Platform independent CPU/FPGA co-design: the OscimpDigital framework

G. Goavec-Merou, J.-M. Friedt, , P.-Y. Bourgeois
FEMTO-ST Time & Frequency department, Besançon, France
Contact: {gwenhael.goavec,jmfriedt,pyb2}@femto-st.fr

Slides at
https://github.com/oscimp/oscimpDigital/tree/master/doc/conferences/fosdem2020

Feb. 02, 2020

Goal

A **coherent ecosystem for co-design** CPU (with Linux) and FPGA, to assemble and generate Digital Signal Processing chains in FPGA controlled from CPU.

- fully pipelined chains (no FIFO): direct sample comsumption;
- comply with GNU/Linux Operating System hierarchy (userspace application, libraries, drivers, IP connected to the CPU);
- vendor independent: able to handle Xilinx Vivado and Intel Quartus (may be extended to others vendor tool)

Validated with / support:

- Red Pitaya (14&16bits): Zynq 7010 & 7020;
- de0nanoSoc: CycloneV Soc
- plutoSDR: Zynq 7010
- ADRV9361: Zynq 7035
- USRP E310: Zynq 7020

Existing ecosystems

Ettus RFNoC:

Pro:

- coherent/transparent for user (UHD abstraction);
- enable/disable IPs at runtime (heterogeneous processing chain)

Cons:

- IP not available in firmware ⇒ new bitstream to be generated;
- limited number of blocks at the same time;
- USRP dependent (motherboard/daughterbord/I2C EEPROM);
- latencies introduced by crossbar

Pavel Demin red-pitaya-notes:

Pro:

- provides plug and play projects;
- documentation about projects;
- direct compatibility with GNU Radio (osmosdr)

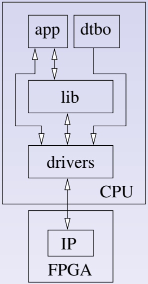
Cons

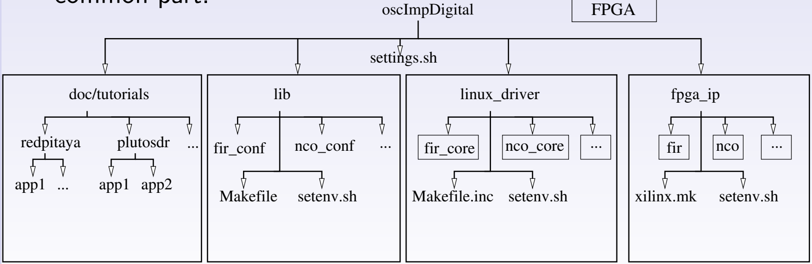
- dedicated to Red Pitaya platform;
- more or less limited to provided project.

OscimpDigital [1] ecosystem

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

- blocks (IP) with algorithm level of implementation (FPGA);
- GNU/Linux hierarchy compliance (driver/library/application);
- tools to generate some files and scripts/Makefile to factor most common part.





[1] created thanks to the Oscillator Instability Measurement Platform (OscIMP), http://oscillator-imp.com

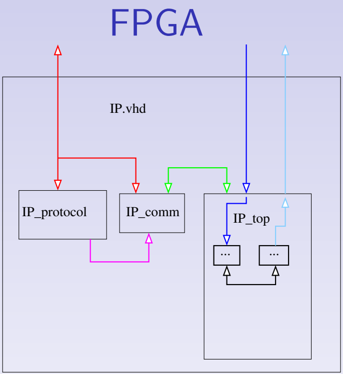
Algorithms or utility functions.

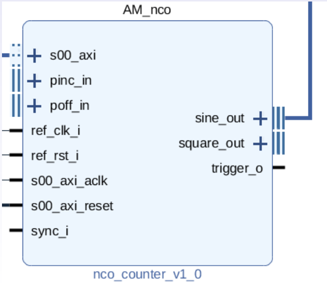
Developer aspect:

- normalize interfaces between blocks
- isolation between implementation and communication

End user aspect:

- 0, 1 or more interface to connect;
- AXI interface automatically connected.





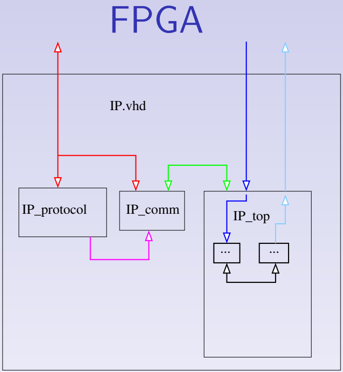
G. Goavec-Merou & al.

