Hardware Software Codesign

Individual Project Assignment

Goals

Suppose the implementation of the following embedded system, where:

- Input Analog signal is going to an Analog to Digital Converter (ADC).
- ADC samples this signal with a frequency of 100 MHz and creates 16 bits samples in the range <-1,1> represented in fixed-point format (1 bit for the integer part, 15 bits for the decimal part).
- The samples are going to Xilinx FPGA Zynq chip, where they are processed using a low pass FIR Filter.
- Output samples are going from FPGA chip to Digital to Analog Converter (DAC), where they are transformed back to an analog signal.

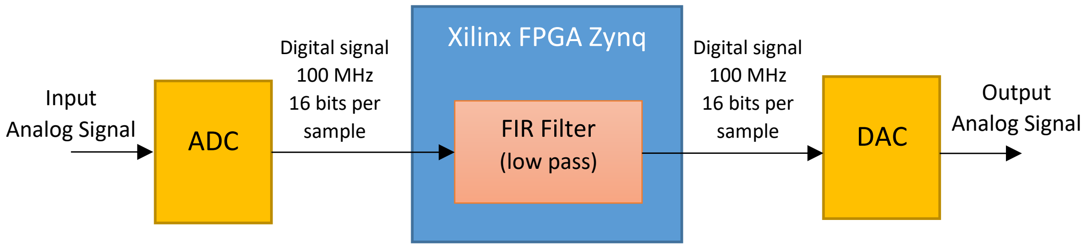

Figure 1: System architecture

The goal of this project is to design and implement Low Pass FIR Filter using Xilinx FPGA Zynq technology. Firstly, suppose that an embedded ARM processor is used for this task and incoming samples are represented using double data type. Then, modify this software implementation to run in the FPGA logic, change input sample's data type to fixed-point and implement filter using Vivado HLS tool.

Recommended Procedure

1. Familiarize with the basic information about FIR Filter design (e.g. [1]).
2. Download the *project.zip* file from the information system and extract it into your local directory.
3. Install necessary design tools (follow the instructions in section *Installation of Necessary Tools*).
4. Change the original FIR Filter source code such that it can run inside the FPGA logic and measure the difference between software and hardware implementation using the attached test bench file (follow the instructions in section *Transformation of FIR Filter into the FPGA Logic*).
5. Extend the hardware FIR Filter implementation such that it can fully utilize the available FPGA logic (follow the instructions in section *Extension of FIR Filter*)
6. Prepare required outputs (see section *Required Outputs*) and submit them into the information system no later than the specified date.

Installation of Necessary Tools

For purposes of this project you will need to install the following software tools:

1. Vivado WebPACK Design Edition – the basic set of tools for the design and implementation of circuits for Xilinx FPGA chips. It includes the High-Level Synthesis tool called Vivado HLS, which is the primary software used in this project.
2. GNU Octave – auxiliary visualization tool. We will use this tool for visualization of original and filtered signal as well as its frequency spectrum.

Vivado WebPACK Design Edition

Follow the instruction in *Vivado Installation Guide.pdf* file available in the information system.

GNU Octave tool

Procedure:

1. Download GNU Octave installation file from the following web page.
2. Run the executable installation file.
3. Finish Installation by selecting Next – Next – Finish.

Transformation of FIR Filter into the FPGA Logic

Create and Setup New Vivado HLS Project

Procedure:

1. Run Vivado HLS tool (use desktop icon or Start menu).
2. Create a New Project.
3. Select Project name and Location of the working directory.
4. Next.
5. Add input Files
    a. Add files: *fir_double.cpp*, *fir_fixed.cpp* and *fir_fixed.h*
    b. As a Top Function select: *fir_fixed (fir_fixed.cpp)*
6. Next.
7. Add test bench file *tb_fir_fixed.cpp*
8. Next.
9. Solution Configuration
    a. Clock Period – keep the default setting 10 ns. You can modify this later.
    b. Uncertainty – keep the default setting.
    c. Part setting
        i. Select **zynq** Family.
        ii. Select **xc7z007sclg225** chip with seed grade **-1**
10. Finish.

