M.EEC041 - Digital Systems Design

2022/2023

Laboratory project 3 - V1.2
12 Dec 2022

(due date: Dec 19th, 2022)

PROFIR - Programmable finite response 8 filter bank

Revision history

date	notes	author
Nov 22, 2022	V0.1 - Preliminary version with first spec of the filter bank	jca@fe.up.pt
Dec 2, 2022	V1.0 – Too many changes added to the initial specification	jca@fe.up.pt
Dec 6, 2022	V1.1 – Updated figure 1: corrects the connection from the ADC interface to the filter bank (this was wrongly connected to the RAM bank); added figure 2 with the timing diagram of the memory read access. New or updated text is in blue.	jca@fe.up.pt
Dec 12, 2022	V1.2 – Added section 1.1 with additional details about the implementation of the fixed-point arithmetic (new text in blue)	jca@fe.up.pt

1 - Introduction and functional description

This project will build a custom digital circuit for implementing a bank of eight different finite response digital filters for processing an ultrasonic signal sampled at 1953.1 kHz (250 MHz/128), with 16 bits per sample. Each digital FIR filter can have up to 128 coefficients represented by 18 fixed-point signed numbers, with 2 integer bits and 16 fractional bits.

For each input sample, the system must compute the outputs of the 8 digital filters, producing 8 digital signals with the same sampling frequency as the input signal. The 8 outputs must be updated exactly in the same clock cycle. The output is produced by calculating the discrete convolution between the input signal (represented by samples $x_{j)}$ and the filter impulse response h_k (the filter coefficients):

$$y_n = \sum_{k=0}^{N-1} h_k \ . x_{n-k}$$

In this expression, the values x_{n-k} represent the N previous input samples, h_k are the N filter coefficients and y_n is the output sample to calculate. The maximum number of coefficients is 128. Shorter filters can be implemented just by filling the non-existent coefficients with zeros. The behaviour of the system, for a single FIR filter can be described by the following pseudo code;

Xpast[128] contains the last 128 input samples received at the datain input
Hcoeff[128] contains the 128 filter coefficients
Datain is the input sample
Dataout is the output sample
For each new input sample arriving at **Datain** (when din_**enable** == 1)

```
   For i=0 to 126
      Xpast[ i ] = Xpast[ i+1 ]
   endFor
   Xpast[ i ] = Datain

   Accum = 0
   For i=0 to 127
      Accum = Accum + Xpast[ i ] * Hcoeff[ i ]
   endFor

   Dataout = Accum
```

The filter coefficients h_k are read from a bank of 8 independent RAM memories. Each memory is organized as 64 words of 36 bits (two 18-bit coefficients per memory location) and is read by a clocked synchronous interface, as described below. The memories will be implemented as dual-port memory blocks, using one write port to upload the memories with the filter coefficients (this part is not included in your project). The low-order filter coefficient will be stored in the 18 least significant bits of each word, as represented below:

RAM address	RAMk[35:18]	RAMk[17:0]
0	h_1	h_0
1	h_3	h_2
2	h_5	h_4
…	**…**	**…**
63	h_{127}	h_{126}

Figure 1 shows a simplified block diagram of the system to design.

