;

 ret = iio_device_register(indio_dev);
...
}
```

and loading the driver yields

```
# cat /sys/bus/iio/devices/iio\:device0/name
iio_dummy_part_no
```

Files describing each interface

```
#define AD5624R_CHANNEL(_chan, _bits) { \
        .type = IIO_VOLTAGE, \
        .indexed = 1, \
        .output = 1, \
        .channel = (_chan), \
        .info_mask_separate = BIT(IIO_CHAN_INFO_RAW), \
        .info_mask_shared_by_type = BIT(IIO_CHAN_INFO_SCALE), \
...
```

The driver describes [1] the hardware functionalities (input/output, channels, calibration coefficients ...)

```
#define DECLARE_AD5624R_CHANNELS(_name, _bits) \
        const struct iio_chan_spec _name##_channels[] = { \
                AD5624R_CHANNEL(0, _bits), \
                AD5624R_CHANNEL(1, _bits), \
                AD5624R_CHANNEL(2, _bits), \
                AD5624R_CHANNEL(3, _bits), \
}

static DECLARE_AD5624R_CHANNELS(ad5624r, 12);
static DECLARE_AD5624R_CHANNELS(ad5644r, 14);
static DECLARE_AD5624R_CHANNELS(ad5664r, 16);
```

[1] include/uapi/linux/iio/types.h in the linux kernel source for existing types

Files describing each interface

```
#define AD5624R_CHANNEL(_chan, _bits) { \
        .type = IIO_VOLTAGE, \
        .indexed = 1, \
        .output = 1, \
        .channel = (_chan), \
        .info_mask_separate = BIT(IIO_CHAN_INFO_RAW), \
        .info_mask_shared_by_type = BIT(IIO_CHAN_INFO_SCALE), \
...
```

The driver describes the hardware functionalities (input/output, channels, calibration coefficients ...)

```
#define DECLARE_AD5624R_CHANNELS(_name, _bits) \
        const struct iio_chan_spec _name##_channels[] = { \
                AD5624R_CHANNEL(0, _bits), \
                AD5624R_CHANNEL(1, _bits), \
                AD5624R_CHANNEL(2, _bits), \
                AD5624R_CHANNEL(3, _bits), \
}
```

and the associated access points will be created:

```
static const struct ad5624r_chip_info ad5624r_chip_info_tbl[] = {
 [ID_AD5624R3] = {.channels = ad5624r_channels,
                  .int_vref_mv = 1250, },
```

Interface definitions

Creation of /sys/bus/iio/devices/iio\:device0 which includes

```
dev                          in_accel_x_raw         in_voltage0_raw
in_voltage3-voltage4_raw     out_voltage0_raw       uevent
in_accel_x_calibbias         in_sampling_frequency  in_voltage0_scale
in_voltage-voltage_scale     power
in_accel_x_calibscale        in_voltage0_offset     in_voltage1-voltage2_raw
name                         subsystem
```

Interfaces have been defined by the component description:

iio_chan_spec [2]

```
static const struct iio_chan_spec iio_dummy_channels[] = {
{ .type = IIO_VOLTAGE,
  .indexed = 1,
  .channel = 0, // Channel has a numeric index of 0
  .info_mask_separate = // liste des infos fournies par le canal
  BIT(IIO_CHAN_INFO_RAW) | BIT(IIO_CHAN_INFO_OFFSET) | BIT(IIO_CHAN_INFO_SC
  .info_mask_shared_by_dir = BIT(IIO_CHAN_INFO_SAMP_FREQ), // sampling_freq
  .scan_index = voltage0,
  .scan_type = { // Description of storage in buffer
               .sign = 'u', /* unsigned */
               .realbits = 13, /* 13 bits */
               .storagebits = 16, /* 16 bits used for storage */
               .shift = 0, /* zero shift */
```

List of information (RAW, OFFSET...) can be found in write

Writing

Functions called when reading or writing:

```
static const struct iio_info composant_info = {
        .write_raw = composant_write_raw,
        .read_raw = composant_read_raw,
        .driver_module = THIS_MODULE,
};
```

requiring the low-level functions:

```
static int composant_write_raw(struct iio_dev *indio_dev,
   struct iio_chan_spec const *chan, int val, int val2, long mask)
{struct composant_state *state = iio_priv(indio_dev);
 int ret;
 switch (mask) {              // kind of transaction
    case IIO_CHAN_INFO_RAW:  // vvv fetches the number of bits in chan
      if (val >= (1 << chan->scan_type.realbits) || val<0) return -EINVAL;
      ret = ad5624_spi_write(state, chan->channel, val);
    default: ret = -EINVAL;
   }
  return ret;
}
```

A unique function called upon writing + kind of transaction selected by info [3] exploiting the function described initially (`ad5624r_spi_write()`)

[3] https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/iio/dac/ad8801.c?id=refs/tags/v4.9-rc8

Reading

Fills *val, *val2 and returns the kind of transaction completed

```
static int composant_read_raw(struct iio_dev *indio_dev,
  struct iio_chan_spec const *chan, int *val, int *val2, long info)
{struct composant_state *state = iio_priv(indio_dev);
 switch (info) {
  case IIO_CHAN_INFO_RAW:
    *val = state->dac_cache[chan->channel];
    return IIO_VAL_INT;              // integer value returned
  case IIO_CHAN_INFO_SCALE:
    *val = state->vrefh_mv - state->vrefl_mv;
