Redpitaya: third example

G. Goavec-Mérou, J.-M Friedt

January 12, 2020

This tutorial is a sequel to the previous one on which it is based. In addition to copying the ADC input to the DAC output, we wish to add to the stream of data sent to the user (PS) a filtered copy of one of the ADC inputs. The Finite Impulse Response (FIR) filter will process the 125 MS/s stream with integer coefficients configured from userspace (PS) (Fig. 1). Furthermore, we wish to extend capability beyond a single complex stream: we will now stream two parallel real data, which could be extended to more than two channels.

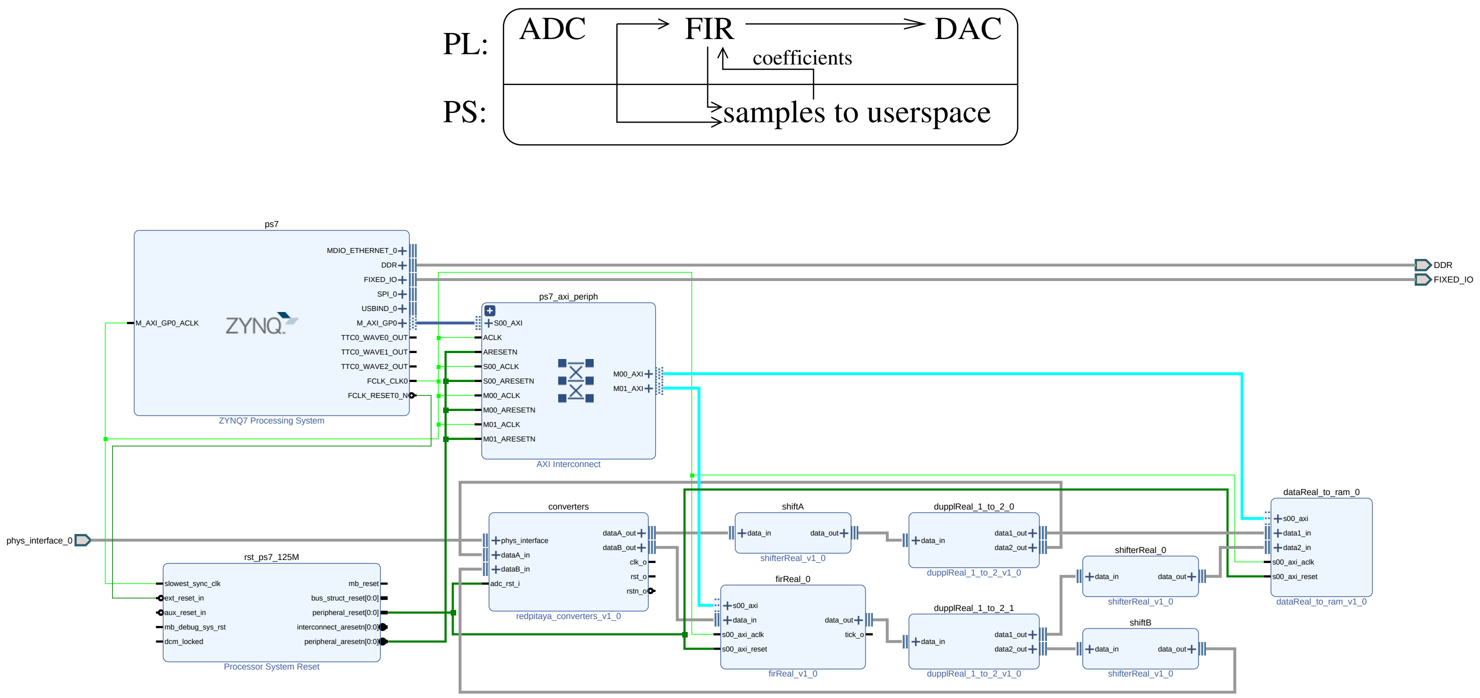


Figure 1: Schematic of the objective and final processing chain described in this document

1 FIR configuration

The ADC to DAC and ADC to RAM blocks are the same as before, with stream dupplication every time a new datastream must reach two users. The novely is now the use of a FIR filter, whose input width the match the incoming datastream width.

Double click on the FIR block and update the Data Width (`Data In Size`) to 14 bits (or 16-bits for the 16-bit Redpitaya). The input data width (in bits) is called P.

