

Embedded systems 8/9

CPU/FPGA co-design: OscimpDigital & al.

J.-M Friedt, G. Goavec-Merou²

FEMTO-ST/time & frequency department
² Enjoy Digital

`jmfriedt@femto-st.fr`

slides at `jmfriedt.free.fr`

December 2, 2025

CPU-FPGA context

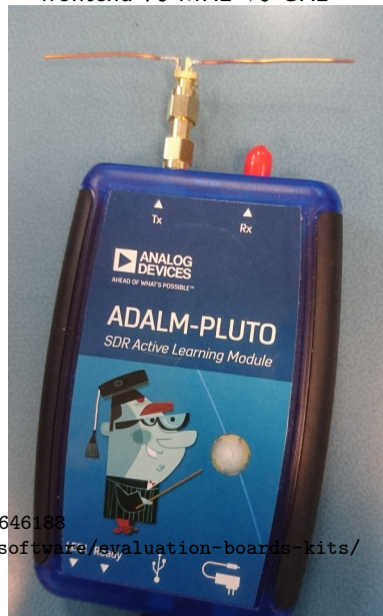
Red Pitaya¹: **baseband** dual ADC & DAC

14/16 bits @ 125 MHz

⇒ Well suited for acquisition and processing radiofrequency signals with co-design CPU/FPGA:

- ▶ FPGA (real-time, pipelined) for fast task
 - ▶ data acquisition;
 - ▶ frequency transposition;
 - ▶ filtering;
 - ▶ decimation.
- ▶ CPU (general purpose OS) for slow task after decimation
 - ▶ post-processing;
 - ▶ display;
 - ▶ transmission;
 - ▶ configuration

PlutoSDR²³: **AD9363 RF**
frontend 70 MHz→6 GHz



⁵<https://fr.rs-online.com/web/p/kit-d-inventions-educatives/2646188>

⁶<https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html>

⁷<https://wiki.analog.com/university/tools/pluto/hackers>

Example: ADC to DAC and to PS, FPGA pre-processing ⁸

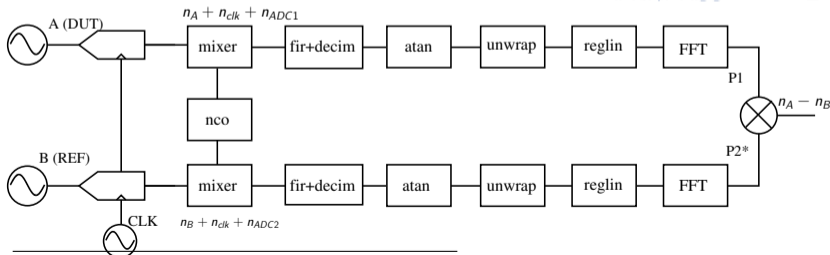
ADC



PL->PS



DAC

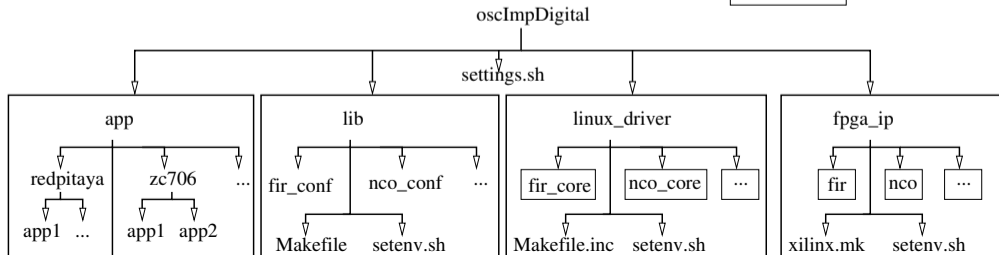
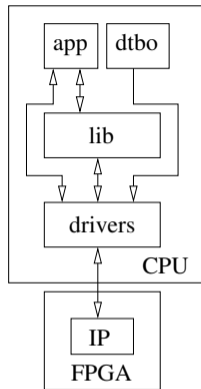


