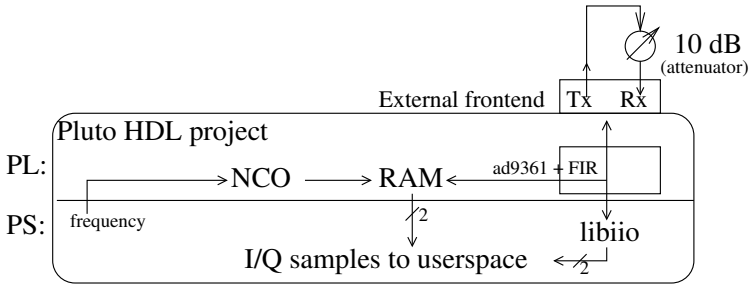
Using ADALM-PLUTO within the OscimpDigital Ecosystem

P.-Y. Bourgeois
pyb2_at_femto-st.fr
May 24, 2019

This tutorial is intended to help you to handle the ADALM-PLUTO board within the OSCIMP framework. It is focused on

- Providing general guidelines to set up the environment
- Performing a Pluto's harware independant check (NCO→RAM, check the data)
- Connecting the Pluto data stream into the RAM and recovering the data



① nco_counter → dataComplex_to_ram

② ad9361 → fir_decimator → axiStreamToComplex → dataComplex_to_ram
fir_decimator → axi_ad9361_adc_dma

Figure 1: Objectives of this tutorial and raw schematics of the processing chains.

Disclaimer

Neither the author nor the developpers of the OSCIMP repositories shall be responsible for misuse of the presented Software and Hardware. All the given codes and guidelines of this tutorial are provided 'as is'.

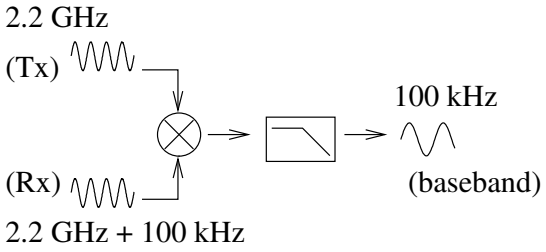
1 Objectives / Experimental set-up

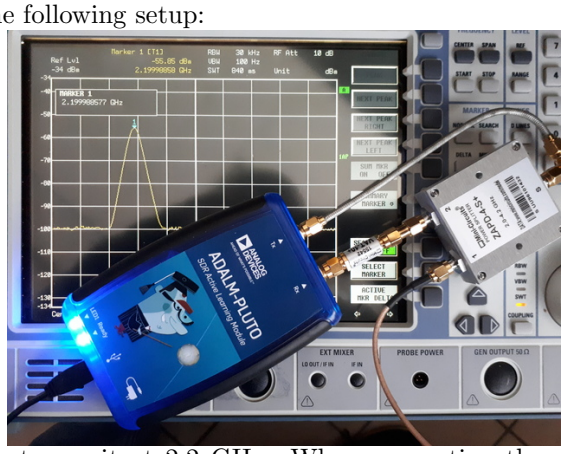
Right out of the box, the ADALM-PLUTO board, when fired up, is able to be fully configurable and flawlessly accessible thanks to *iio* contexts of the **libiio**. Thanksfully AnalogDevicesInc provide the library but also distribute the HDL code:) It becomes then interresting to test the compatibility of the original Pluto HDL project within the OSCIMP ecosystem and beneficiate for extra HDR/SDR features.

For this tutorial, we want to recover I/Q samples of a 100 kHz beatnote. Achieving this result is first assessed by fetching samples from the output of a 100 kHz NCO operating independently of the AXI Stream provided by ADI. Having checked that the samples are properly generated and recovered, we mix the output of the AXI Stream including the I/Q coefficients from the AD9363 and check that the result is consistent. Thus:

- the first part aims at checking the correct bitstream synthesis of the project provided by ADI using an additional NCO and RAM transfer from the OSCIMP ecosystem. In this setup (called sanity check), the NCO acts as a fake 100 kHz beatnote generator, the added IPs having no relationship with the initial firmware,
- in the second part, we connect custom blocks to the AXI Stream from the AD9363 and demonstrate how custom processing can be added to the original ADI processing chain.