    *val2 = 8;
    return IIO_VAL_FRACTIONAL_LOG2; // mantissa and exponent (float)
  case IIO_CHAN_INFO_OFFSET:
    *val = state->vrefl_mv;
    return IIO_VAL_INT;
  default:
    return -EINVAL;
 }
 return -EINVAL;
}
```

FPGA peripheral: XADC

- XADC [4]: 1 MS/s, 12 bit analog to digital converter ...
- ... connected to PS and PL
- Called as a peripheral from Vivado (XADC Wizard)
- PL: data source towards the PS or other processing blocks in the FPGA for real time processing

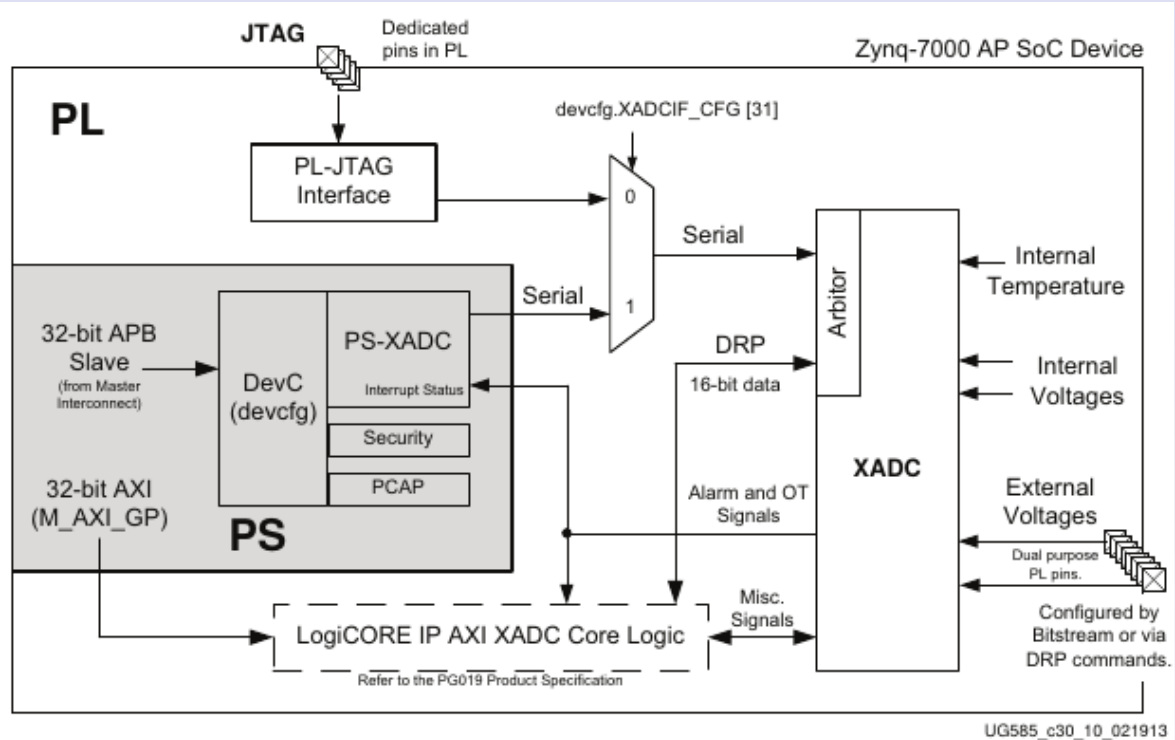
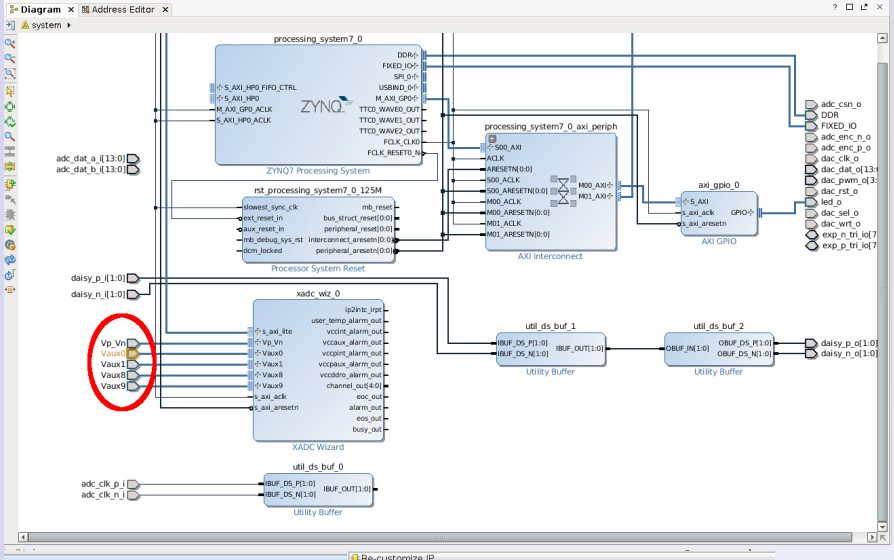


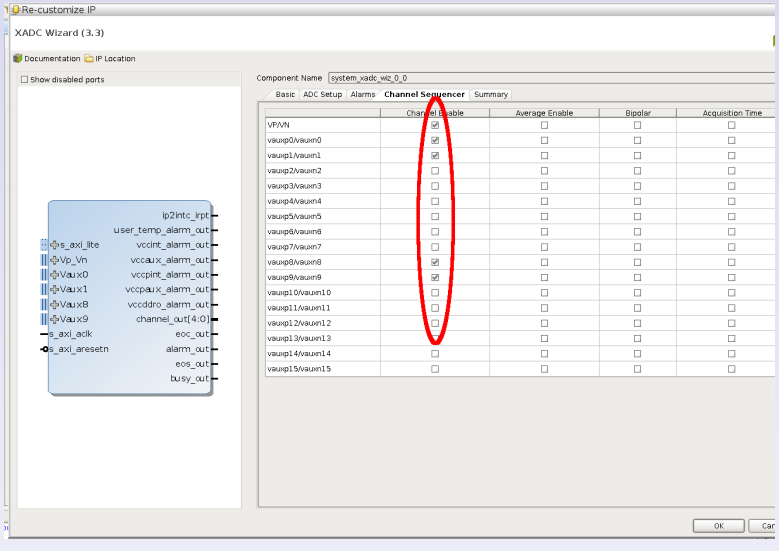
Figure 30-1: **XADC Module System Viewpoint**

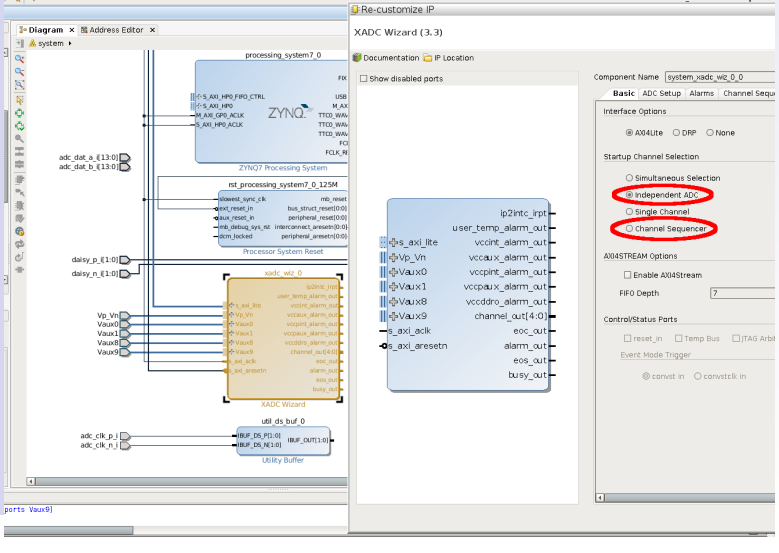
[4] https://www.xilinx.com/support/documentation/ip_documentation/xadc_wiz/v3_0/pg091-xadc-wiz.pdf

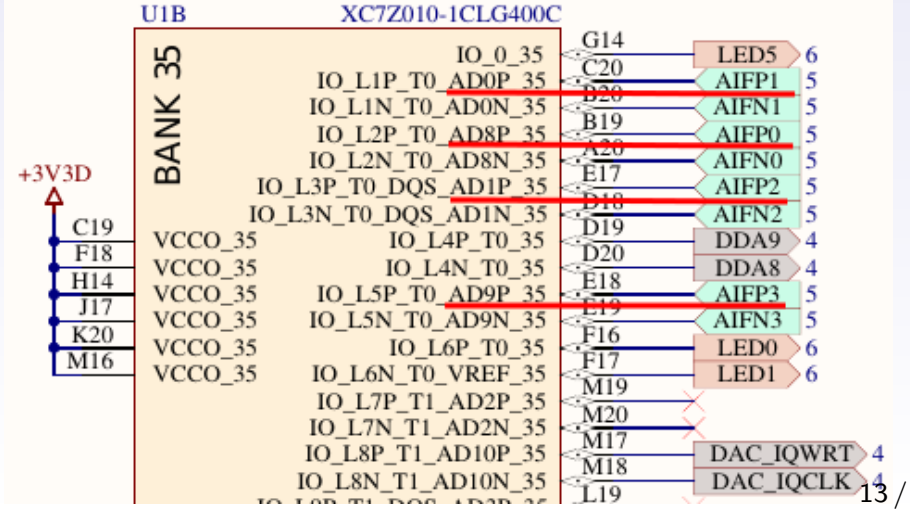
Vivado configuration

- Continuous measurement (no trigger)
- Activate channels 8 and 9
- Autoroute clocks and AXI bus signals









Accessing XADC from the PS

- Xilinx provides an IIO XADC diver: `xilinx_xadc.ko`
- this driver is configured from the *devicetree*
- Reconfiguring the driver requires defining it as a module
- Avoid rebooting upon loading XADC in the devicetree: **overlay**