The coefficient width M (`COEFF Size`) will define the size of the data transferred from the PS to the PL, as stored as a column of decimal values written in human readable ASCII format. The number of coefficients N (`Nb COEFF`) and number of bits defining each coefficient are defined at synthesis time and must match the largest expected filter (as well as available resources).

Theoretically, $Q = P + M + \log_2(N)$ bits are generated at the output of the FIR convolution. Practically, this number of bits is grossly over-estimated, and truncation can be considered based on the knowledge of the coefficients defining the FIR. Taking the $\log_2$ of the sum of the absolute values of the coefficients will give a much more conservative number of bits relevant at the output. While all internal computations are performed on Q bits, the output can be *truncated* to the *lower* Output Size bits (`Data Out Size`).

The user might then decide to shift to the right the result to drop some of the least-significant bits, as illustrated here with the selection of outputting 16 bits from the FIR filter (e.g. for further computations in the PL) and shifting the 2 least significant bits to send 14-bit wide words (as expected from the legacy Redpitaya ADC datastream – 16 bits for the 16-bit Redpitaya) to the PS.

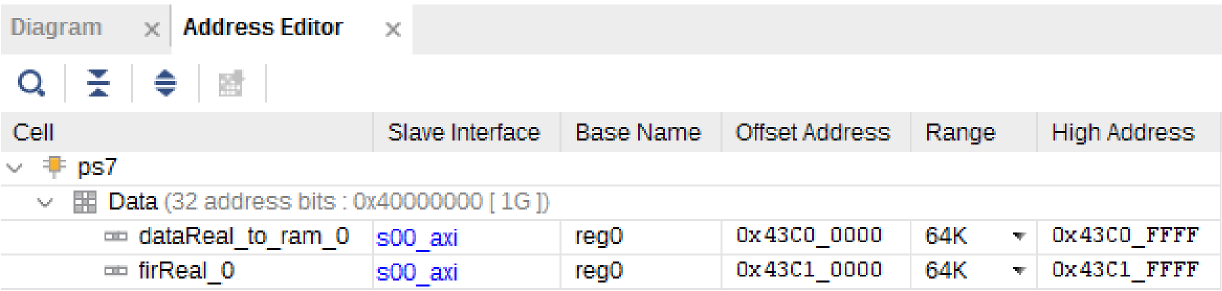
2 PS: Linux kernel driver

Rather than using a single complex stream to communicate with the PS, we have now selected the `dataReal_to_ram` to define multiple real streams. The driver is the same than `dataComplex_to_ram`, so that this part of the XML may be left unchanged (memory address must be checked, as shown for example on Fig. 2), in addition to which we wish to communicate with the FIR to define the coefficients. This time, the `module_generator` XML configuration file should look like

```xml
<?xml version="1.0" encoding="utf-8"?>
<project name="tutorial4" version="1.0">
  <ips>
    <ip name ="dataReal_to_ram" >
      <instance name="data00" id = "0"
        base_addr="0x43c00000" addr_size="0xffff" />
    </ip>
    <ip name ="firReal" >
      <instance name="fir00" id = "0"
        base_addr="0x43c10000" addr_size="0xffff" />
    </ip>
  </ips>
</project>
```

3 On the Redpitaya ...

For communicating with the FIR, we will benefit from an existing library function as implemented in `liboscimp_fpga` (see $OSCIMP_DIGITAL_LIB). This library must be installed by:



Cell	Slave Interface	Base Name	Offset Address	Range	High Address
ps7					
Data (32 address bits : 0x40000000 [1G])					
dataReal_to_ram_0	s00_axi	reg0	0x43C0_0000	64K	0x43C0_FFFF
firReal_0	s00_axi	reg0	0x43C1_0000	64K	0x43C1_FFFF