In order to create a "beatnote", one can use the following setup:





The **Tx** port of the PlutoSDR board is set to transmit at 2.2 GHz. When connecting the PlutoSDR Sink (select an input level lower than 1 to avoid saturation and unwanted signal beyond the carrier), the Tx transmits nothing but the carrier at 2.2 GHz (i.e. with no signal of interrest nor message). The receiver part is slightly detuned at 2.2 GHz+100 kHz. After demodulation (RF-hardware) and decimating filter (`fir_decimator` block within the PL-side of the FPGA), a remaining "beatnote" at $f_b = 100$ kHz occurs. Once we know in advance the frequency of this beatnote, when the sampling rate is set up at $fs = 50$ MHz (**JUSTIFIER : AXI Stream clock bus**), we also know that a window of $N = 2048$ samples will present around 4 periods of the beatnote ($N/fs/fb$). This "beatnote" trick is an easy trick that is routinely used when developping RF applications, to check the correctness of the data.

On the photography, a splitter has been added (before the 10 dB attenuator) to also send the **Tx** port on a spectrum analyzer which enable to check if the right transmit carrier frequency has been set up through *iio*.

2 Setting up the environment

To complete this tutorial, we suppose some experience in using the OSCIMP ecosystem has been acquired. Also up to date versions of the following repositories must be fetched:

- oscimpDigital (https://github.com/oscimp/oscimpDigital). Remember to recursively clone the repository
  `git clone --recursive https://github.com/oscimp/oscimpDigital`
  and set the `BOARD_NAME` to redpitaya since the PlutoSDR is fitted with the same Zynq model than the Redpitaya
- The BR2_EXTERNAL framework for Analog Device's PlutoSDR Zynq (https://github.com/oscimp/PlutoSDR)
- The HDL project from analogDevices (see further below).

In order to build the HDL project for the ADALM-PLUTO board, only the **hdl** repository (https://github.com/analogdevicesinc/hdl) is needed.

At the time of writing (May 24, 2019), the following versions are functional:

- **adi_hdl** branch **hdl_2018_r2** (401395cdd1980827fd1f7043ce1a10770f666c64)
- Vivado 2018.3

To build the HDL project for the Pluto board, `source` the `settings64.sh` script of your Vivado 2018.2, and `make` the `/somewhere_analogdevicesinc/hdl/projects/pluto`.

(On some computers, several `make` in a row are required to undergo some error messages).

A successful build of the HDL project is returned by the console message:
`Building` **pluto** `project [/pathTo_adi_hdl/projects/pluto/pluto_vivado.log] ...` **OK**

3 Sanity check (NCO→RAM→UserSpace)

This tutorial allows us to use the OscimpDigital blocks with the embedded Zynq of the PlutoSDR board. It is seen as a first-step demonstration of compatibility.

3.1 Design

Once the pluto HDL project is successfully built, open *pluto.xpr* within Vivado.

Add the *oscimpDigital/fpga_ip* repository (Icon "Settings" → "Project Settings" → "IP" → "Repository" → + then Apply and close the window).

Open the block design, right-click, "Add IP", then add the *dataComplex_to_ram* and *nco_counter*. Run the "Connection Automation": the NCO counter and dataComplex_to_ram should connect to the AXI bus and have addresses assigned to their registers: dataComplex_to_ram base address should be 0x43C0_0000 and nco_counter base address should be 0x43C1_0000. If adresses mismatch, keep them in mind as you will need it for the Device Tree overlay.

The next step is to configure the added IPs: we set

- the NCO block `Data Size` to 16 bits, `Counter Size` to 28 bits, and `LUT Size` to 10 bit,
- the Data Complex to RAM block with `Data Size` to 16 bits, `Number of Inputs` to 1, and `Number of Samples` to 1024.