```
/dts-v1/;
/plugin/;
/ {compatible = "xlnx,zynq-7000";

   fragment@0 {
        target = <&fpga_full>;
        #address-cells = <1>;
        #size-cells = <1>;
        __overlay__ {
                #address-cells = <1>;
                #size-cells = <1>;
                firmware-name = "system_wrapper.bit.bin";
                };
        };
   fragment@1 {
        target = <&adc>;
        __overlay__ {
                xlnx,channels {
                     #address-cells = <1>;
                     #size-cells = <0>;
                     channel@0 {reg = <0>;};
                     channel@1 {reg = <1>;};
                     channel@2 {reg = <2>;};
                     channel@9 {reg = <9>;};
                     channel@10 {reg = <10>;};
                    };
                };
        };
};
```

The `fragment@1` node overloads the parameters of the adc node of the original devicetree

```
adc: adc@f8007100 {
   compatible = "xlnx,zynq-xadc-1.00.a";
   reg = <0xf8007100 0x20>;
   interrupts = <0 7 4>;
   interrupt-parent = <&intc>;
   clocks = <&clkc 12>;
};
```

with `compatible` argument identical to the one found in the driver `drivers/iio/adc/xilinx-xadc-core.c`

```
static const struct of_device_id xadc_of_match_table[] = {
  {.compatible = "xlnx,zynq-xadc-1.00.a", (void *)&xadc_zynq_op
  {.compatible = "xlnx,axi-xadc-1.00.a", (void *)&xadc_axi_ops
  { },
};
```

The `fragment@0` node refers to the `fpga_manager` for overloading the bitstream (located in `/lib/firmware`)

Loading the bitstream and the IIO driver

