Redpitaya: third example

G. Goavec-Mérou, J.-M Friedt

December 20, 2018

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.
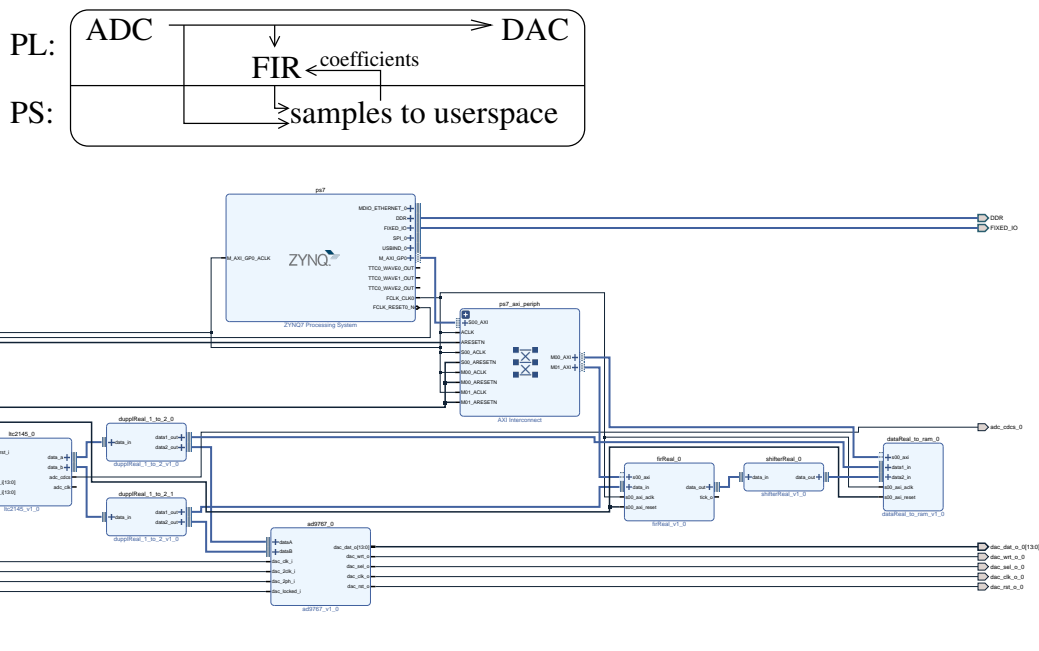

Figure 1: Schematic of the objective and final processing chain described in this document

1 FIR configuration

The ADC to DAC and ADC to RAM blocks are the same as before, with stream duppli-cation 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. 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 ADC datastream) 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"?>
<drivers name="tutorial4" version="1.0">
  <driver name ="data_to_ram" >
    <board_driver name="data00" id = "0"
      base_addr="0x43c00000" addr_size="0xffff" />
  </driver>
  <driver name ="fir" >
    <board_driver name="fir00" id = "0"
      base_addr="0x43c10000" addr_size="0xffff" />
  </driver>
</drivers>
```

The name of the appropriate module has been found by looking for a directory with the name than the IP in the `fpga_driver` directory.

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:

```
make && make install
```

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

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/types.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 Fig. 3: various test FIRs are designed by selecting all coefficients except the first ones, which might be either set to 1 for the first 16 coefficients, or 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.
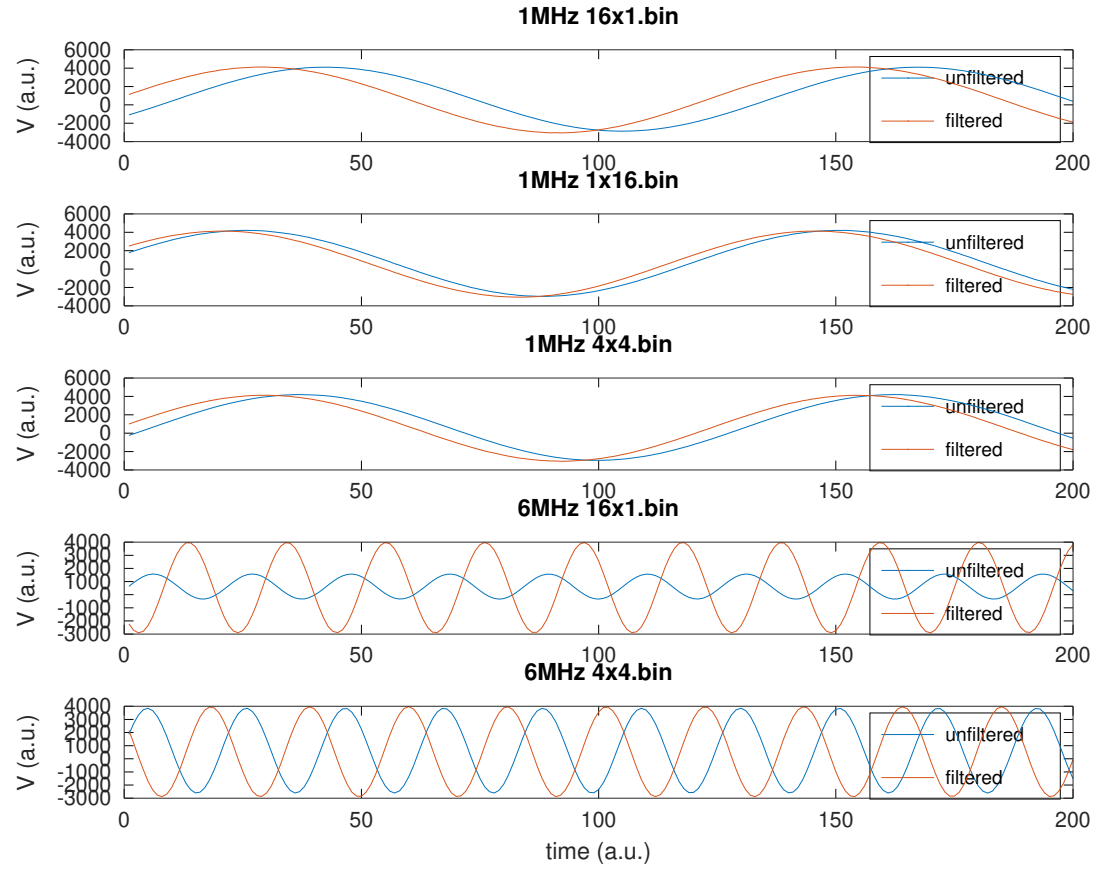

Figure 3