Additional connections (Fig. 2):

- NCO
  - `ref_rst_i` → `s00_axi_reset` on the same block
  - `ref_clk_i` → `s00_axi_aclk` which refers to the PS7's FCLK_CLK0 at 100 MHz. We might as well clock and reset all blocks on the AD936x signals.
- NCO → RAM
  - `sine_out` → `data1_in`

At last, click on "Generate Bitstream", and once the synthesis is done, place the `/pathTo_adi/hdl/projects/pluto/pluto.runs/impl_1/system_top.bit` into the `/somewhere/PlutoSDR/board/pluto` BR2_EXTERNAL directory.

3.2 Oscimp library / Linux Drivers

The library **liboscimp_fpga** should be compiled for your environment. We have selected to statically link the library to the application to avoid confusion with multiple library versions: when compiling the userspace application using the `Makefile` provided by `module_generator` (see below) by adding `USE_STATIC_LIB=1` after the `BASE_NAME`

Also both drivers for the NCO (`nco_counter_core.ko`) and RAM (`data_to_ram_core.ko`) should be compiled.

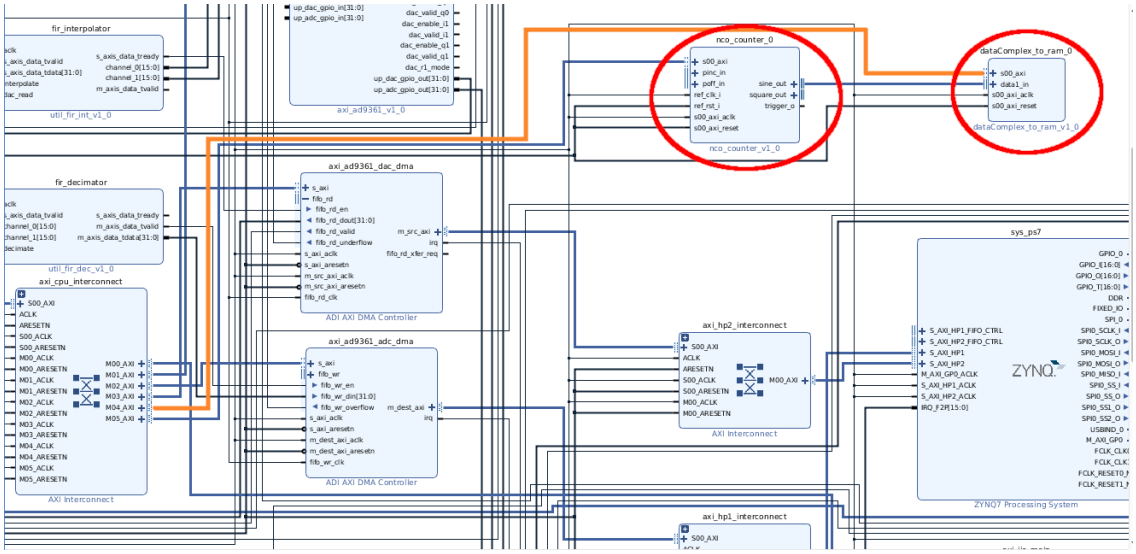


Figure 2: Connection of the custom blocks to the PlutoSDR firmware.

3.3 Device Tree Overlay

The devicetree overlay will be generated by `modules_generator`: the configuration file (`config.xml`) looks like

```
<?xml version="1.0" encoding="utf-8"?>
<drivers name="PlutoSDR_custom" version="1.0">
  <driver name ="data_to_ram" >
    <board_driver name="data00" id = "0" base_addr="0x43c10000" addr_size="0xffff" />
  </driver>
  <driver name ="nco_counter" >
     <board_driver name="nco00" id = "0" base_addr="0x43c00000" addr_size="0xffff" />
  </driver>
</drivers>
```

where `data00` and `nco00` are the `/dev` entries and the base addresses should match thoses provided by Vivado. A few updates must be brought to the output of `module_generator` after running `module_generator -dts config.xml`

- in dans le `.dts`: replace `target = <&fpga_full>;` with `target = <&fpga_axi>;` and comment out the bitstream name
- in the `Makefile`, add `USE_STATIC_LIB=1` and `LDFLAGS+=-liio` after the project name but prior to the `Makefile.inc` inclusion
- in the shell script, comment the lines creating the firmware directory and attempting to copy the bitstream to this location.

