Lab 5 – hardware implementations

Example entry test questions

- Sketch a graph of {2nd order FIR *or* 1st order *or* IIR biquadratic IIR section} using delay/scale/add blocks.
- Why is saturation arithmetic safer for signal processing than pure two's complement?
- How large is magnitude of rounding error if we use 11 bit signed fixed-point arithmetic (express the magnitude in percent of the full signal scale)? What is the ratio of error (noise) power to the power of full-scale signal?

Resources

In the following exercises we will implement signal processing algorithms using a programmable digital hardware device. The device will be a "Zynq" chip, which consists of an ARM family microprocessor and an FPGA (Field-Programmable Gate Array). This chip, together with an A/D and D/A converter is housed in National Instruments "MyRio" development device and will be programmed via the LabView environment.

We will process signals on the FPGA part of the chip, using ARM only for communication with PC. The PC will be needed for:

- downloading the cofiguration bitstream to FPGA,
- supplying variable parameters (e.g. filter coefficients),
- displaying signals (however we will prefer to use an external oscilloscope).

The ARM processor is referred to as "RT" (realtime) processor by National Instruments environment.

The FPGA, which is for us the main part of the hardware will communicate with A/D and D/A converters and do the delay-multiply-sum operations – thus, actually, after being configured, it will do all the processing on its own.

In a standalone system, the FPGA configuration may be stored in a tiny ROM chip on the same board; the system with PC is typically used for development and experiments (but this is what we do).

You will need: MyRio, USB cable type A-B, power unit, 2x 2BNC⟶ miniJack cables. MyRio number must be the same as your table number (license sticks to the PC computer...).

Experiments

Italics denote optional tasks. **Bold suggests what should be in the report**

Connect MyRio to the USB and power. A "MyRio USB Monitor" will appear on your screen – close it.

1. Run a trivial filter on FPGA. The filter will be a second-order FIR with coefficient vector [1, 0, 0], so actually it will be pass-thru. You will use a precompiled FPGA configuration (to save time).
    - Copy the folder `D:/LMyRio/FIR-FPGA` to your desktop. This will allow you to edit your copy of the design. If the destination folder already exists, overwrite it.
    - Open your FIR-FPGA folder and click `CYPS_MYRio_FIRFPGA.lvproj`

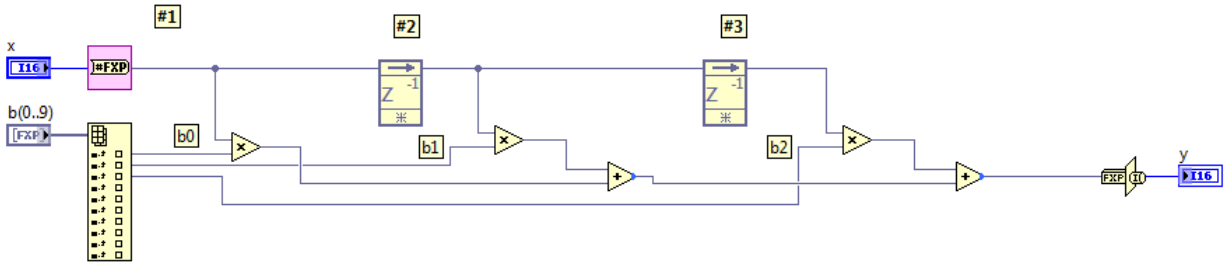

Figure 1: FIR implementation in FPGA

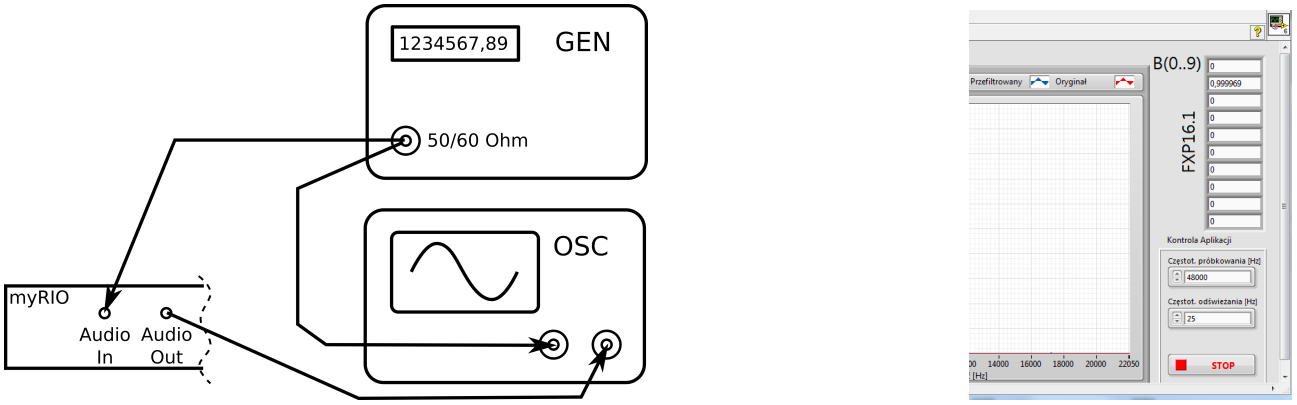

Figure 2: Connections for testing FPGA filter and RT_main window with coefficient control