Figure 2: Address range used by the IPs as defined by Vivado.

```
make && make install
```

to place the .so in `buildroot target` dir (the board must be flashed again), or by

```
make && make install_ssh
```

to transfer the .so to the embedded board's `/usr/lib` directory by using `ssh`. By default target IP is 192.168.0.10, which can be updated to match another network configuration with

```
make && IP=AA.BB.CC.DD.EE make install_ssh
```

```c
#include <stdio.h>
#include <stdint.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include "fir_conf.h" // library for communicating with the FIR


int main()
{int k;
 char c[16384];
 int fi,fo;
 fi=open("/dev/data00",O_RDWR);
 fo=open("/tmp/data.bin",O_WRONLY|O_CREAT,S_IRWXU);
 fir_send_confSigned("/dev/fir00","coefs.txt",32);
 for (k=1;k<5;k++)
  {read(fi,c,16384);
   write(fo,c,16384);
  }
 close(fi);
 close(fo);
 return 0;
}
```

Examples of outputs of running this program are given in Figs. 3 and 4: all filter lengths are set to 32, with a varying number of non-null values set to 1 in the beginning of the filter. Since in this design the FIR output is routed to the DAC, an oscilloscope is first used to monitor the generated signal frequency and the filtered signal amplitude (oscilloscope “Measure” functions set to “Frequency C1” and “Peak to Peak Amplitude C3”). In all cases the generated signal is 1.5 V_{pp}.

The transfer function of a FIR filter is the Fourier transform of its coefficients so that the transfer function of rectangular window FIR filters are sinc functions. The expected transfer functions are displayed in Fig. 3 (top) and the experiemental amplitude of the FIR output for varying filter window width and notch frequency are shown in Fig. 3 (bottom). The notch frequency are in excellent agreement, the amplitude measurements are plagued by poor resolution of the oscilloscope measurement.

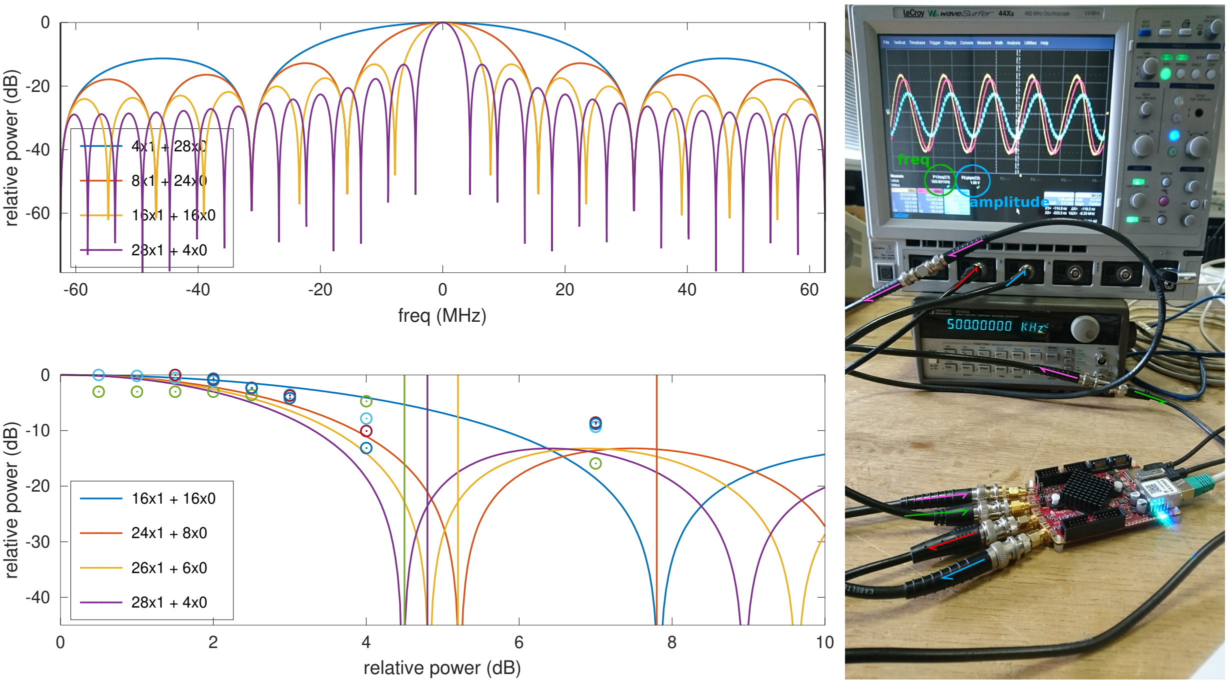


Figure 3: Left: transfer function of rectangular FIR windows (top) and measured amplitude and notch frequencies (bottom). On the bottom chart, vertical lines are located at the observed notch frequency. Right: experimental setup, including arrows showing that the same driving signal is feeding both ADCs, and the two DAC outputs are either copies of one ADC or the filtered output of the other ADC.

