MSc in Computer Engineering

Electronics and Communication Systems

UNIVERSITÀ DI PISA

IIR FILTER

Gabriele Pianigiani

Summary

Introduction

Infinite impulse response (**IIR**) is a property applying to many linear time-invariant systems that are distinguished by having an impulse response which does not become exactly zero past a certain point but continues indefinitely.

A digital IIR filter is often described and implemented in terms of a difference equation that defines how the output signal is related to the input signal.

IIR filters are used in various applications, such as IoT smart sensors, biomedical signal processing, audio applications and so on.

The designed IIR filter is specific for audio applications, so it could be used to adjust the frequency response of systems such as amplifiers and speakers, to reduce the background noise of a microphone, to reduce the size of audio files etc. The designed IIR filter must implement the following difference equation:

$$y[n] = y[n-1] - \frac{1}{4}x[n] + \frac{1}{4}x[n-4]$$

Equation 1 - IIR filter difference equation.

Requirements

The logical view of the circuit is represented in the following image.

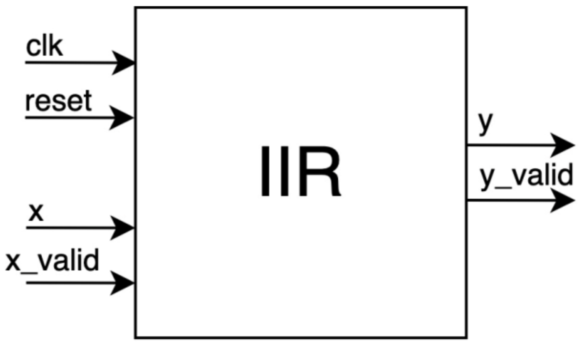


Figure 2 - IIR filter logical view.

The **clock** and the **reset** are composed by 1 bit each, the input **x** and output **y** are composed by 16 bits. New data is evaluated only when the 1 bit **x_valid** signal is asserted (= 1). Filtered samples are delivered in output for one cycle, setting the 1 bit **y_valid** signal to 1.

Possible Solutions

One of the possible solutions can be to implement the difference equation with a sequential process, because it needs to perform simple arithmetic operations taking into account also the management of the possible overflows, since with that formula there are some values that can generate either positive or negative overflows.

Theoretical Model

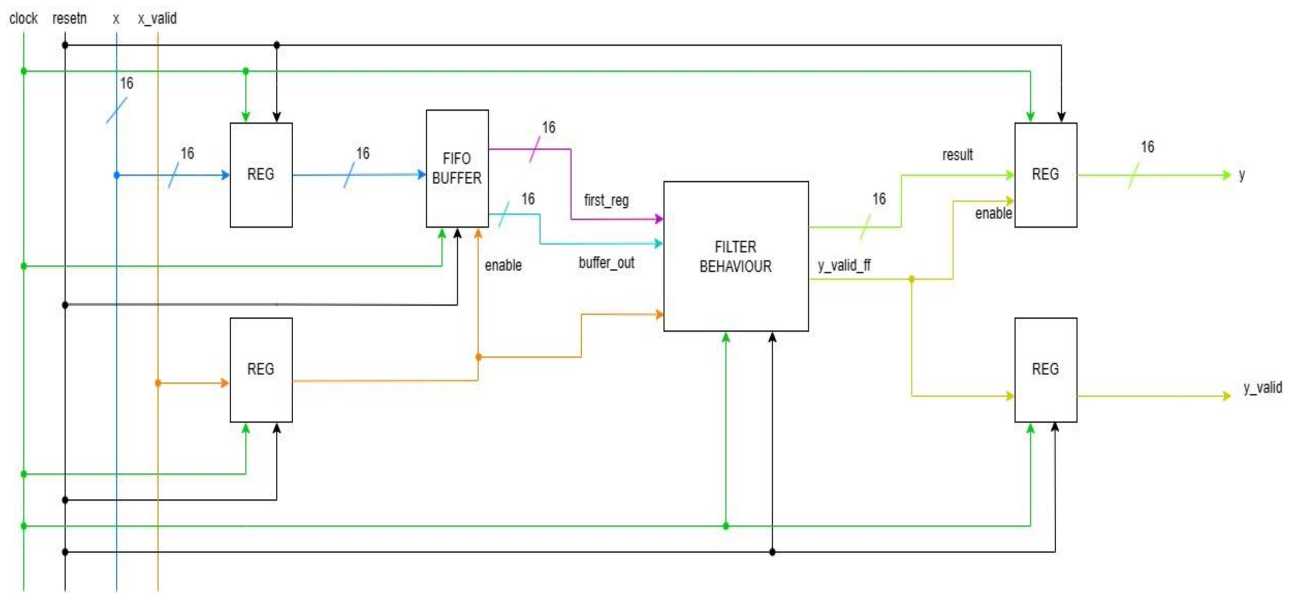


Figure 2 – Theoretical design of the circuit.