The last step is then to rebuild the firmware (`pluto.frm`) :

- remember that we have copied the bitstream to the PlutoSDR directory in `/somewhere/PlutoSDR/board/pluto`, hence providing buildroot with the appropriate bitstream named `system_top.bit`.
- `cd pathTo/buildroot-version/`
- `make`

3.4 Flash the pluto

NB: For the moment, the repository PlutoSDR provides only support for rootfs and linux (the bootloader part still remains to be added). Hence, any change of the bistream requires to perform the steps presented here:

- Fire up the PlutoSDR // wait for the mass-storage to be available (via `dmseg -w` for example)
- `mount /dev/sdX1`[1] `/mnt/somewhere/`
- `cp pathTo/buildroot-version/output/images/pluto.frm /mnt/somewhere/`
- `eject /dev/sdX1`

You should see the **blue LED1** flashing quickly, thus do not disconnect nor touch anything until the PlutoSDR has rebooted or you may brick the board at this stage.

Alternatively, the DFU image can be transfered to the PlutoSDR flash by rebooting, from the target GNU/Linux prompt, to DFU mode with

```
device_reboot sf
```

and then running on the host personnal computer

```
dfu-util -a firmware.dfu -D my_pluto_firmware.dfu
```

with `my_pluto_firmware.dfu` matching the name of the image to be flashed. Again wait for the transfer to complete and the message stating that no error was detected to appear.

The PlutoSDR is now ready to be tested.

3.5 Application

The application-side makes use of *libiio* as well as *liboscimp*. For the former, one can find examples from M. Hennerich presentation at GNU Radio Conference 2018

https://www.gnuradio.org/grcon/grcon18/presentations/plutosdr/

or following guidelines written in the ADI wiki website

https://wiki.analog.com/resources/tools-software/linux-drivers/iio-transceiver/ad9361

and

https://wiki.analog.com/university/tools/pluto/controlling_the_transceiver_and_transferring_data

to tweek the configuration.

For this part, we have just changed Hennerich's snippet code to the desired configuration of TX and RX carriers, Sampling rate, the "process" part only consists of writing the ram buffer into a data file.

An example file is provided in the folder **get_data/nco_data2ram**. Running this example requires loading the devicetree overlay and the kernel modules: all operations are taken care of by the bash script in the application directory. The `dmesg` output should display something like

```
[<c01d64b0>] (SyS_read) from [<c01070a0>] (ret_fast_syscall+0x0/0x48)
---[ end trace 3cf21b0945e698e4 ]---
data_to_ram_core: loading out-of-tree module taints kernel.
probing data00 with dts
name: data00 4 6
dataToRam 43c00000.data00: data00: Add the device to the kernel, connecting cde
dataToRam 43c00000.data00: 6data00 loaded
probing nco00
name: nco00 4 5
nco_counter 43c10000.nco00: nco00: Add the device to the kernel, connecting cde
nco_counter 43c10000.nco00: 6nco00 loaded
```

with the first two lines kernel messages resulting from the bitstream loading, then the `data_to_ram` driver and the `nco` driver. The two `/dev/nco00` and `/dev/data00` defined in the devicetree source should have been created.

[1] X is the index of the pluto mass-storage, e.g. /dev/sde1

3.6 Testing

The example is run from the PlutoSDR: the `/dev/data00` device is opened, the output of the NCO fed to this block read from the PS, and stored in a file. Plotting the resulting dataset yields charts similar to those shown in Fig. 3.