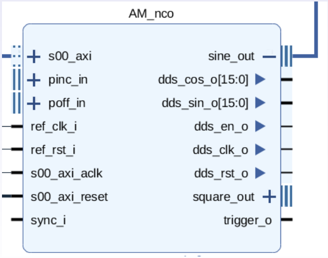
Algorithms or utility functions.
Developer aspect:

- normalize interfaces between blocks
- isolation between implementation and communication

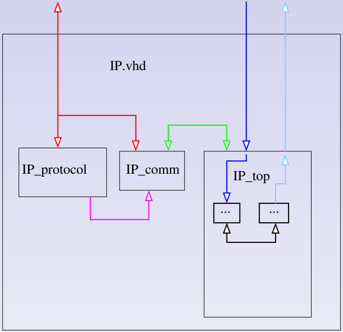
End user aspect:

- 0, 1 or more interface to connect;
- AXI interface automatically connected.





FPGA

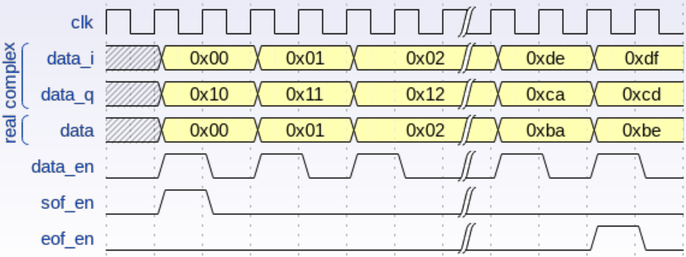


Algorithms or utility functions.
Developer aspect:

- normalize interfaces between blocks
- isolation between implementation and communication

End user aspect:

- 0, 1 or more interface to connect;
- AXI interface automatically connected.



Independant TCL set

All vendor tools have a TCL mode, but:

- each vendor provides custom functions
- way to build project/block design are differents

⇒ need to provide an set of functions and Makefile to have add an abstraction.

- same script may be used with different boards for different manufacturers;
- only need to reimplement a few procedures for supporting a new tool

Vivado:

```
create_bd_cell -type ip -vlnv ggm:cogen:myIP myip]
set_property -dict [ list CONFIG.PARAM1 14 \
    CONFIG.PARAM2 true ] myIP
```

Quartus:

```
add_instance myip myIP 1.0
set_instance_parameter_value myip PARAM1 14
set_instance_parameter_value myip PARAM2 true
```

OscimpDigital:

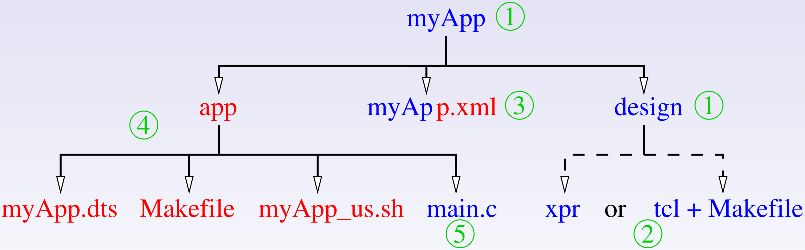
```
add_ip_and_conf myIP myip {
    PARAM1 14 PARAM2 true}
```

Makefile:

```
PRJ=myGateware
CONSTR_redpitaya=leds.xdc   # use for Red Pitaya board
CONSTR_de0nanoSoc=leds.tcl # use for de0nanoSoc board
TCL_LIST=myScript.tcl
-include $(OSCIMP_DIGITAL_IP)/fpga_ip.mk
```

Project structure

- TCL script or GUI generated FPGA design
- devicetree (`.dts`) provides list of drivers and base addresses;
- `Makefile` to cross-compile application & 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)

development flow

CPU: module_generator

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

```
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
```

tutorial5.dts

```
/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>;
            };
        };
    };
};
```

Makefile

```
BASE_NAME=tutorial5
CORE_MODULES_LIST=$(OSCIMP_DIGITAL_DRIVER)/nco_core/*.ko \
        $(OSCIMP_DIGITAL_DRIVER)/data_to_ram_core/*.ko
include $(OSCIMP_DIGITAL_APP)/Makefile.inc
```