Inside the *filter behaviour* block there is the implementation of the two validity checks, the one to verify if the buffer is full and the one to check the overflow. There is also the implementation of the difference equation.

VHDL Code

Directory

The VHDL source code has the following structure:

- **d_flip_flop.vhd**
    - this component implements a d flip flop of 1 bit.
- **d_flip_flop_n.vhd**
    - this component implements a d flip flop of n (=16) bits.
- **buffer_fifo.vhd**

- this component implements a FIFO buffer to memorize the current input and the previous four inputs.

- **iir_filter.vhd**
  - this is the top-level component, the IIR filter.

Overall Schematic

The overall schematic can be seen in the *iir_filter_schematic.jpg* file, since it is too big, it has not been inserted in the documentation, but it is in a separate file.

The schematics of main components of the filter are shown in the figures below.

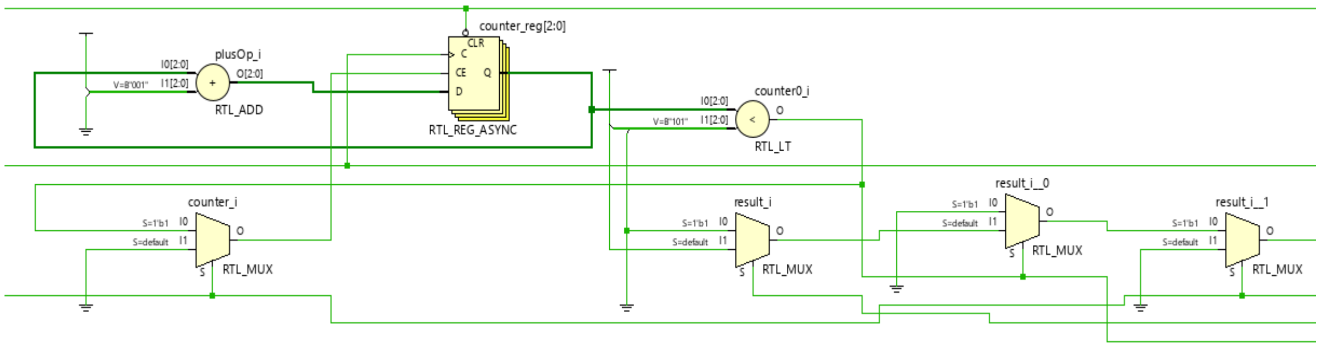


Figure 3 – Full buffer checker.

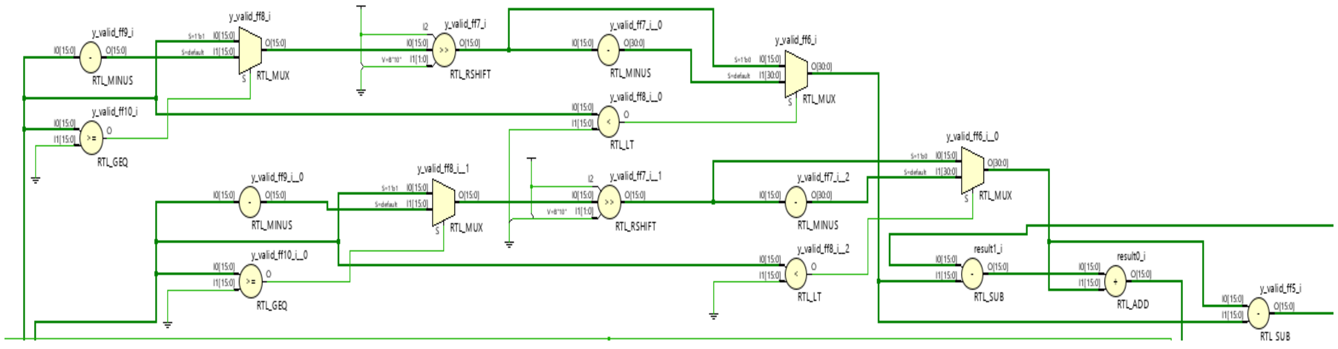


Figure 4 – Implementation of difference equation.