Fixed-point FIR Filter with Overflow Problems

Open the floating-point filter implementation by double-clicking on the *fir_double.cpp* file in the Vivado Source folder.

This is a design of N-tap FIR filter. It consists of two loops, the SHIFT loop which implements the tap shift register, and the MAC loop which performs the filter multiply and accumulate of taps against coefficients. Note that all the variables are declared as double precision. This implementation is intended for running in software (ARM processor).

Open the fixed-point filter implementation by double-clicking on the *fir_fixed.cpp* file in the Vivado Source folder. Note that this is almost the same source code except used datatypes. These datatypes are defined in *fir_fixed.h* file. Please open it and check that the datatypes are set as follows:

```
typedef ap_fixed<16,1> sig_t;
typedef ap_fixed<16,1> tap_t;
typedef ap_fixed<16,1> out_t;
```

This implementation is intended for FPGA logic. Double precision datatypes are not suitable here and therefore they are converted into the fixed-point format. Specifically, as the inputs and outputs we use one bit for the integer part and 15 bits for the decimal part.

Open the C++ test bench by double-clicking on the *tb_fir_fixed.cpp* file in the Vivado Source folder. Note that, this test bench generates the input signal composed of three sine waves and then it runs both the floating-point and fixed-point filters side-by-side and computes the quantization error between them. Note that both filters are set up with the same four tap impulse response that implements a low pass filter.

Run this test bench file by selecting menu item **Project** → **Run C Simulation** or by clicking the icon on the toolbar. At the end of the simulation, you will see SQNR = -1.64999 dB. This is too low value and it means that something is wrong.

Copy *analyze.m* script into the directory with simulation outputs. Usually *<selected project directory>/solution1/csim/build*. You should see *input.txt*, *output_double.txt* and *output_fixed.txt* files in this directory.

Run *analyze.m* script by double-clicking on the file or by executing GNU Octave GUI and selecting the appropriate directory and file name. After script running, you should see the output as shown in Figure 4.