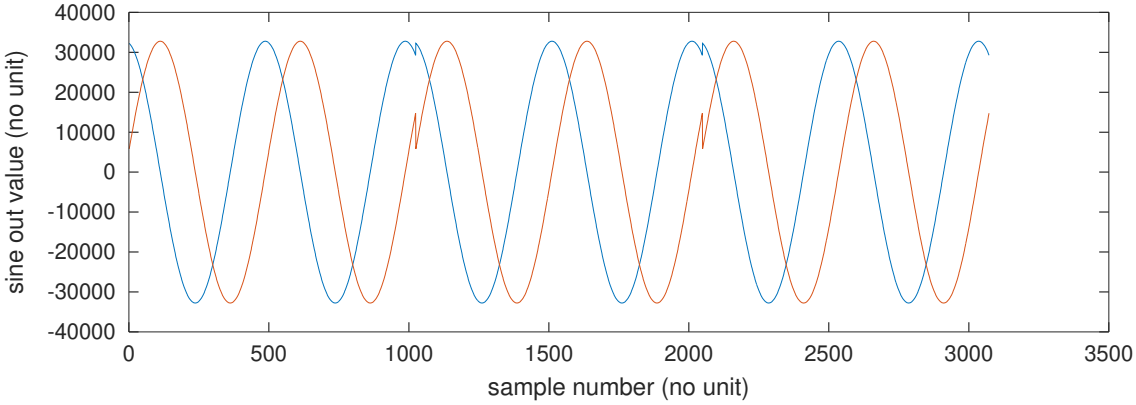


Figure 3: Collected I/Q datasets. The 500 samples/period match the expected sampling rate of 5 MS/s as a 100 kHz sine wave is generated by the NCO. Three successive (discontinuous) datasets were collected.

4 PlutoSDR data stream → RAM → Userspace

This second tutorial is a demonstration of the compatibility of the OSCIMP ecosystem with the use of the existing Pluto HDL. Once complete, you should be able to plug any DSP task to the initial design and perform extra processing chains (Figs. 4 and 5).