⁸https:

OscIMP Digital ecosystem

Purpose: provide a coherent environment to create design (FPGA), and application:

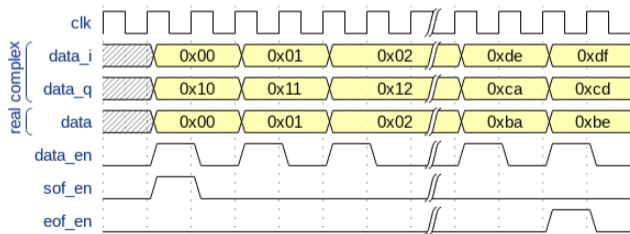
- ▶ blocks (IP) with algorithm level of implementation (FPGA);
- ▶ GNU/Linux hierarchy compliance (driver/library/application);
- ▶ tools to generate some files and scripts/Makefile to factorize most common part;
- ▶ consistent addressing & bitstream loading under supervision of Device Tree Binary Overlay (dtbo).



FPGA

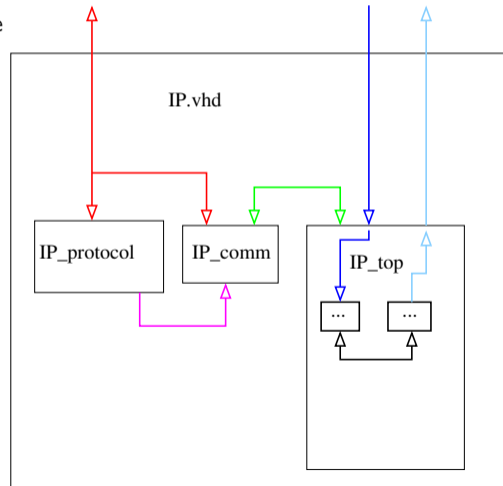
Developer aspect:

- ▶ normalize interfaces between blocks
- ▶ isolation between implementation and communication:
 - ▶ IP_protocol: “complex” AXI bus management to provide a “simple” interface for register management
 - ▶ IP_comm: register management
 - ▶ IP_top: algorithm implementation independent of the configuration communication issues
 - ▶ Data streams between IPs ↓



End user aspect:

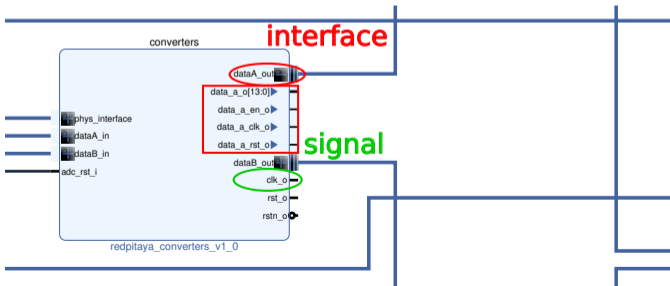
- ▶ 0, 1 or more interface(s) to connect;
- ▶ AXI interface automatically connected.



FPGA

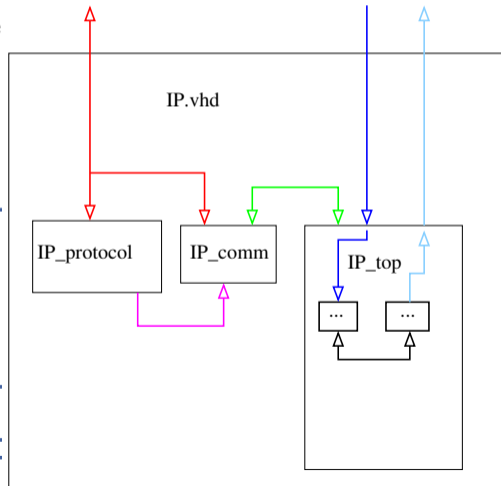
Developer aspect:

- ▶ normalize interfaces between blocks
- ▶ isolation between implementation and communication:
 - ▶ IP_protocol: “complex” AXI bus management to provide a “simple” interface for register management
 - ▶ IP_comm: register management
 - ▶ IP_top: algorithm implementation independent of the configuration communication issues
 - ▶ Data streams between IPs ↓



End user aspect:

- ▶ 0, 1 or more interface(s) to connect;
- ▶ AXI interface automatically connected.