- Unfold `MyRio-1900` and open `RT_Main.vi` – this is the code for the ARM processor, which will be used for downloading the configuration to FPGA, supplying coefficients and displaying signals.
- Unfold `Chassis MyRio` $\longrightarrow$ `FPGA Target`, open `FPGA Main.vi` (main "wrapper" for the processing code) and `FPGA FIR.vi` (the processing code itself). Now you see the "front panel" i.e. the input and output declarations.
- Within each of FPGA windows press ctl-E to open the graphical view of the implementations. Take a look at the filter implementation. The two "FXP" elements are actually marking – for the FPGA compiler – the fact that calculations will be done with fixed-point numbers. A box with "b(0..9)" connector is an interface for supplying coefficients from ARM. The rest should be obvious (but DO spend a minute on understanding it).

  Note that we use generic elements of the schematic – registers, multipliers and adders. We do not use explicitly the Xilinx-specific "DSP48" block which you knnow from the lecture – the compiler will probably configure it better than we would....
- Connect a generator and an oscilloscope to the MyRio jack terminals (see figure). Use right channel – black BNC connector. The left (red) one does not go through the filter in the default configuration.
- Verify the coefficients and sampling frequency in the RT_Main window. Note that you cannot have "+1" in fractional fixed-point – the maximal value is $1 - 2^{-15}$.
- Run the whole LabView code by pressing "Run"  at the RT_Main window This will start the PC part, initialize ARM (you may need to confirm "Save"), and download the configuration bitstream to FPGA.
- Check that the signals you see actually pass through the MyRio (change frequency, disconnect cables etc.).

- **Plot** the output signal (from oscilloscope!) for a low frequency sinusoid (less than 1/10 of f_s), investigate the "staircase" effect. Discuss it with colleagues and try to **describe** the effect and its causes. **Hint:** Use the RUN/STOP button of the oscilloscope to freeze picture.
- Switch to rectangular signal shape and try to **measure the delay between input and output signal. Hint:** Change the frequency to make sure which output edge corresponds to which input edge.
- Change filter coefficients to [0, 1, 0] and **measure the delay again**. **Hint:** In order to read new coefficients you will have to stop the hardware (use big "STOP" button) and start it again. 
- Switch back to sine wave and check what happens when input signal frequency is above Nyquist range. Use different signal shapes. **Describe the effects**.

2. Use Matlab to design a 2-nd order FIR with a notch (i.e. stopband) at a specified frequency (the teacher will specify the frequency between zero and $f_s/2$) **Hint:** With such a low order `poly()` and `freqz` will be easier than `fdatool()`.

   Write down the coefficients, **sketch** the designed frequency response.

   Change the coefficients to the designed ones. Note that due to the fixed-point implementation of hardware filter there are some limitations (e.g. $|b_i| \leq 1$) – you may need to rescale coefficients.

   Use generator (make sure the input signal amplitude is not more than 1 V) and oscilloscope to verify the frequency response in 2–4 selected frequency points (**mark your measurements on the previous sketch, comment**).

3. Open the design of `FPGA FIR.vi` and modify it to create a FIR filter with order 6.

   - **Sketch your idea of the 6th order FIR schematic**
   - Copy-paste the hardware drawing, connect wires.
   - Extend (dragging with mouse) the object which passes the coefficients from the PC.

   Note that you have changed the hardware connections – you have to *recompile the FPGA configuration*. Find the outermost project window (*.lvproj). Right-click on the `Project...` ⟶ `MyRio....` ⟶ `Chassis` ⟶ `FPGA target` ⟶ `Build specification` ⟶ `FPGA FIR Build` and select `Rebuild`; confirm with "OK".

   The compilation will last about 5 minutes – you may take a break or use Matlab in the meantime to design the filter (next item).

4. Design a filter by yourself: find a FIR coefficient set for your own specification (e.g. bandstop for 4-12 kHz) (use Matlab – functions that you know from previous lab, or simply `fdatool()`).