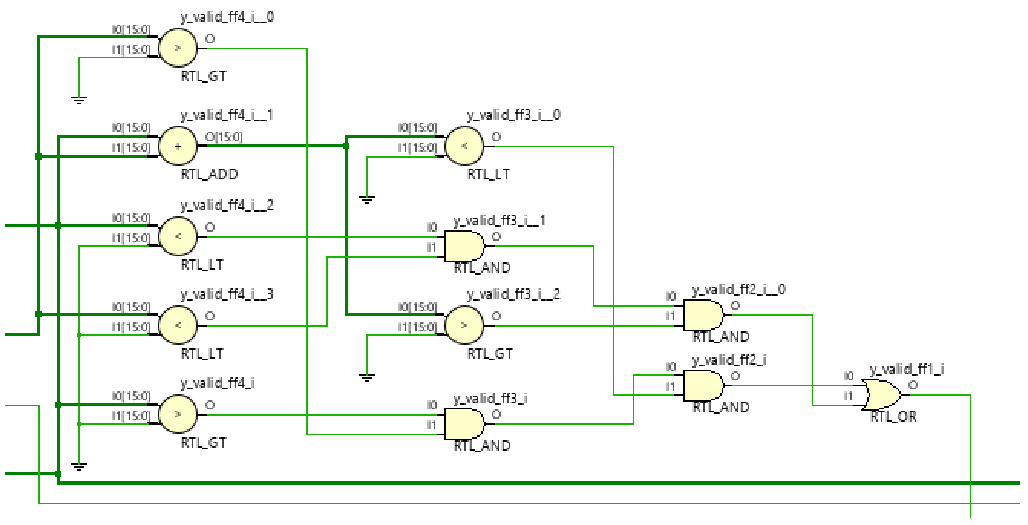


Figure 5 – Overflow checker.

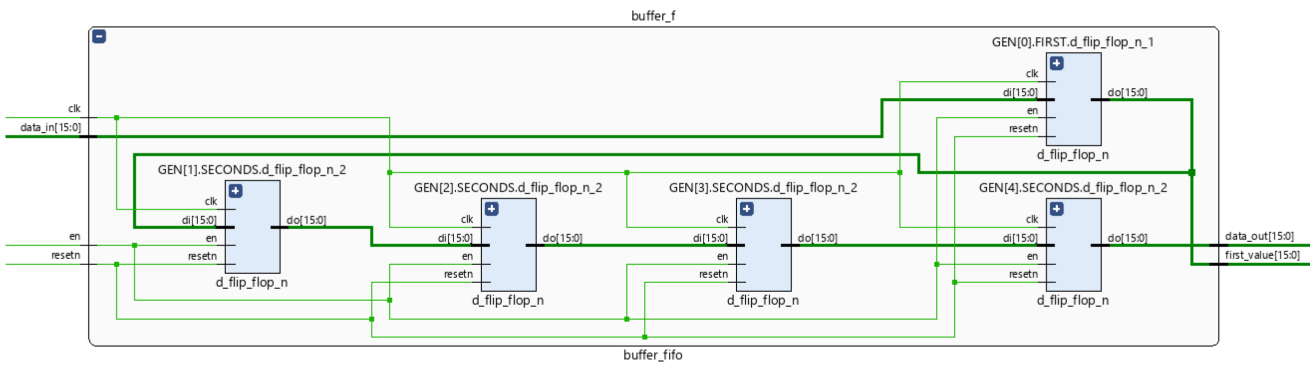

buffer_f
GEN[0].FIRST.d_flip_flop_n_1
GEN[1].SECONDS.d_flip_flop_n_2
GEN[2].SECONDS.d_flip_flop_n_2
GEN[3].SECONDS.d_flip_flop_n_2
GEN[4].SECONDS.d_flip_flop_n_2
d_flip_flop_n
clk
data_in[15:0]
en
resetn
di[15:0]
do[15:0]
data_out[15:0]
first_value[15:0]
buffer_fifo

Figure 6 – Fifo buffer schematic.

Testbench and ModelSim

The testbench created to test the implementation is composed in 5 parts:

Reset test

In this section the reset is tested. The reset signal is assigned to 0 and then to 1 again (*observing the image below, the reset goes down for two clock cycles*).

Output validity test

In this section is tested the validity of the output of the filter while the buffer FIFO is not full yet. In details it has been tested that the y_valid output stays equal to 0 until all the five registers of the buffer are not full (full means that all the five registers contain a value).

Positive overflow test

In this section the positive overflow is tested, the filter receives some particular inputs in order to generate an overflow, when the overflow is generated the y_valid output turns to 0 and the y output keeps the last valid output. It keeps it because it is necessary in the different equation, the invalid output must not be used to compute a new result.

Negative overflow test

In this section the negative overflow is tested, the filter receives some particular inputs in order to generate an overflow, when the overflow is generated the y_valid output turns to 0 and the y output keeps the last valid output for the same reason as the positive case.

