Politecnico di Torino

III Facoltà di Ingegneria

Laboratory 2
Digital arithmetic

Master degree in Electrical Engineering

Authors: Group 21

Nicola Dilillo, Stefano Moncalvo, Lorenzo Carrano

November 20, 2021

Many thanks to Prof. Mariagrazia Graziano for providing us with this template.

Contents

CHAPTER 1

Reference model development

1.1 Introduction

The goal of this laboratory is to design a Finite Impulse Filter filter (FIR) with a cut frequency of 2 kHz, and then applying some optimization techniques such as unfolding and pipelining to the basic structure. Filter has is design according two parameter: order and number of bits. The order employ for the following filter is 10 and the number of bits are 9.

A prototype version of the filter has been developed in C language and Matlab in order to be able to compare them with the results coming from the simulation of the HDL design.

In the following main filter specifications are summarized:

Filter Specification	Value
Filter Type	FIR filter
Cut-off frequency	2kHz
Sampling frequency	10kHz
Filter order	10
Number of bits	9

1.2 Design the filter with Matlab

First step is the generation of coefficients. To do this Matlab function fir1 has been used. The coefficients are shown in table 1.1.

Those coefficients are subjected to a quantization operation over Nb bits and expressed both in integer and real form It is possible to esimate the effect of this quantization by comparing the designed transfer function to the quantized one, as shown in figure 1.1.

At this point, another Matlab script is executed in order to perform different simulatios with prototype filter with a cut-off frequency of 2 kHz and a sampling frequency of 10 kHz, taking as input signal the average value between two sinusoidal waves of frequency of 500 HZ and 4.5 kHz respectively. After this execution two files have been generated:

1. *sample.txt*, which contains the sample values that have fed the input of FIR;
2. *result.txt*, which contains the output values that has been elaborated from our FIR.

Number	Quantize	Normalize
0	-1	1
1	-2	1
2	-4	1
3	8	0
4	35	1
5	50	1
6	35	1
7	8	1
8	-4	1
9	-2	1
10	-1	1

Table 1.1: All coefficients.