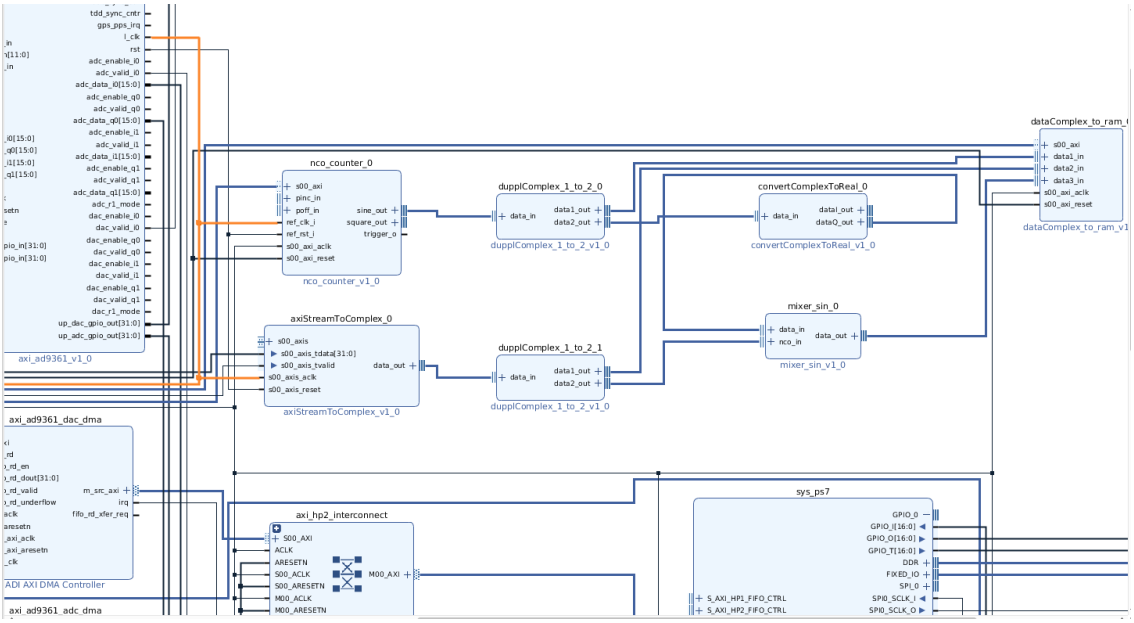


Figure 4: Signal processing chain as defined in the Vivado 2018.3 graphical user interface (added blocks are on the top right and include the nco_counter, axiStreamToComplex, two duppComplex, convertComplextoReal, mixer_sin and finally dataComplex_to_ram).

This time, since data from the AD9363 will be used in the flow chart, a consistent clocking and signal reset distribution must be used:

- connect `ref_clk_i` of the `nco_counter` to the `axi_ad9361` IP `l_clk` (as is the `s00_axis_aclk` of the `axiStreamToComplex` block)
- connect `ref_rst_i` of the `nco_counter` to the `axi_ad9361` IP `rst` (as is the `s00_axis_reset` of the `axiStreamToComplex` block)
- these clock and reset signals are then propagated to the other processing blocks through their interfaces.

Failing to use these connections will result in timing errors including negative WNS and TNS values related to the fact that the mixer is fed data from two different clock domains – 100 MHz for the AXI bus and 50 MHz for the AD9361.

Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
sys_ps7					
Data (32 address bits : 0x40000000 [1G])					
dataComplex_to_ram_0	s00_axi	reg0	0x43C0_0000	64K	0x43C0_FFFF
axi_ad9361	s_axi	axi_lite	0x7902_0000	64K	0x7902_FFFF
axi_ad9361_adc_dma	s_axi	axi_lite	0x7C40_0000	4K	0x7C40_0FFF
axi_ad9361_dac_dma	s_axi	axi_lite	0x7C42_0000	4K	0x7C42_0FFF
axi_iic_main	S_AXI	Reg	0x4160_0000	4K	0x4160_0FFF
nco_counter_0	s00_axi	reg0	0x43C1_0000	64K	0x43C1_FFFF
axi_ad9361_dac_dma					
m_src_axi (29 address bits : 512M)					
sys_ps7	S_AXI_HP2	HP2_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
axi_ad9361_adc_dma					
m_dest_axi (29 address bits : 512M)					
sys_ps7	S_AXI_HP1	HP1_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF

Figure 5: Address map generated by Vivado: the devicetree source file must match this configuration for the Linux drivers to reach registers at the right addresses.

Three datastreams lead to the DataToRAM block: the (complex) NCO output as addressed previously, the raw ADC (complex) data resulting from the I/Q demodulation within the AD9363 RF frontend, and the output of the mixer of the NCO with the ADC I/Q stream. Since the ADI ADC is fed to a FIR decimator with no interface on the AXI Stream bus, a direct connection to the axiStreamToComplex block is possible. Since the complex valued output exhibits an interface, the resulting stream must be dupplicated (dupplCompl) when multiple processing – here communication and mixing – are to be performed on the complex valued datastreams.

The resulting datasets are plotted on Fig. 6, left with the PlutoSDR disconnected from its input (noise measurement on the ADC) and right with the RF output connected to the input through a 10 dB attenuator.

The original PlutoSDR used with GNU Radio remains functional despite the updated bitstream. The same result than the one demonstrated from the OscimpDigital framework but running on the host computer is shown in Fig. 7.