Normal cases test

In this section the design is tested with some normal values, even testing the behaviour of the filter when a non-valid input is assigned, in this case, the output becomes non-valid in the following clock cycle and the y output keeps the last valid output for the same reason of the overflow cases.

Waveform

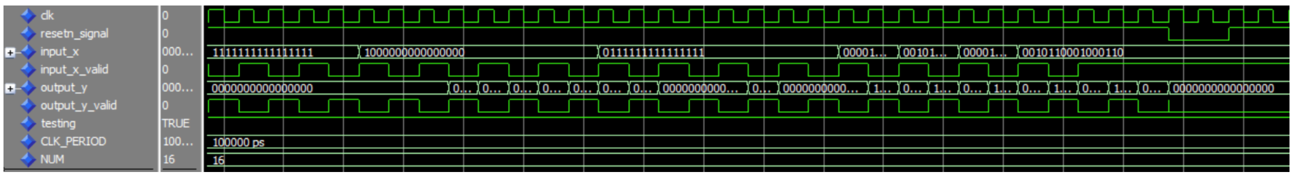


Figure 7 - Example of the waveform (simulation with registers).

Theoretical verification

A small theoretical verification has been performed to verify the results of the filter's equation during normal cases, the results obtained by the verification are the same that are shown during the simulation with ModelSim. These results can be found in the file called "*verification.xlsm*".

Vivado

To use Vivado, a new project called *iir_filter has been created* in the Vivado folder.

Open elaborated phase

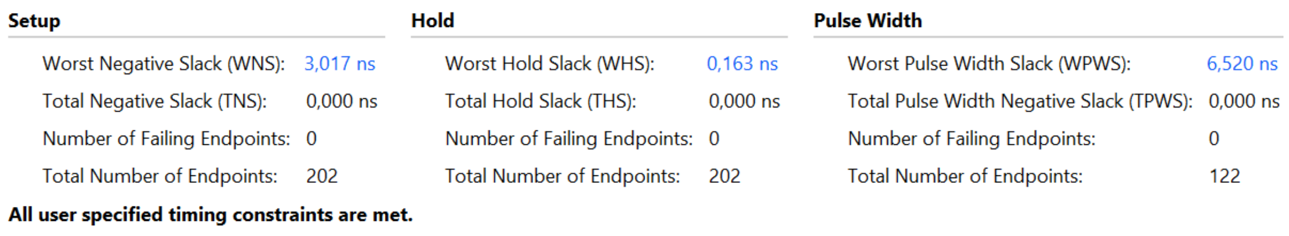
The same board used during lectures (xc7z010clg400-1) has been chosen and the initially chosen clock period is of 15 nanoseconds.

Synthesis phase

The synthesis phase was completed without any error or warning. The result of the timing report was that the timing constraints were largely met, so after this, the next step is the implementation phase.

Implementation phase

The implementation phase was completed without any error or warning. The number of I/O pins of the circuit is (16 + 16 + 2 + 2) = 36, respectively 19 for inputs and 17 for outputs.



Setup

Worst Negative Slack (WNS):	3,017 ns
Total Negative Slack (TNS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	202

Hold

Worst Hold Slack (WHS):	0,163 ns
Total Hold Slack (THS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	202

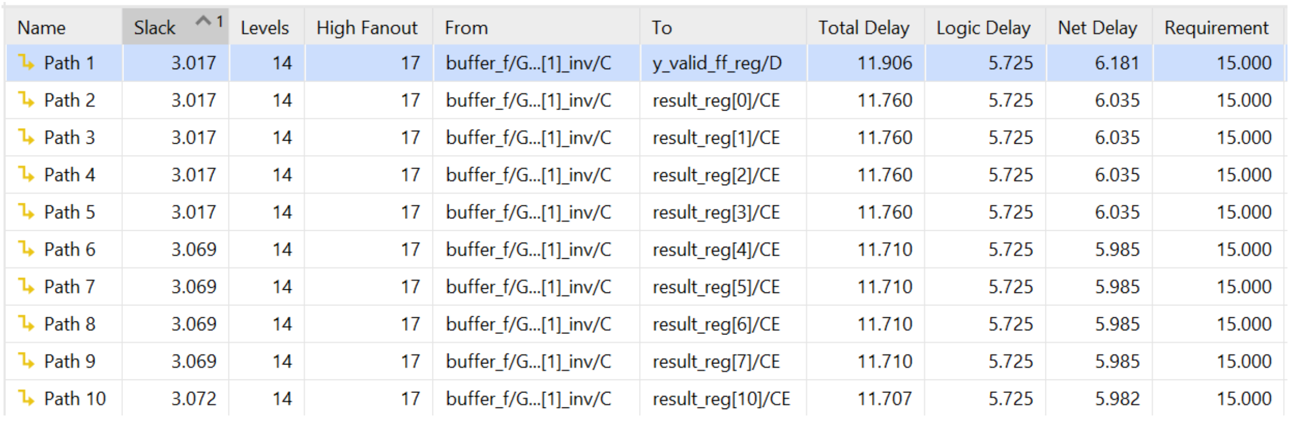
Pulse Width

Worst Pulse Width Slack (WPWS):	6,520 ns
Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	122

All user specified timing constraints are met.