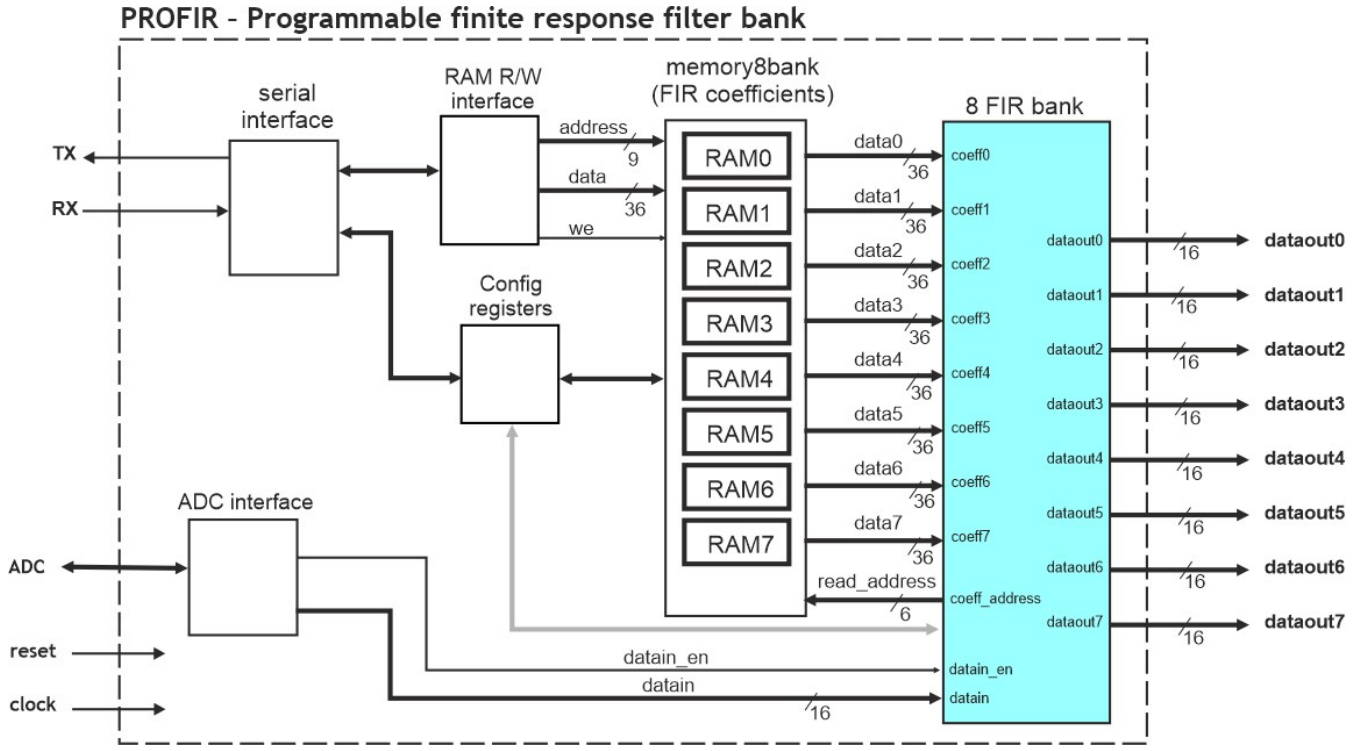

Figure 1 – Block diagram of the top-level system. The filter bank is represented by the blue box.

Your design will implement only the 8 digital filters (the blue box) and will be integrated in a complete design that includes the memories holding the filter coefficients, the interface to upload those memories from an external system and the rest of the digital processing system. The block labeled “Config registers” implements a set of registers for general configuration of the system. The gray connection from this module to the filter bank is not implemented in the current testbench and can be ignored for now.

1.1 - About the fixed point arithmetic

The filter coefficients are signed values represented by 18-bit words. If we "read" them as representing a fixed-point value with 16 fractional bits, their values H are equal to the 18-bit integer word Hi divided by 2^16.

For example, the value H=1.23456 can be represented as a 18-bit fixed-point number with 16 fractional bits Hi by multiplying H by 2^16 and taking only the integer part of that result: 1.23456 x 2^16 = 80908.12416000001 and discarding the fractional part we have the integer number Hi=80908 (or in binary 01_0011_1100_0000_1100) which represents the value H’ = 80908 / 2^16 = 1.234558105… as an approximation of the original real number H.

If these fixed-point values are multiplied by a 16-bit integer X, the result will have 18+16 bits and the value represented by the result is equal to H’ x X = (Hi / 2^16) * X = (Hi x X) / 2^ 16. Thus, of the 34 bits of this product 16 represent the fractional part of H’ x X. When adding 128 of these products (the result of the inner product H[ ] * X[ ]) it is necessary to add more 8 bits to represent this result (this is equivalent to multiplying by 128 which is represented by 8 bits). Thus, the final summation result will have 42 bits, with 16 bits representing the fractional part of the result and the remaining 26 bits representing the integer part.

If the FIR filter coefficients are such that the digital filter has a maximum gain not greater than 1 (or ≤ 0 dB), then the result should "fit" into the least significant 16 bits of those 26 output bits. However, if the filter gain exceeds 0 dB it will be necessary to apply a scale factor so that the result is scaled down to fit into 16 bits. This can be done by dividing the 26-bit result by a factor to be calculated based on the filter response or apply a scale factor equal to 2^K (remember that dividing X by 2^K is equivalent to shifting the bits of X to the right by K positions.