Available processing blocks

Radiofrequency signal handling

- nco_counter — local oscillator
- mixerComplex_sin, mixer_sin — frequency transposition
- redpitaya_converters — Red Pitaya platform hardware interfaces (in/out)
- axi_deltaSigma — low frequency output (audio output)
- gen_radar_prog, syncTrigStream — pulsed RADAR, synchronization

Radiofrequency signal processing

- cacode — GPS Gold code generator
- firReal — CPU configurable Finite Impulse Response filter
- prn, prn20b, xcorr_prn_slow_complex — pseudo-random sequence generator, correlator

Two types of interfaces: real values and complex values (*: valid for both types):

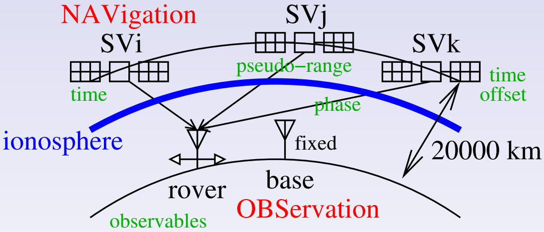
- add_const*, adder_substracter_*, mean* — linear operations
- convertComplexToReal, convertRealToComplex — type conversion
- duppl*_1_to_2 — stream splitting
- expander*, shifter* (bit shifts) — bit shifts
- switch* — flow control

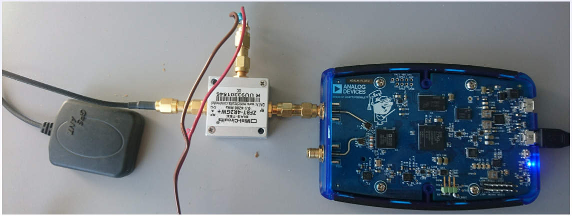
Interface between real/complex and AXI bus or CPU:

- axiStreamTo*, *ToAxiStream, axi_to_dac — AXI to complex/real
- data*_to_ram, data*_dma_direct — FPGA→CPU communication

Why SDR-based GNSS decoding ?

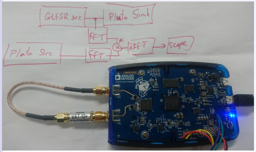
1. Flexibility of adding new features without updating hardware
2. Beyond timing & positioning: access to the raw I/Q stream
   - basic physics (reflectometry)
   - security (phased array for spoofing detection)
   - 1575.42 MHz within range of the PlutoSDR (AD9363 + Zynq SoC)

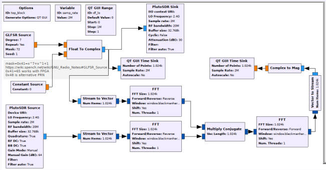


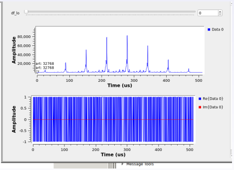


Using the embedded FGPA

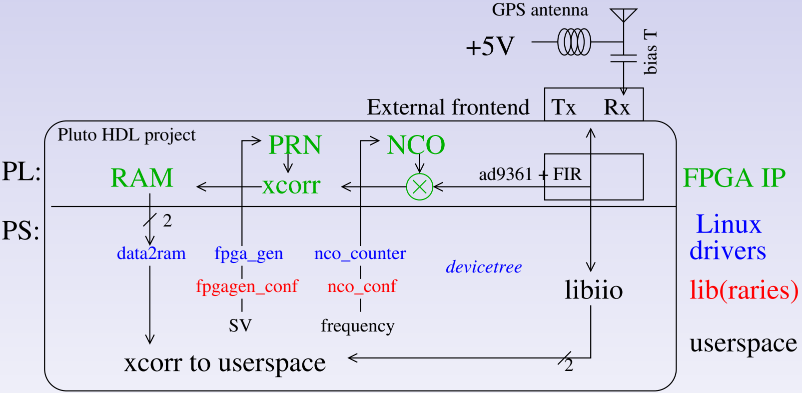
- GNU/Octave implementation: 1 to 2 second/satellite
  $\Rightarrow$ $\simeq$1 min for acquisition depending on frequency steps
- The PlutoSDR Zynq official firmware is only used for data collection and transfer to the PC (bandwidth **limited by USB**)
- Preprocessing on the Zynq FPGA **removes the communication bandwidth bottleneck**
- Making best use of the available resources on the embedded FPGA (PL)
- Possible additional pre-processing on the embedded CPU (PS) running GNU Radio before sending over USB