Figure 8 – Timing report.

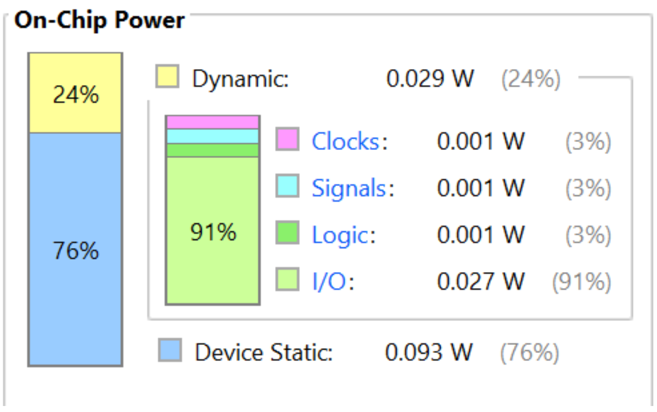
Critical Paths



Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	3.017	14	17	buffer_f/G...[1]_inv/C	y_valid_ff_reg/D	11.906	5.725	6.181	15.000
Path 2	3.017	14	17	buffer_f/G...[1]_inv/C	result_reg[0]/CE	11.760	5.725	6.035	15.000
Path 3	3.017	14	17	buffer_f/G...[1]_inv/C	result_reg[1]/CE	11.760	5.725	6.035	15.000
Path 4	3.017	14	17	buffer_f/G...[1]_inv/C	result_reg[2]/CE	11.760	5.725	6.035	15.000
Path 5	3.017	14	17	buffer_f/G...[1]_inv/C	result_reg[3]/CE	11.760	5.725	6.035	15.000
Path 6	3.069	14	17	buffer_f/G...[1]_inv/C	result_reg[4]/CE	11.710	5.725	5.985	15.000
Path 7	3.069	14	17	buffer_f/G...[1]_inv/C	result_reg[5]/CE	11.710	5.725	5.985	15.000
Path 8	3.069	14	17	buffer_f/G...[1]_inv/C	result_reg[6]/CE	11.710	5.725	5.985	15.000
Path 9	3.069	14	17	buffer_f/G...[1]_inv/C	result_reg[7]/CE	11.710	5.725	5.985	15.000
Path 10	3.072	14	17	buffer_f/G...[1]_inv/C	result_reg[10]/CE	11.707	5.725	5.982	15.000

Figure 9 – The worst paths of the circuit.

Power Consumption



On-Chip Power

🟨	Dynamic:	0.029 W	(24%)
🟪	Clocks:	0.001 W	(3%)
🟦	Signals:	0.001 W	(3%)
🟩	Logic:	0.001 W	(3%)
🟩	I/O:	0.027 W	(91%)
🟦	Device Static:	0.093 W	(76%)

Figure 10 – Power consumption report.

So, from the latter we obtain the maximum clock frequency:

$$f_{max} = \frac{1}{T_{clk} - WNS} = 83{,}452\ MHz$$

Equation 2 – Maximum clock frequency.

Conclusions

In conclusion, the design and implementation phases of the iir_filter has followed the following steps:

- Design of the architecture of the filter in VHDL by defining all the necessary modules which are the following:
  - d_flip_flop;
  - d_flip_flop_n;
  - buffer_fifo;
  - iir_filter.
- Implementation of the behaviour of the filter with a VHDL sequential process, in which the two outputs (y and y_valid) are computed.
- Synthesis and implementation in Vivado with timing, critical paths and power consumption analysis.

After these steps the IIR filter is ready to be used in one of the audio applications mentioned in the introduction. A possible improvement that can be done is to create a module for each specific functionality implemented in the sequential process, one for the full buffer check, one for the overflow check and one for the difference equation.

This improvement can be useful for a better organization of the code and its readability, moreover it is also better for future changes in the behaviour of the filter because it would be more complicated to modify a non-modular code such as that of the process, especially when the code becomes bigger.