The examples of LP/BP/HP filters implemented in the Matlab script genfilterdata.m have a nominal 0 dB gain in the passband, but with the "ripple" of the frequency response there are some frequency values for which the gain passes slightly above 0 dB. This means that, if a full-scale signal of that frequency is applied to the filter, 16 bits will not be enough to represent the output signal. There are some simple ways to get around this problem, but to simplify the implementation we can admit that the programmed filters have to guarantee a gain below 0dB for all the frequency range, which implies that the final result necessarily will fit in the 16 integer bits. Note that if that assumption is not true and the filter coefficients can be any value in the possible range [-2.0000000, + 1.9999847], then the final integer result will really have to be represented in 26 bits and then the easy solution to reduce it to 16 output bits is simply dividing it by 2^10 = 1024.

2 - Implementation

The system must be designed as a single clock domain synchronous digital circuit, clocked at 250 MHz. The Verilog code below represents the interface of the system to design:

```
module profir(
        input                   clock,          // Master 250 MHz clock, active in the rising edge
        input                   reset,          // Master reset, synchronous, active high
        input signed  [15:0]    datain,         // Input signal sample
        input                   din_enable,     // When 1, a new sample is present at datain input (lasts 1 clock)
        output         [5:0]    coeffaddress,   // Address to read all the coefficient memories
        input signed  [35:0]    coeff0,         // Coefficient read data for filter 0
        input signed  [35:0]    coeff1,         // Coefficient read data for filter 1
        input signed  [35:0]    coeff2,         // Coefficient read data for filter 2
        input signed  [35:0]    coeff3,         // Coefficient read data for filter 3
        input signed  [35:0]    coeff4,         // Coefficient read data for filter 4
        input signed  [35:0]    coeff5,         // Coefficient read data for filter 5
        input signed  [35:0]    coeff6,         // Coefficient read data for filter 6
        input signed  [35:0]    coeff7,         // Coefficient read data for filter 7
        output signed [15:0]    dataout0,       // Output data of filter 0
        output signed [15:0]    dataout1,       // Output data of filter 1
        output signed [15:0]    dataout2,       // Output data of filter 2
        output signed [15:0]    dataout3,       // Output data of filter 3
        output signed [15:0]    dataout4,       // Output data of filter 4
        output signed [15:0]    dataout5,       // Output data of filter 5
        output signed [15:0]    dataout6,       // Output data of filter 6
        output signed [15:0]    dataout7        // Output data of filter 7
        );
```

2.1- System design

The 8 RAMs holding the filter coefficients can be read in parallel with a single address that is connected to the address buses of all the memories. The read ports of these memories are implemented by the following Verilog process (see the Verilog file **./src/verilog-rtl/memory8bank.v**):

```
always @(posedge clock)
  coeff <= RAMCOEF[ coeffaddress ];
```

The timing of the read operation is as follows: the coefficient memory address **coeffaddress** is set by your circuit in clock transition K, the memory output data register **coeff** is loaded in clock transition K+1 and your system can use that value in clock K+2. As the memory read access is pipelined, a different memory address can be read at each clock cycle. This is illustrated by the timing diagram shown in figure 2 where signal **Y** represents a register loaded with a result computed with the RAM output **coeff**.

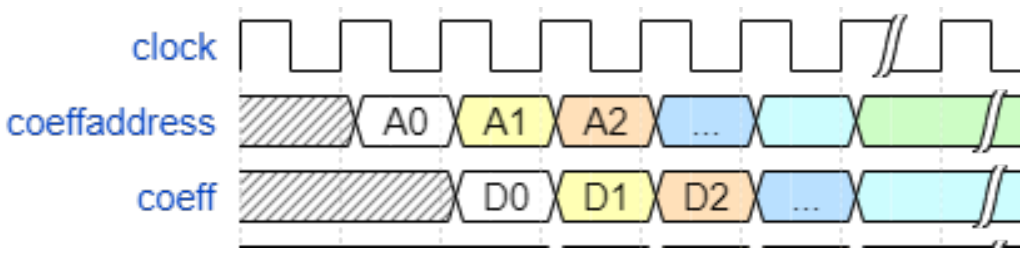

Figure 2 – Timing diagram illustrating the synchronous memory read operation

The memory bank (module **memory8bank,** Verilog source **./src/verilog-rtl/memory8bank.v**) implements the 8 RAM memories with a common read address and eight output 36-bit data buses. The write port will not be used by your design and will be connected in the top-level design to a low speed interface for loading these memories from an external computer. The memories are pre-loaded for simulation and synthesis with the data read from hex data files using the Verilog system task **$readmemh( )**. These data files files are located in folder **./simdata** and contain the filter coefficients generated by the Matlab script **./Matlab/genfilterdata**.