Figure 1.1: Designed Transfer Function vs Quantized Transfer Function

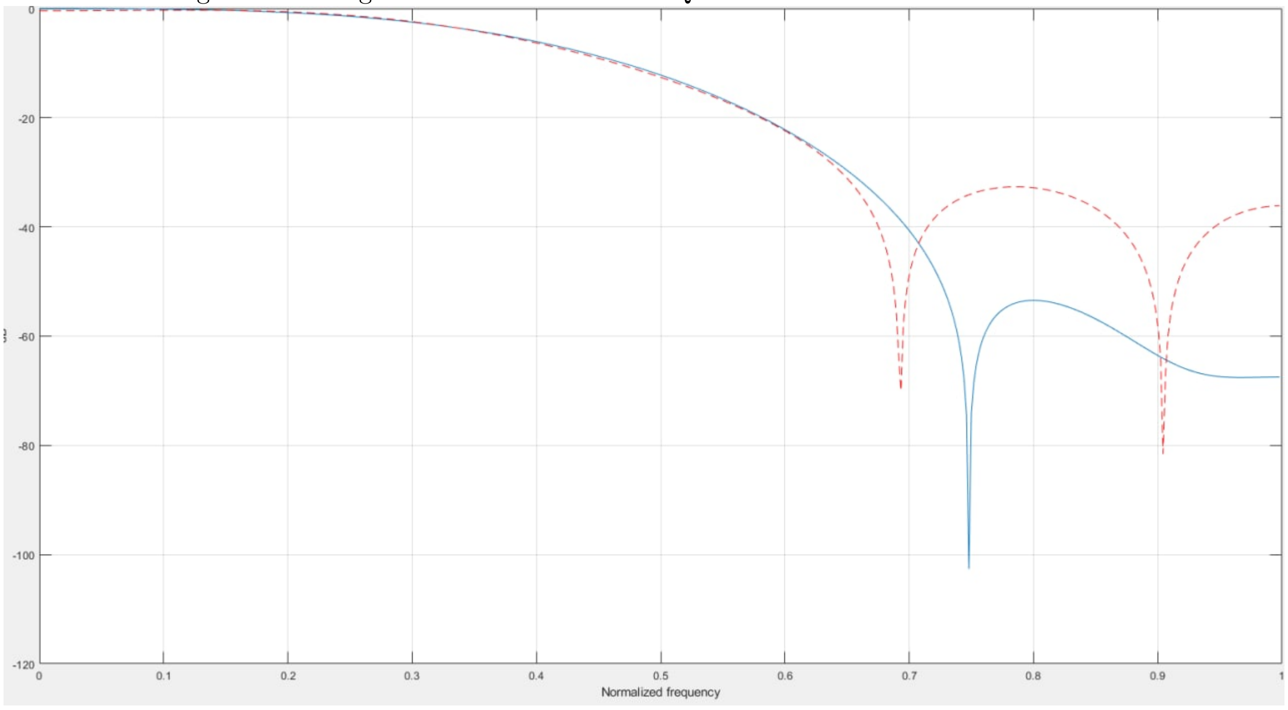


Figure 1.2 shows the two sinusoidal waveforms and the effective filter's input.

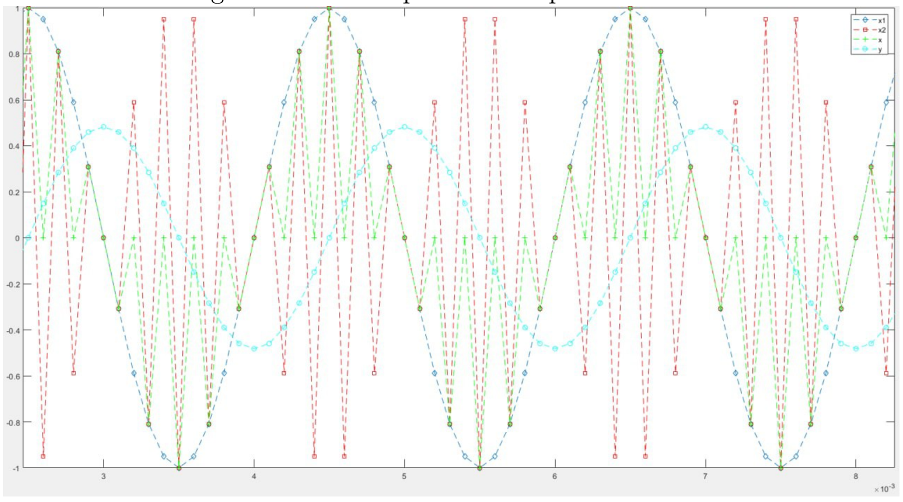
1.3 C prototype

The C-language script simualtes a FIR filter implementing the following relation:

$$[!h]y_i = \sum_{n=0}^{10} x_{i-n} \cdot b_n$$

Inside the script, a function is defined in order to evaluate the output y[n] at a specific time instant, knowing the input sample x[n].

Figure 1.2: FIR Input and Output overt time



FIR constants are hardcoded inside the script, declared as a constant array of integers, while an internal buffer is needed to store the previous input values and shift them for each function call, and the output is obtained by summing coefficient-input products in a loop.

In order to emulate the finite internal parallelism of the hardware architecture, a shift operation is performed on the result before storing it into the accumulator. Of course, this truncation operation introduces an error in the evaluation of the output samples. The maximum accurancy would be obtained with a parallelism that is equale to the double of the bits number used for the architecture.

Thanks to this script is possible to compare the performance of the fixed-point version with respect to the Matlab computation results.