CPU: environment

Char device drivers to add abstraction, GNU/Linux hierarchy compliance and communication improvement:

- ▶ 1 IP with communication \Rightarrow 1 (or more) driver(s);
- ▶ a core driver knows how to communicate with an IP but not where;
- ▶ device tree overlay used to provide which drivers must be probed and base address for each of them;

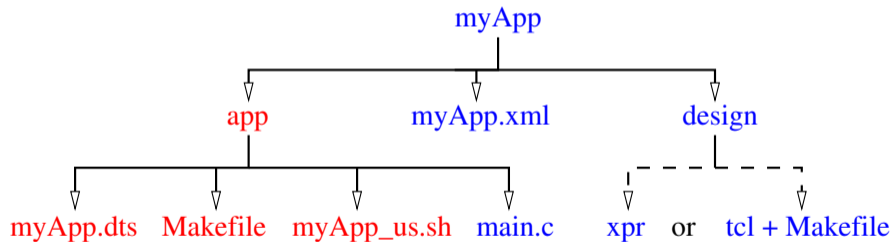
libraries to simplify some common and long (number of line) tasks.

TODO \Rightarrow challenging IIO integration for each individual IP (as opposed to one single integrated component representing all aspects of the AD936x)

CPU: application

Application structure:

- ▶ dts to provides which driver must be used and base address;
- ▶ Makefile to cross-compile application and generate the dtbo from dts
- ▶ applicationName_us.sh: a shell script used to flash FPGA, load devicetree and drivers;
- ▶ main.c: user application



user defined

automatically created by module_generator (based on XML file)

CPU: module_generator

- ▶ Used to generate some files in app directory.
- ▶ use an XML file for describing design information.

```
module_generator -dts myApp.xml
```

```
<?xml version="1.0" encoding="utf-8"?>  
<project name="tutorial5" version="1.0">  
  <options>  
    <option target="makefile" name="USE_STATIC_LIB">1</option>  
    <option target="makefile" name="LDFLAGS">-liio</option>  
  </options>  
  <ips>  
    <ip name ="dataComplex_to_ram" >  
      <instance name="data1600" id = "0"  
        base_addr="0x43c00000" addr_size="0xffff" />  
    </ip>  
    <ip name ="nco_counter">  
      <instance name="nco" id = "0"  
        base_addr="0x43c10000" addr_size="0xffff" />  
    </ip>  
  </ips>  
</project>
```

XML is generated from synthesized project with `make xml` from the design directory

CPU: module_generator

- ▶ Used to generate some files in app directory.
- ▶ use an XML file for design's informations.

```
module_generator -dts myApp.xml
```

myApp.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<project name="tutorial5" version="1.0">  
  <ips>  
    <ip name="data_to_ram" >  
      <instance name="data1600" id="0"  
        base_addr="0x43c00000" addr_size="0xffff" />  
    </ip>  
    <ip name="nco_counter">  
      <instance name="datanco0" id="0"  
        base_addr="0x43c10000" addr_size="0xffff" />  
    </ip>  
  </ips>  
</project>
```

tutorial5_us.sh

```
cp ../bitstreams/tutorial5_wrapper.bit.bin /lib/firmware  
rmdir /sys/kernel/config/device-tree/overlays/fpga  
mkdir /sys/kernel/config/device-tree/overlays/fpga  
cat tutorial5.dtbo > $DTB_DIR/dtbo  
insmod ../../modules/data_to_ram_core.ko  
insmod ../../modules/nco_counter_core.ko
```