5. Copy coefficients to the FPGA control window on PC and run the program ("Run"  at the RT_Main window)– this will download configuration and start the FPGA.

6. Finally, test the frequency response with generator and oscilloscope. In the report **sketch** the designed frequency response and **mark** the results of your check.
   Hint: Since your filter uses 12 bits for calculations, consider how many decimal digits to use when copying coefficients.

   A useful trick: in `fdatool` use "Export ⟶ coefficients ⟶ to workspace", then in Matlab you'll see the coefficients in a variable which can be displayed.

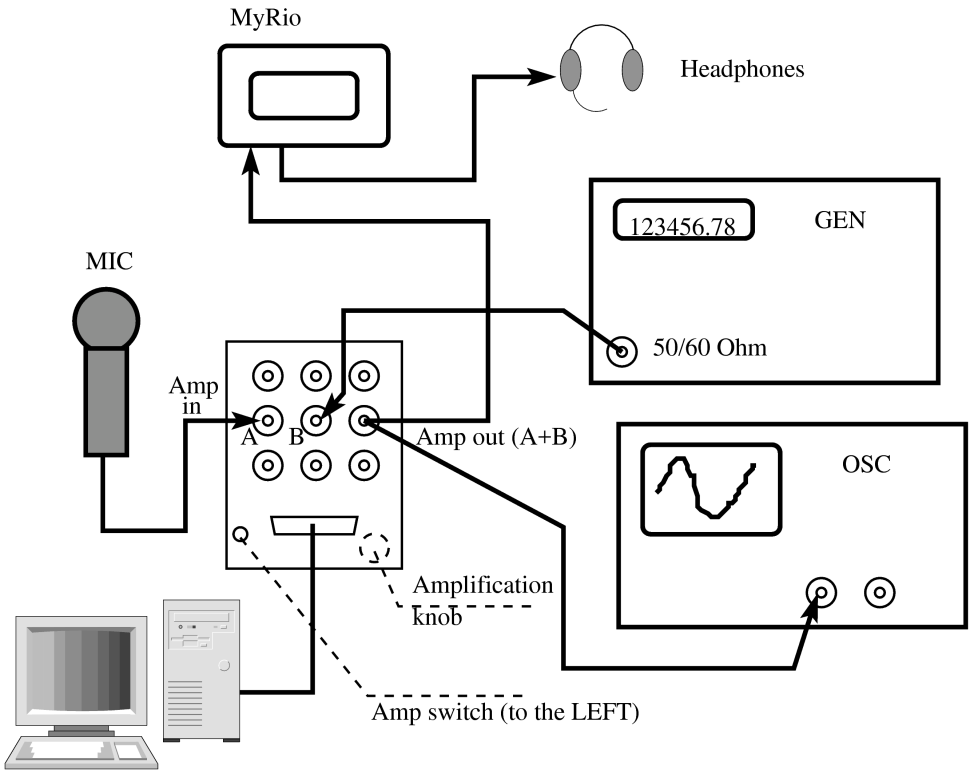

Figure 3: Microphone plus interference connections

7. Verify your filter behavior with large signal magnitude. **Note** maximum magnitude on input which still produces correct output all over the useful frequencies.

   - Scale the coefficients down to allow larger signal at the input. **Note** what happened to output magnitude.
   - Check the properties of arithmetic blocks (right-click on a block$\longrightarrow$ "properties"$\longrightarrow$ "output configuration"). Sketch your filter graph and mark maximum possible magnitudes at all nodes. Try to increase number of bits in the arithmetic blocks to improve the dynamic range, also switch to "saturation" mode at adders.
   - Rebuild your filter (yes, again 5 minutes wait) and verify it again. **Note** how much were you able to improve the filter dynamic range. Notice how the filter behaves now with "saturation".

   Hint: The dynamic range is the ratio of maximum signal magnitude to the magnitude of noise. A very rough approximation would be to assume that the numerical noise will have the magnitude of a least significant bit at the output, so the larger is the allowable signal at the output, the better the dynamic range.

8. *Extra task: change the hardware to an IIR (use a copy of* `D:/LMyRio/IIR-FPGA`*, with a main project file* `CYPS_MyRio_IIRFPGA`*), design a better filter; if you want an order* > 2*, do it as a cascade of biquadratic sections.*

   There is a Matlab function to convert transversal filter coefficients to a cascade of biquads: `LCPS_casc_biquad()`

9. *Extra task: set up a "microphone plus interference" experiment (see the figure) and remove the interference with a good notch filter (similarly to previous lab on filters).* ***Be careful with sound volume – never switch things with headphones on your head, and always start with "Att -20 dB" attenuator engaged on the generator.***