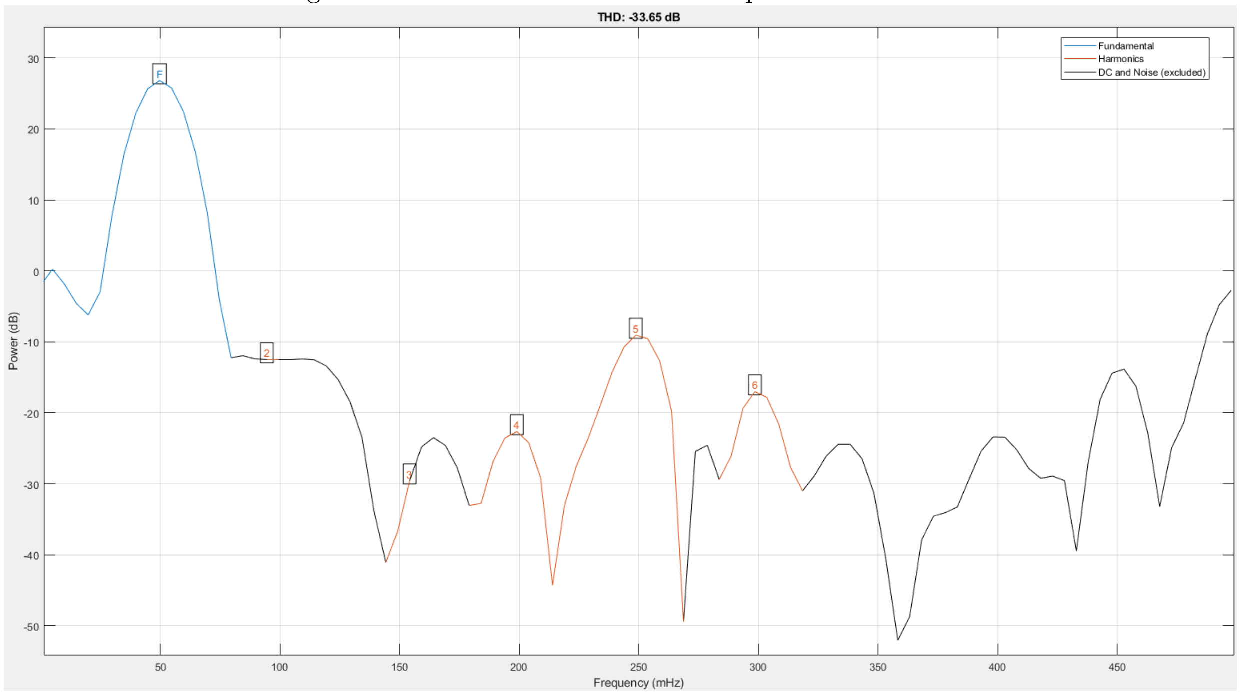
1.3.1 Evaluate the THD

The purpose of this step is to evaluate the Total Harmonic Distortion (THD), trying to obtain a maximum value of -30dB. If THD exceeds the maximum tolerated value, it is necessary to increase the bit numbers in order to reduce its amount, while if there is a gap between the maximum tolerated value and the obtained one, it is possible to reduce bit numbers and thus the complexity of the FIR implementation.

With 9 bits used for data, obtained THD is -40 dB. Reducing the number of bits to 8 the obtained value of THD is -33 dB, thus still acceptable. Applying a further bit-number reduction, thus with a 7-bit parallelism, the value of THD exceeds the maximum allowed amount, reaching the value of -27dB.

At the end of this analysis, it has been decided to use 8 bits for the final implementation of the FIR, in order to reduce the area while still accomplish maximum THD amount, shown in figure 1.3.

Figure 1.3: Final THD for FIR filter implementation



CHAPTER 2

VLSI implementation

2.1 Starting architecture development

The purpose of this section is to develop in VHDL the architecture of the previously designed filter. The architecture of the filter is composed by four elements:

- Adders
- Multipliers
- Flipflops
- Registers

The 8-bit input is recived and then propagated through a chain of 10 registers; the output of each register is multiplied by the corresponding coefficient, and the results are summed together to form the filter's output. All registers use VIN as an enable signal, in order to avoid unwanted propagation of data. The VIN signal, delayed of two clock cycles, is also used to drive VOUT. Every input and output signal is loaded or produced by registers or flipflops, to reduce the risk of interference from external signals.

2.2 Simulation

The design was simulated using a testbench written in both Verilog and VHDL. The testbench is composed of four disinct entities:

- **clk_gen:** generates a clock signal of the specified frequency, and a reset signal.
- **data_maker:** reads the samples.txt file and provides an input every clock cycle and its validity using the VIN signal.
- **data_sink:** recives the outputs of the filter every clock cycle and writes them in the output.txt file if VOUT is equal to 1.
- **tb_fir:** is the testbench top entity written in Verilog.

At the end of the simulation the values stored in Output.txt were compared with the ones produced by the C prototype. The two files are equal, which means that the filter is behaving correctly.

2.3 Logic synthesis

After the simulation the design must be synthetized. To estimate the maximum working clock frequency of the filter, the clock period in the design compiler is set to 0 ns. In this way the compiler optimizes the circuit as much as possible, and the negative slack of the timing report corresponds to the maximum clock frequency.

After running the synthesis at the computed frequency the area is evaluated.

It is requested to set the frequency to 25% of the maximum value. After the synthesis a new area estimation is produced.

The constraints on the clock have a significant role in the estimation of the area: by allowing the frequency to be lower, the size of the circuit will be smaller.