Beyond these oscilloscope measurements, datasets can be dumped from the `data_to_ram` for further processing. Various test FIRs are designed by clearnig to 0 all coefficients except the first ones, which might be either set to 1 for the first 16 coefficients, or set to 4 for the first 4 coefficients (keeping the total power constant). Since sampling is at 125 MS/s, the cutoff frequency of a 16-tap long coefficient must be about 7.8 MHz: most of the recorded datasets (1 or 6 MHz) are not affected by the filter except for the bottom one in which the signal frequency becomes close to the cutoff frequency of the filter. The group delay (time needed between filling input and providing the first output) is observed as the phase shift between input and output (Fig. 4)

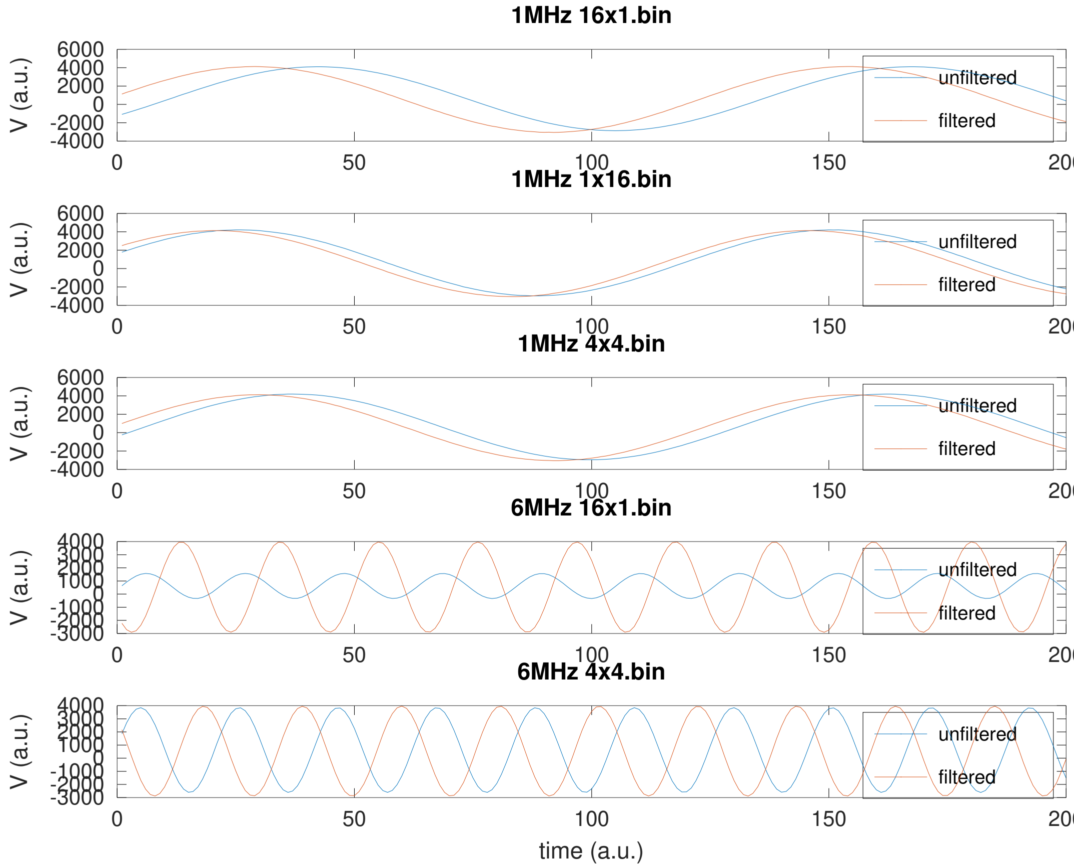


Figure 4: Filtered output (blue) with respect ot the input (red) driving signal: `data_to_ram` outputs.

Collecting the filtered output on the `data_to_ram` device demonstrates how the FPGA can be efficiently used either to process the stream straight from the acquisition interface through the various processing blocks, or as co-processor from data provided by the CPU and running through the FPGA before being recovered by the CPU.