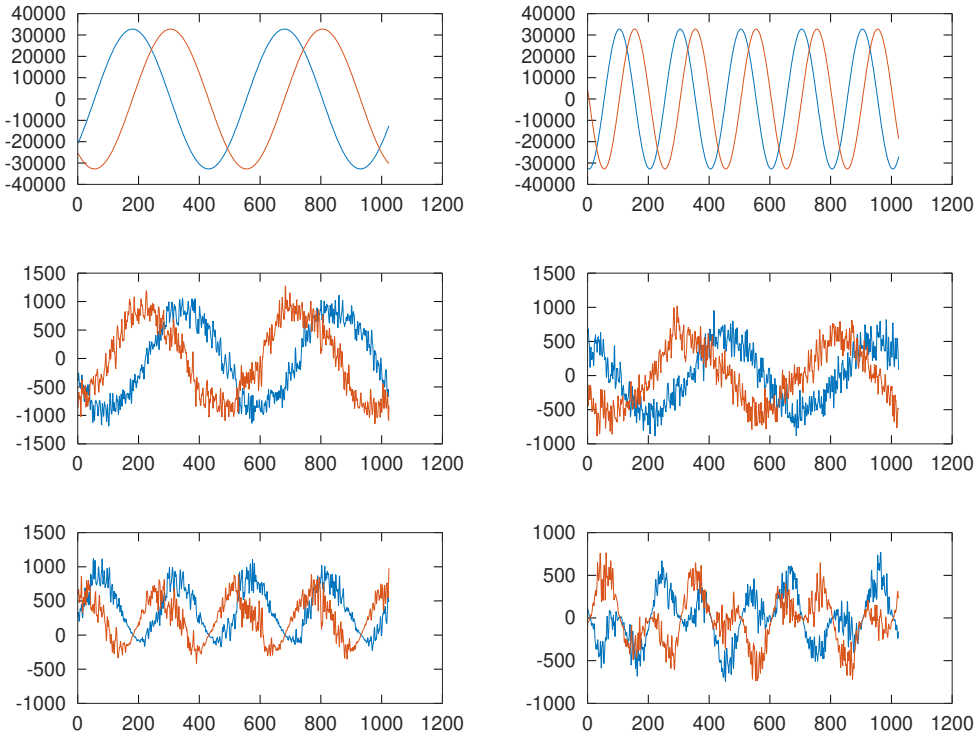


Figure 6: Measurements results with a 100 kHz local oscillator (left) and with (right) a 250 kHz local oscillator, while in both cases the output is offset by 100 kHz from the input. Top to bottom: NCO, measured I/Q streams, and mixer output.

5 TCL update of the original ADI PL configuration

Rather than bothering with the graphical user interface, adding new functionalitites brought by the OscimpDigital framework to the original ADI PL configuration file is most efficiently achieved by updating the TCL script.

The TCL script provided by ADI is located at `/somewhere/hdl/projects/pluto/system_bd.tcl` and must updated with the following:

- at the beginning of the file, insert the location of the OscimpDigital IP repository with

```
variable fpga_ip    $::env(OSCIMP_DIGITAL_IP)
set_property  ip_repo_paths [list ${fpga_ip} ${lib_dirs}] [current_project]
update_ip_catalog
```

- append the design with the additional blocks:

```
#nco
ad_ip_instance nco_counter nco
ad_ip_parameter nco CONFIG.DATA_SIZE 16
ad_ip_parameter nco CONFIG.COUNTER_SIZE 28
ad_ip_parameter nco CONFIG.LUT_SIZE 10

ad_connect axi_ad9361/rst nco/ref_rst_i
ad_connect sys_rstgen/peripheral_reset nco/s00_axi_reset
ad_connect axi_ad9361/l_clk nco/ref_clk_i
```