Principle

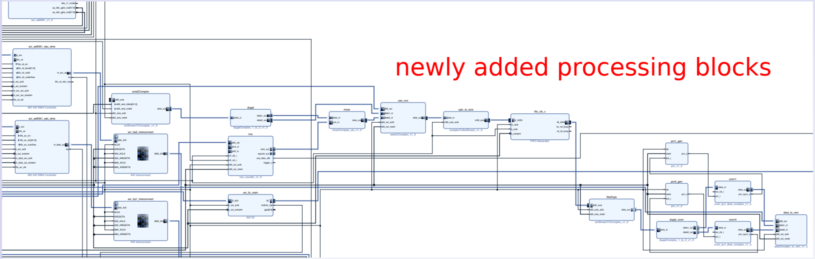


- PL: collect data from AD9363, frequency transposition (NCO), Gold Code generation & correlation
- PS: loop frequency, loop space vehicle number, fetch correlation, control AD9363 (libiio)

Complex interaction between FPGA processing blocks and processor userspace through Linux drivers (modules)

Application to GPS decoding

- TCL scripts define the processing functions, their settings and how they are connected to each other
- Zynq on the PlutoSDR ⇒ Xilinx Vivado (despite platform independence of OscimpDigital)

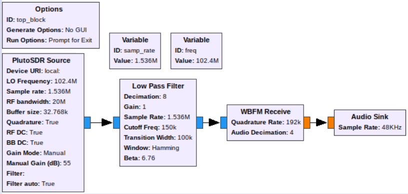


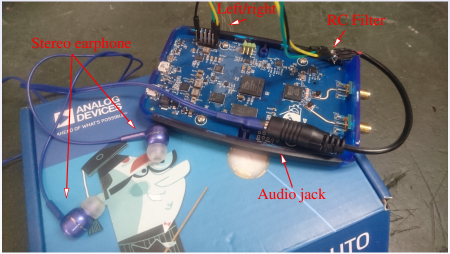
Dual PRN generator and cross-correlation with the received datastream frequency transposed using the NCO.

22 s on Zynq PL (limited by the FPGA area limiting the number of parallel correlations) v.s **108 s on 2.6 GHz PC** (GNU/Octave)
⇒ **move to ADRV9361 to have more resources (WIP)**

PlutoSDR: embedded broadcast FM receiver with audio card output [2]

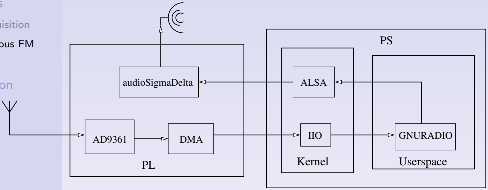
- GNU Radio in PlutoSDR firmware
- add sound card IP in parallel to processing chain
- Python or C++ flowgraph



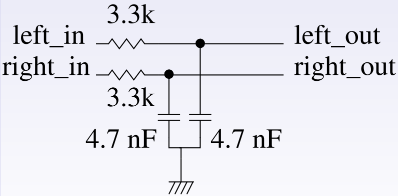


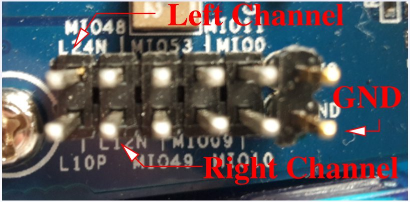
[2] https://github.com/oscimp/oscimpDigital/tree/master/doc/tutorials/plutosdr/99-gnuradio-audio

PlutoSDR: embedded broadcast FM receiver with audio card output



- dual sigma-delta IP for stereo output
- alsa compatible driver
- use local iio backend





Conclusion

- OscimpDigital as a **flexible framework** for assembling signal processing blocks on the FPGA in charge of collecting radiofrequency data
- Platform **independence** (useful investment for Intel/Altera SoC as well)
- **Consistent** IP–Linux kernel module–library–userspace application
- **Perspective**:
  - finalize/validate GNSS parallel Gold Code correlation to ADRV9361 (Zynq 7035 $\gg$ 7010)
  - improve documentation
  - demonstration with FPGA standalone and RiscV softcore

Resources:

`https://github.com/trabucayre/redpitaya` (Buildroot BR2_external)
`https://github.com/oscimp/PlutoSDR` (Buildroot BR2_external)

`https://github.com/oscimp/oscimpDigital` (IP, driver, lib, tools & doc)

Clone repository and submodules:

```
git clone --recursive https://github.com/oscimp/oscimpDigital.git
```

Acknowledgement: PIA platform grant OscIMP