```
/dts-v1/;  
/plugin/;  
/ {  
    compatible = "xlnx,zynq-7000";  
    fragment0 {  
        target = <&fpga_full>;  
        #address-cells = <1>;  
        #size-cells = <1>;  
        __overlay__ {  
            #address-cells = <1>;  
            #size-cells = <1>;  
            firmware-name = "tutorial5_wrapper.bit.bin";  
            data1600: data1600@43c00000 {  
                compatible = "ggm,dataToRam";  
                reg = <0x43c00000 0xffff>;  
            };  
            datanco0: datanco0@43c10000 {  
                compatible = "ggm,nco_counter";  
                reg = <0x43c10000 0xffff>;  
            };  
        };  
    };  
};
```

tutorial5.dts

Play with repositories⁹

1. Assumption: functional Vivado and buildroot frameworks. In rooms 2{1,3}5B
/home/jmfriedt/buildroot-2025.05.1_redpit/
2. Generate a function OscimpDigital framework: example in room 235B
/home/jmfriedt/oscimpDigital/
 - 2.1 Clone repository and submodules:

```
git clone --recursive https://github.com/oscimp/oscimpDigital.git
```
 - 2.2 Configure: fill variables in settings.sh and source configuration file

```
export BOARD_NAME=redpitaya  
export BR_DIR=/home/jmfriedt/buildroot-2025.05.1_redpit/  
...
```
 - 2.3 make in linux_driver to build kernel modules
 - 2.4 make in lib to compile userspace library (copy to Red Pitaya)
3. Discover:
In oscimpDigital/doc/tutorials/redpitaya

See first oscimpDigital/README.md to configure your shell environment.

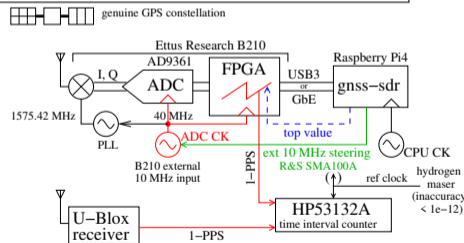
Cheat-Sheet for Redpitaya available at jmfriedt.free.fr/redpitaya_cheat-sheet.pdf

⁹https://github.com/sdenis6/Base_Designs_tuto

Alternatives to OscimpDigital and HDL programming

- ▶ Python description of hardware signals: (n)Migen¹⁰, Amaranth^{11 12}, LiteX^{13 14}
- ▶ Python testbench/verification framework: cocotb at <https://www.cocotb.org/>
- ▶ High Level-language Synthesis (HLS): C to VHDL converter, difficult to extract best performance of FPGA parallelisation, see #pragma directives¹⁵ to steer the compiler

Notice that only ADC timestamps N matter¹⁶, all subsequent buffering/streaming without sample loss allows implicit time computation (N/f_s)



¹⁰<https://github.com/m-labs/migen>

¹¹<https://github.com/amaranth-lang/amaranth>

¹²https://github.com/oscimp/amaranth_twstft/

¹³<https://github.com/enjoy-digital/litex>

¹⁴https://github.com/enjoy-digital/litex_wr_nic

¹⁵<https://docs.amd.com/r/en-US/ug1399-vitis-hls/pragma-HLS-aggregate>

¹⁶D. Rabus, G. Goavec-Merou, G. Cabodevila, F. Meyer, J.-M Friedt, *Generating a timing information (1-PPS) from a software defined radio decoding of GPS signals*, Proc. Joint EFTF/IFCS (2021)

Outline of the lab session

1. ADC→DAC (PL standalone application)
2. ADC→DAC & PS (DataToRAM): add kernel driver for reading
3. ADC→DAC & PS after processing (FIR filter): add kernel driver for configuring
4. ADC→DAC & PS after processing (frequency transposition by NCO)
5. TCL programming instead of graphical user interface

Summary:

```
cd design
make clean && make xpr && make xml && make && make install
cd ..
module_generator my.xml
webserver_generator my.xml
cd app
make clean && make
```