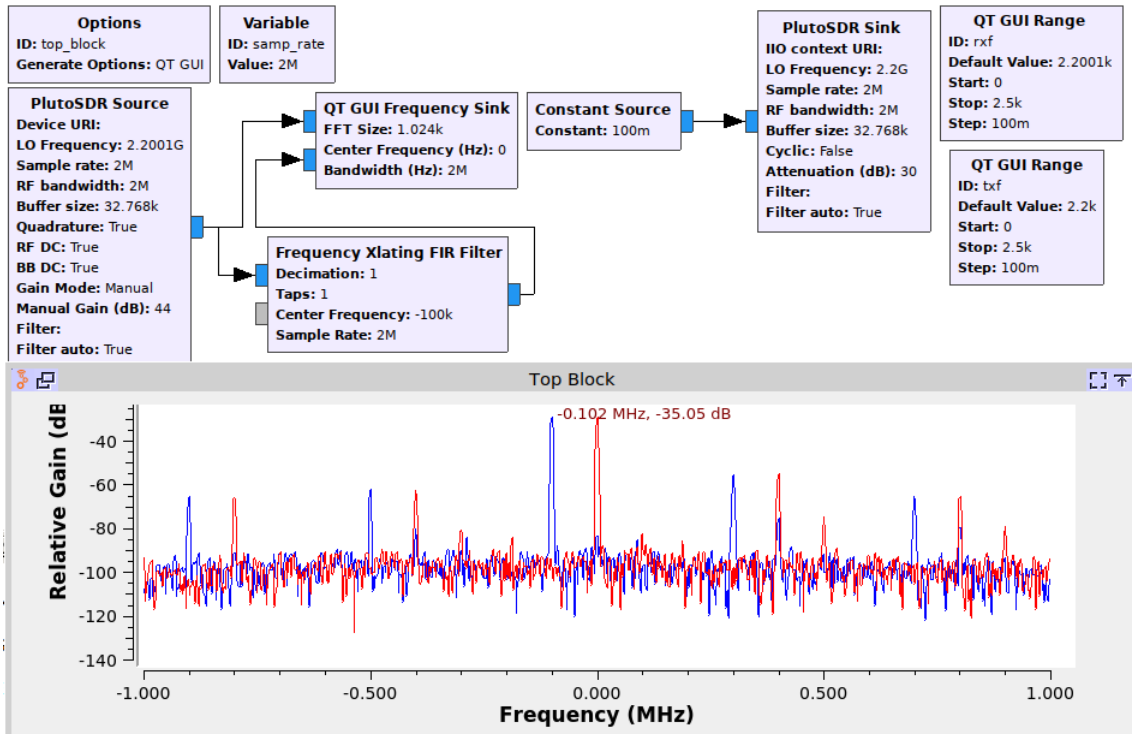


Figure 7: GNU Radio flowchart accessing the PlutoSDR as source and sink with the updated bitstream configuring the PL.

```
ad_cpu_interconnect 0x43C00000 nco

#dataComplex
ad_ip_instance dataComplex_to_ram data_to_ram
ad_ip_parameter data_to_ram CONFIG.NB_INPUT 1
ad_ip_parameter data_to_ram CONFIG.DATA_SIZE 16
ad_ip_parameter data_to_ram CONFIG.NB_SAMPLE 2048

ad_connect nco/sine_out data_to_ram/data1_in
ad_connect sys_rstgen/peripheral_reset data_to_ram/s00_axi_reset

ad_cpu_interconnect 0x43C10000 data_to_ram
```

- update the `Makefile` by removing

  ```
  M_DEPS += ../../library/xilinx/common/ad_iobuf.v
  M_DEPS += ../../library/axi_ad9361/axi_ad9361_delay.tcl
  ```

  and replacing with

  ```
  M_DEPS += $(ADI_HDL_DIR)/library/xilinx/common/ad_iobuf.v
  M_DEPS += $(ADI_HDL_DIR)/library/axi_ad9361/axi_ad9361_delay.tcl
  ```

  with `ADI_HDL_DIR` set to the `/somewhere/hdl/` location,

- in the `Makefile`, remove

  ```
  include ../scripts/project-xilinx.mk
  ```

  and replace with

  ```
  include $(ADI_HDL_DIR)/projects/scripts/project-xilinx.mk
  ```

- in the `system_project.tcl`, replace

  ```
  source ../scripts/adi_env.tcl
  ```

  with

  ```
  variable adi_hdl_dir $::env(ADI_HDL_DIR)
  source $adi_hdl_dir/projects/scripts/adi_env.tcl
  ```

Following these updates, `make` with synthesize the bitstream which can be used as described earlier in the text to generate the `.frm` and `.dfu` images to reflash the PlutoSDR.