The calculation of all the 8 output samples must be completed before the arrival a new input sample. Thus, the computation of the 8 digital filters must be completed within 128 clock cycles (this is equivalent to a computational complexity of 4 giga integer operations per second). Figure 3 shows a timing diagram illustrating the arrival of the input samples and their relationship with the cock and the **din_enable** signal.

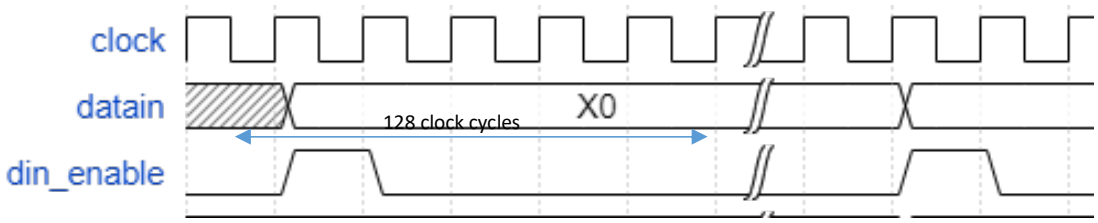

Figure 3 – Timing diagram of the input/output signals. The output signal is represented by Y0 and it must be updated before the arrival of the next sample.

2.2 – Simulation environment

The project kit includes a basic testbench for simulating the module **profir** (the 8 filter bank) and a simulation project already created for QuestaSim. The project kit includes the following folders and files. Note that it is mandatory to keep the projects and files in the correct folders as the Matlab scripts and Verilog files use relative references to some filenames

./src/verilog-rtl	Verilog source files for design implementation
./src/verilog-tb	Verilog source files for design verification/simulation
./src/data	Additional data files required for FPGA implementation
./simdata	Data files used for pre-loading the coefficient memories and for simulation
./Matlab	Matlab scripts for generating the data files
./sim	Simulation project for QuestaSim. Includes a QuestaSim script for configuring the waveform viewer (file **./sim/wave_profir.do**)

The Verilog source files provided are the following:

./src/verilog-rtl/filterbank.v	The header of the Verilog module to develop. Do not change the signal names and sizes. If needed you can add additional inputs/outputs.
./src/verilog-rtl/memory8bank.v	A synthesisable model of the memory bank with the 8 RAM holding the filter coefficients. These memories are pre-loaded for simulation and synthesis with the data files located in folder **./simdata**.
./src/verilog-tb/profir_tb.v	A basic testbench for simulating the filter bank. The testbench instantiates the memory bank and the filter bank and applies the input samples to the filter module with the given sampling frequency.

The Matlab scripts are:

./Matlab/genfilterdata.m	creates the data file **./simdata/coefficients0.hex** with the 128 coefficients implementing a digital FIR filter. This script can be easily adapted to generate different filters, one for each of the 8 channels. For now, the 8 coefficient data files implement the same filter but you should validate your system with 8 different filters.
./Matlab/geninputdata.m	creates the data file **./simdata/datain.hex** with the input signal to be applied to used in the simulation of the testbench provided. This script creates a sum of a few sinusoidal waves, with added white noise and modulated in amplitude with another slow sine wave. Feel free to modify the parameters and the sinal used.
./Matlab/genoutputdata.m	creates the data file **./simdata/goldendataout.hex** with the output signal created by the convolution between the input signal in file **./simdata/datain.hex** and the FIR filter defined by the coefficients in file **./simdata/coefficients0.hex**. created by the script **genfilterdata.m**. The testbench provided loads this file to the vector **goldenoutput[ ]** and this data should be used for implementing an automatic verification process.

2.3 – Design evaluation and grading

The design must target the Spartan6 FPGA and minimize the global logic complexity, measured as the number of main FPGA resources: lookup-tables, flip-flops and DSP48 blocks.

The project stages to complete and the grading to assign to each stage are:

1. Design of the Verilog HDL code and functional verification with logic simulation [20%]
2. RTL Synthesis and post-synthesis validation with logic simulation [20%]
3. Integration in a FPGA project (a complete project will be provided) [15%]
4. Final RTL synthesis, design optimization and validation post-synthesis [20%]
5. Place and route and post-P&R simulation [10%]
6. Final report [15%]