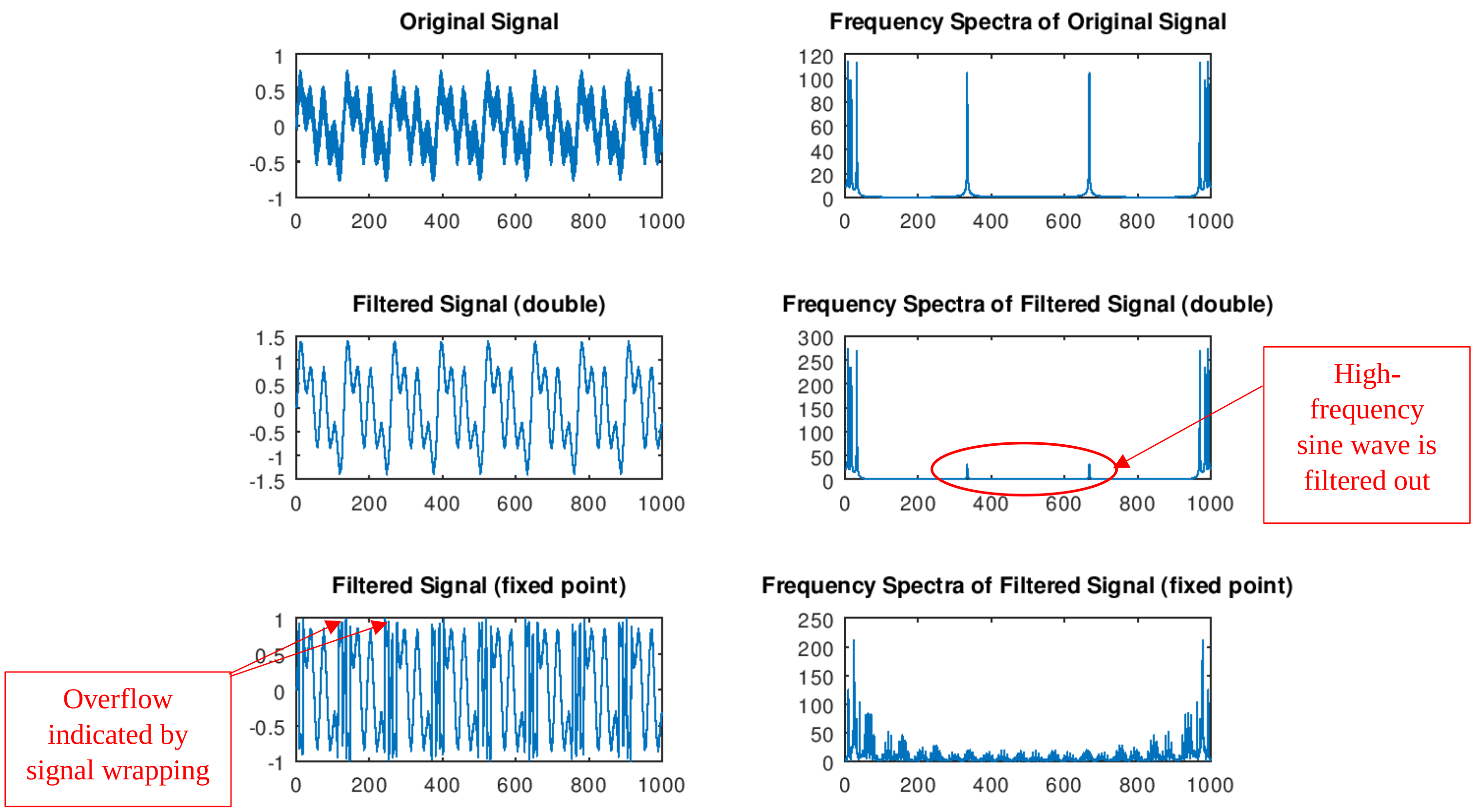

Figure 4: Signal characteristics

The Octave simulation shows that the fixed-point filter has overflow problems, and hence the output is very noisy.

Fixed-point FIR with Saturation

One way to prevent the kind of overflow seen in the previous example is to enable saturation in the *ap_fixed* data types. Although this is generally not sufficient for achieving a well-conditioned design, it does prevent the kind of catastrophic failure seen in the previous example.

Open the *fir_fixed.h* file and change datatype definition in following way:

```
typedef ap_fixed<16,1> out_t; →  typedef ap_fixed<16,1,AP_TRN,AP_SAT> out_t;
```

Run test bench file again and at the end of the simulation, you should see SQNR = 34.8107 dB. Enabling saturation has improved the signal to noise ration to about 35 dB.

Run the *analyze.m* script in GNU Octave and look at the resulting waves. You can see that the overflow effect is missing, but still, there are some non-linear deformations caused by saturation.