The Design Compiler produces the Verilog netlist of the synthetized circuit and a .sdf file containing the circuit's delays. Those files are used by Modelsim to simulate the netlist with the correct timing parameters and obtain the switching activity of the nodes, which is saved in a .vcd file. The results of this simulation have been checked to assure that they are coherent with the ones of the original circuit.

This file is converted to .saif and used by Design Compiler to generete a power report.

2.4 Place & Route

The last section requires to perform the place and route on the synthetized circuit to obtain the switching activity and power report. To do that using Innovus several steps are necessary:

- **Structuring the floorplan**, where Innovus allocates the area for the cells;
- **Inserting power rings**, two rings for power (VDD) and ground (VSS) are inserted around the floorplan;
- **Standard cell power routing**, horizzontal wires for power and ground are prepared for the cells;
- **Placement**, the cells are placed in the floorplan but are still to be connected between them;
- **Post Clock-Tree-Synthesis optimization**, the design is optimized to achieve the required timing constraints;
- **Place filler**, filler cells are placed to ensure continuity in n+ and p+ wells in each row;
- **Routing**, the cells are connected among each other;
- **Post routing optimization**, the design is optimized again to achieve the timing constraints. After this step, the design is saved as a .enc file;
- **Parasitics extraction**, Innovus extracts the parasitic values of resistencies and capacitances;
- **Timing analysis**, the performance of the circuit is evaluated, if the slack is negative the constraints are violated;
- **Design analysis and verification**, Innovus checks for the presence of floating wires and violations on the constraints on the geometric features of the circuit. Finally the area and gate count, the netlist and a file with delay annotations are saved.

Since the slack values for setup and hold are positive and the verification on connectivity and geometry returned no errors, the final step is to simulate the produced netlist with Modelsim, to check if the circuit behaves correctly and to calculate the switching activity. The following results have been obtained:

CHAPTER 3

Advanced architecture development

In this section the purpose is to improve the FIR performance. In first and unfolding technique has been used to improve the throughput and then pipeline technique has been used to reduce the longest path and improve maximum frequency.

3.1 Unfolding

Unfolding of order 3 has been applied to FIR filter (N = 3) and the equations derived to build the new system are the following:

$$\begin{aligned} y[3n] = a_0 \cdot x[3n] + a_1 \cdot x[3(n-1)+2] + a_2 \cdot x[3(n-1)+1] + \\ a_3 \cdot x[3(n-1)] + a_4 \cdot x[3(n-2)+2] + a_5 \cdot x[3(n-2)+1] + a_6 \cdot x[3(n-2)] + \\ a_7 \cdot x[3(n-3)+2] + a_8 \cdot x[3(n-3)+1] + a_9 \cdot x[3(n-3)] + a_{10} \cdot x[3(n-4)+2] \end{aligned} \tag{3.1}$$

$$\begin{aligned} y[3n+1] = a_0 \cdot x[3n+1] + a_1 \cdot x[3n] + a_2 \cdot x[3(n-1)+2] + \\ a_3 \cdot x[3(n-1)+1] + a_4 \cdot x[3(n-1)] + a_5 \cdot x[3(n-2)+2] + a_6 \cdot x[3(n-2)+1] + \\ a_7 \cdot x[3(n-2)] + a_8 \cdot x[3(n-3)+2] + a_9 \cdot x[3(n-3)+1] + a_{10} \cdot x[3(n-3)] \end{aligned} \tag{3.2}$$

$$\begin{aligned} y[3n+2] = a_0 \cdot x[3n+2] + a_1 \cdot x[3n+1] + a_2 \cdot x[3n] + a_3 \cdot x[3(n-1)+2] + \\ a_4 \cdot x[3(n-1)+1] + a_5 \cdot x[3(n-1)] + a_6 \cdot x[3(n-2)+2] + a_7 \cdot x[3(n-3)+1] + \\ a_8 \cdot x[3(n-2)] + a_9 \cdot x[3(n-3)+2] + a_[10] \cdot x[3(n-3)+1] \end{aligned} \tag{3.3}$$

Using this method of optimization the two more input and output port have been added becouse now 3 inputs are processed and produce, at the same time, 3 output. To overall throughput has been triplicate.

3.2 Pipeline

A further optimization has been applied. This method allows to reduce the size of critical path.

From schematic of previous FIR is possible to see that the critical path is the chain off adders. To achieve the best result is not necessary split each of them, but it is needed to group them in order to have a total time path that is less than the amount of time for performed a multiplication.