```
$ mkdir /sys/kernel/config/device-tree/overlays/toto
$ rmmod xilinx_xadc
$ cp system_wrapper.bit.bin /lib/firmware
$ cat xadc.dtbo > /sys/kernel/config/device-tree/overlays/toto/dtbo
$ dmesg | tail -1
fpga_manager fpga0: writing system_wrapper.bit.bin to Xilinx Zynq FPGA Mana
$ modprobe xilinx_xadc
$ pwd
/sys/bus/iio/devices/iio:device0
$ ls
...
in_voltage10_vaux1_scale   in_voltage8_vaux9_raw
in_voltage11_vaux0_raw     in_voltage8_vaux9_scale
in_voltage11_vaux0_scale   in_voltage9_vaux8_raw
in_voltage12_vpvn_raw      in_voltage9_vaux8_scale
```

- Standard interfaces for a data acquisition device (`raw`, `scale`, `offset` ...)

Raw access from the PS

- Vivado has provided a base address for the peripheral
- The XADC IP documentation explains the meaning of the registers at relative addresses

Table 2-4: **IP Core Registers *(Cont'd)***

Base Address + Offset (hex)	Register Name	Reset Value (hex)	Access Type	Description
C_BASEADDR + 0x260	$V_{AUXP}[8]$/ $V_{AUXN}[8]$	0x0	R(6)	The 12-bit MSB justified result of A/D conversion on the auxiliary analog input 8 is stored in this register.
C_BASEADDR + 0x264	$V_{AUXP}[9]$/ $V_{AUXN}[9]$	0x0	R(6)	The 12-bit MSB justified result of A/D conversion on the auxiliary analog input 9 is stored in this register.
C_BASEADDR + 0x268	$V_{AUXP}[10]$/ $V_{AUXN}[10]$	0x0	R(6)	The 12-bit MSB justified result of A/D conversion on the auxiliary analog input 10 is stored in this register.

Conclusion

- Loss of the general "everything is a file" system call philosophy of `/dev` ...
- ... with the gain of a uniform filesystem representing each peripheral/bus type [5].
- Interfaces separating data and configurations.
- New abstraction layer with a description separating device and drivers.
- Data exchange as ASCII sentences.

Resources:

- IIO documentation in the kernel sources [6]
- M. Ripard, *IIO, a new kernel subsystem*, FOSDEM 2012 [7]
- L.-P. Clausen, *Using the Linux IIO framework for SDR – A hardware abstraction layer*, FOSDEM 2015 [8]
- J. Madieu, *Linux Device Drivers Development*, Packt (2017), chapter 10: "IIO Framework"

[5] `https://archive.fosdem.org/2018/schedule/event/plutosdr/`

[6] `lxr.free-electrons.com/source/drivers/staging/iio/Documentation/`

[7] `https://archive.fosdem.org/2012/schedule/event/693/127_iio-a-new-subsystem.pdf` and `free-electrons.com/blog/fosdem2012-videos/` for the video of the talk

[8] `https://archive.fosdem.org/2015/schedule/event/iiosdr/`

Demonstration on STM32MP157

Accessing the STM32MP157 SOC ADC defined in the devicetree [9] through IIO driver[10] /dev entry following a configuration of the clock source [11]:

```
echo 1 > /sys/bus/iio/devices/iio\:device0/scan_elements/in_voltage0_en
echo 1 > /sys/bus/iio/devices/iio\:device0/scan_elements/in_voltage1_en
cat /sys/bus/iio/devices/iio\:device0/trigger/current_trigger
cat /sys/bus/iio/devices/trigger0/name
echo tim6_trgo > /sys/bus/iio/devices/iio\:device0/trigger/current_trigger
cat /sys/bus/iio/devices/iio\:device0/trigger/current_trigger
echo 100 > /sys/bus/iio/devices/iio\:device0/buffer/length
cat /sys/bus/iio/devices/iio\:device0/buffer/length
echo 1 > /sys/bus/iio/devices/iio\:device0/buffer/enable
echo 100000 > /sys/bus/iio/devices/trigger0/sampling_frequency
cat < /dev/iio\:device0 | xxd
```

⇒ `gr-iio` to use IIO peripheral as GNU Radio source

[9] https://wiki.st.com/stm32mpu/wiki/ADC_device_tree_configuration
[10] https://wiki.st.com/stm32mpu/wiki/ADC_Linux_driver
[11] https://www.linux4sam.org/bin/view/Linux4SAM/IioAdcDriver