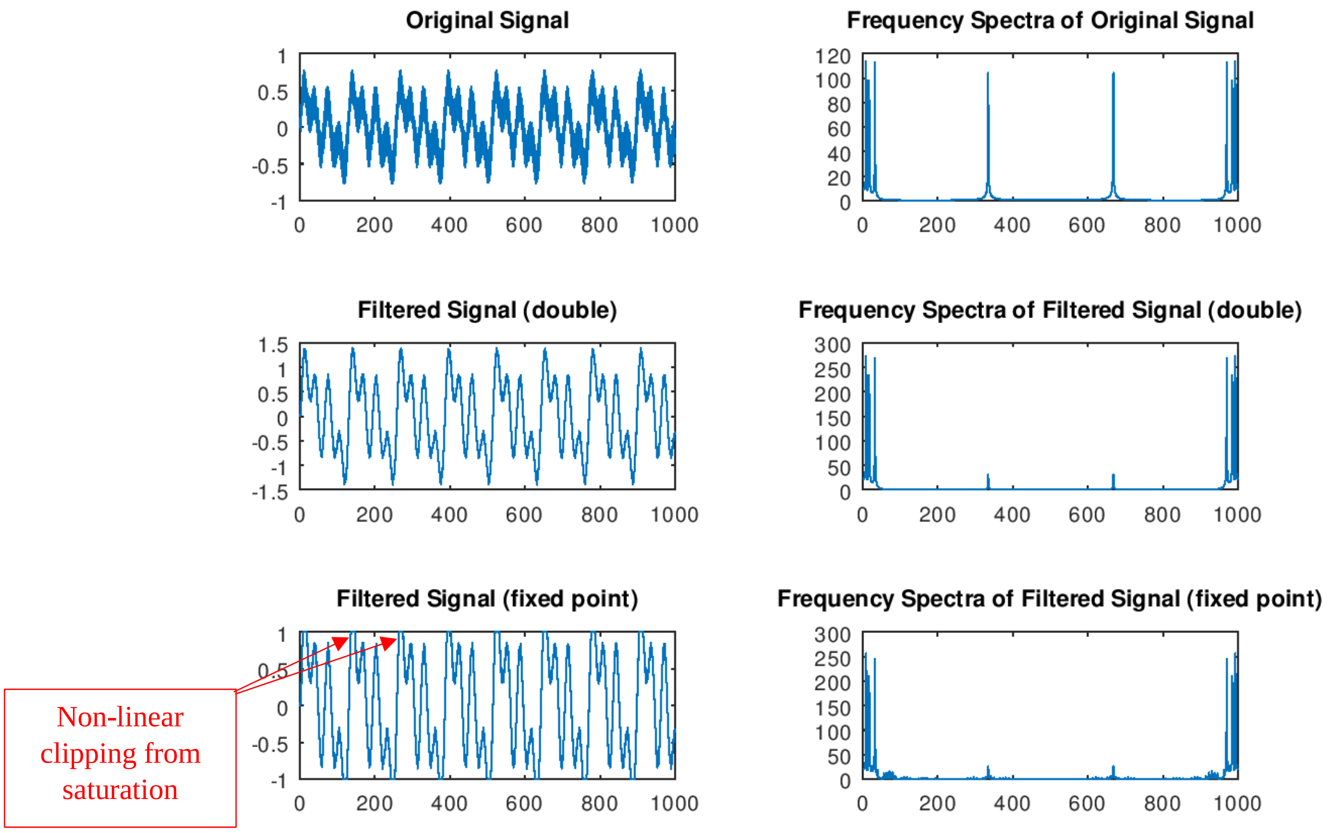

Figure 5: Signal characteristics after saturation setting

NOTE: DO NOT just turn on saturation for every fixed-point variable as this can lead to a big increase in area. Saturation should only be used in special cases. Usually, the systems or algorithm engineer should decide this.

Fixed-point FIR with Larger Internal Bit-widths

Choosing the right internal bit-widths for the fixed-point implementation is essential for achieving a well-functioning design. The choice of bit-widths is dependent on the dynamic range of the inputs as well as the internal bit growth of the algorithm.

Open the *fir_fixed.h* file and change datatype definition in the following way:

```
typedef ap_fixed<16,1,AP_TRN,AP_SAT> out_t; →  typedef ap_fixed<17,2> out_t;
```

This should have a sufficient dynamic range to prevent any overflow. Also, note that there is no saturation used for the data type.

Run test bench file again and at the end of the simulation, you should see SQNR = 161.445 dB.

Run the *analyze.m* script in GNU Octave and look at the resulting waves. You can see that there is neither an overflow effect nor non-linear deformations. SQNR = 161.445 dB represents sufficient quality.

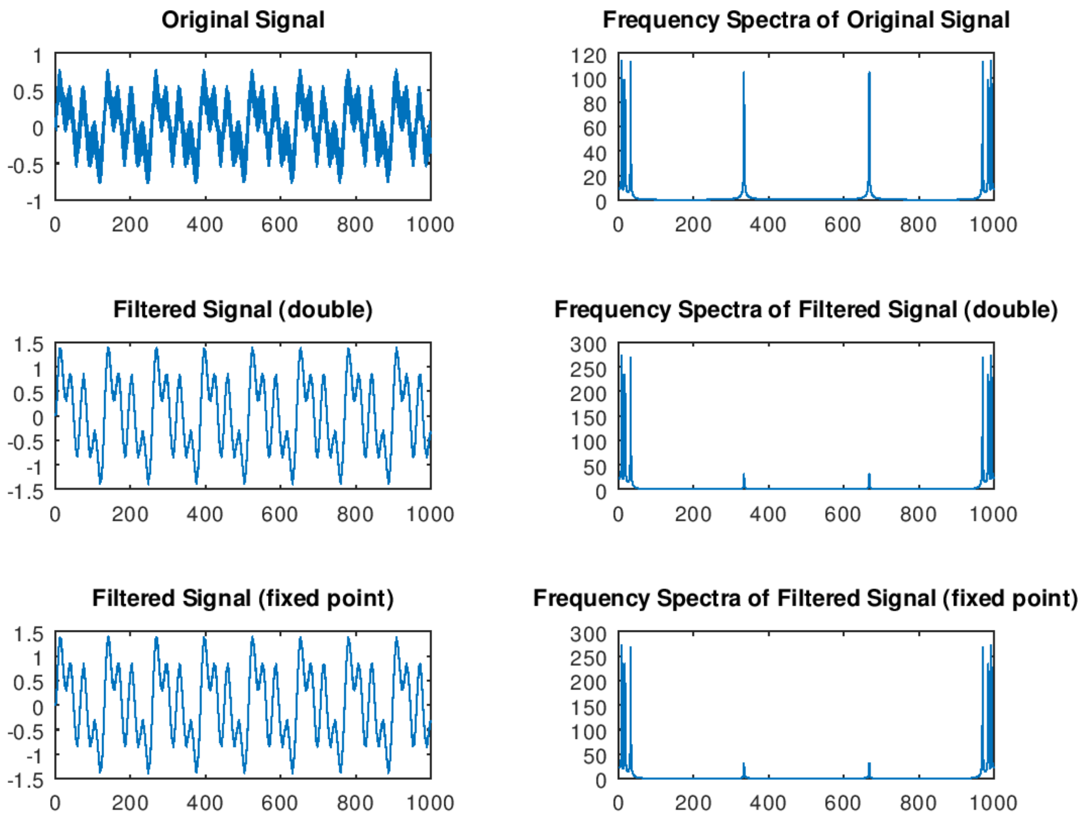

Figure 6: Signal characteristics after a bit with extension

Fixed-point FIR Implementation and Analysis

Run C synthesis by selecting menu item **Solution → Run C Synthesis** or by clicking the ▶ icon on the toolbar.

Inspect the output Synthesis Report. Focus on Performance Estimates and Utilization Estimates sections. Please note that this simple solution requires 14 clock cycles and consumes approximately 1% of the chip.

Run design analysis by selecting menu item **Solution → Open Analysis Perspective** or by clicking the Analysis icon on the toolbar. Inspect resulting Schedule, Performance profile and verify you fully understand the achieved results.

Extension of FIR Filter

The basic implementation of FIR Filter uses quite short impulse response with four taps only. Based on the theory, more taps means more stopband attenuation, less ripple, narrower filters, etc. (see Figure 7).

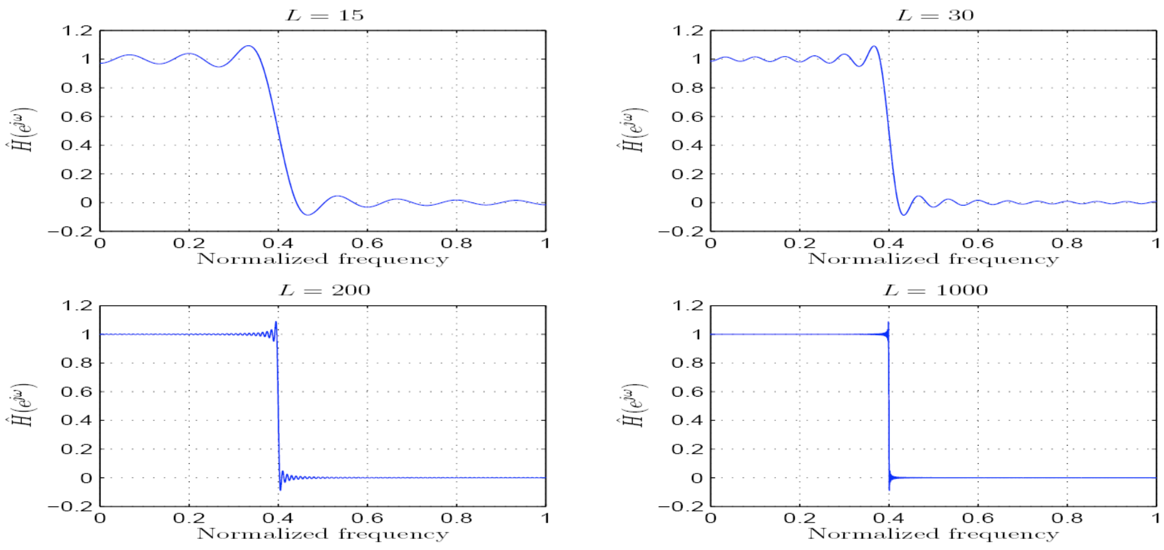

Figure 7: Impact of impulse response length on filter quality [3]

The goal of this step is to extend the basic hardware implementation of FIR Filter such that it has as many taps as possible. For this extension, you can utilize all available FPGA logic. Please note, you still have to process input signal samples coming at frequency 100 MHz.

During the extension process, please consider the following techniques:

- Extension of design frequency (you should go up to 300 MHz).
- Loop pipelining and unrolling techniques.
- Appropriate array partitioning.
- Appropriate interface setting.

A detailed description of these techniques is available in [2].

Sometimes the HLS tool has problems with automatic tree balancing. In the case of FIR filter, this relates to multiply and accumulate operations (MAC loop). If you see that the accumulate operations are not balanced into the tree structure in the final schedule, then rewrite the source code as follows:

```
MAC:for (int i = 0; i<TAPS; i++)
{
  temp += h[i]*regs[i];
}
```

```
out_t mult_temp[TAPS];

MUL:for (int i = 0; i<TAPS; i++) {
  mult_temp[i] = h[i]*regs[i];
}

ADD:for (int i = 0; i<TAPS; i++) {
  temp += mult_temp[i];
}
```

Please ensure that:

- Timing constraints are not violated. If you look at the Synthesis Report – Timing section, you should not see any red-colored values regarding the *ap_clk* signal.
- Available computation resources are not exceeded. If you look at the Synthesis Report – Utilization Estimates, you should not see any red-colored values in the list of resources.

In the output technical report please provide the following information:

1. Short description of used transformations.
2. Performance comparison of software and hardware implementation. Try to ask the following questions:
   a. How many taps could be implemented in ARM processor? For simplicity, suppose the ARM core is running at 766 MHz and single MAC operation takes 1 CPU clock cycle.
   b. How many taps, you were able to implement in FPGA logic?
   c. What kind of FPGA resource limits the further extension of the FIR Filter?

Required Outputs

As the outputs the following parts are required:

- Source codes and constraint file of hardware FIR filter (extended version) for Vivado High-Level Synthesis tool.
  - *fir_fixed.h*, *fir_fixed.cpp*
  - *directives.tcl* (located in *<selected project directory>/solutionX* directory)
  - *script.tcl* (located in *<selected project directory>/solutionX* directory)
- Technical Report named *report.pdf* describing:
  - Short description of steps performed during FIR filter extension.
  - Performance comparison of ARM-based software solution and implementation in FPGA logic (answers related to the questions at the end of section Extension of FIR Filter).
  - The technical report should not exceed one page A4 format.

Source codes and technical report pack into a zip archive named *project.zip* and upload it into the information system no later than the specified date.

References

1. Michael Barr, Introduction to Finite Impulse Response Filters for DSP, 2002, available online:
2. Xilinx, Vivado Design Suite User Guide, High-Level Synthesis, UG902, 2019, available online
3. Elena Punskaya, Design of FIR Filters, available online