Harish V. Dixit, Dr. Vikas Gupta / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 5, September- October 2012, pp.303-307

IIR filters using Xilinx System Generator for FPGA Implementation

Harish V. Dixit , Dr. Vikas Gupta; V.C.E.T

Abstract:

This paper proposes the implementation of IIR filter using Xilinx System Generator software on an FPGA. While designing any digital filter, the overflow and quantisation effects must be considered for stability. The speed of computation is greatly increased by implementing a filter on an FPGA, rather than a Conventional DSP processor. We first design the filter analytically from the given specifications and simulate it using the Simulink environment in Matlab. A method to implement this filter using Xilinx System Generator is proposed.

Keywords : Simulink, Xilinx, FPGA, System Generator, Filter

1. Introduction

Digital signal processors or Application Specific Integrated Circuits (ASICs) are commonly used for implementing the digital filtering algorithms. However recent advances in the technology of Field Programmable Gate Array (FPGA) have led to the implementation of these algorithms on FPGAs. This paper proposes a approach to implement an IIR filter on Field Programmable Gate Arrays (FPGAs) using Xilinx System Generator software.

A conventional DSP processor is a serial device and typically has 1-4 MAC units along with barrel shifters and other circuits for efficient computations. A substantial part of the computations involves the use of multiplication and accumulation operations. Supposing a DSP unit has a single MAC unit. A 256 tap filter involves 256 MAC operations per sample. Hence with a single MAC unit, it takes 256 clock cycles for the output to be computed in a typical DSP processor. The speed of computation can be increased by increasing the clock rate which in turn increases the system complexity. The following figure shows the computation factors involved in a DSP device:

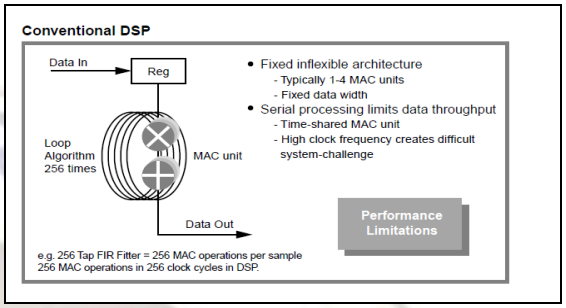


Fig1. Implementation of Filter on DSP device [1]

The speed of computation can be increased substantially without affecting the system complexity using the parallelism feature of FPGA. The FPGA contains a large number of gates and millions of transistors. Hence the filter can be implemented in a parallel manner as shown in the fig. 2. The implementation consists of 256 registers and 256 multiplier unit along with an adder for final partial product. Hence what took 256 clock cycles on a DSP processor takes a single clock cycle on a FPGA. Fig. 2 shows the implementation of the same filter on an FPGA.

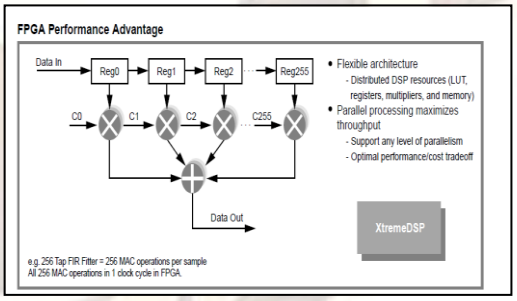


Fig2. Implementation of Filter on FPGA [1]

2. Digital Filters

A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. Based on the length of impulse response, digital filters are classified into two types

1) Finite Impulse Response (FIR) Filters
2) Infinite Impulse Response (IIR) Filters

The IIR filters are not well supported by softwares and Intellectual Property cores as compared to the FIR filters. In this paper, we discuss the implementation of IIR Filters on FPGAs.

3. IIR Filters

The IIR filter can be implemented using the following structures:

1) Direct Form I
2) Direct Form II

In this paper, we use the Direct Form I approach.

IIR filter consists of a forward path as well as a feedback path, both paths contributing to the output outgoing samples. A forward path is typically a short FIR like structure. The forward FIR filter, also known as all-zero filter, comprises of the numerator, or b, coefficients for the zeros, and a feedback FIR for the denominator, or a, coefficients for the poles. A way to express an IIR Filter is as a z-transfer function with numerator coefficients b_i and denominator coefficients a_i.

Each output sample is a sum of a new input and earlier input values and subtraction of previous output values, all multiplied by their respective

$$H(z) = \frac{\sum_{i=0}^{n} biz_i}{\sum_{i=0}^{m} aiz_i}$$

coefficients Computationally, feedback subtraction equals to addition with inverted coefficient values. The time domain expression for the IIR is shown in following, and it can be seen that some delayed version of the y(n) output is playing a part in the output:

a(i) and b(i) are the coefficients of the IIR filter.

The expression is now showing summation, multiplication, and subtraction, which are basic DSP building blocks and can be implemented in FPGA architecture using tools like System Generator.

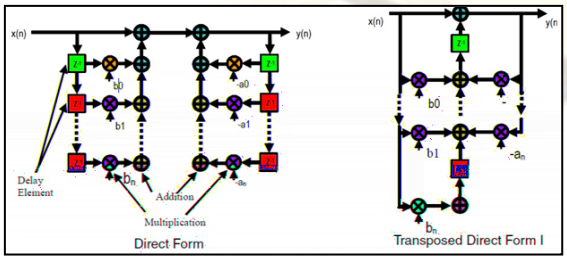


Figure 3 - Structure For IIR Filter [2]

4. Implementation Issues of IIR Filter

To ensure satisfactory performance of the fixed point implementation of IIR Filter, the following factors should be considered [2]. These factors are-

1) Co-efficient Quantisation
2) Internal Quantisation
3) Overflow
4) Stability

Coefficient Quantisation

For implementing a digital filter, sofwares are used to define the filter coefficients based on the given filter specifications. The filter design software usually computes and displays the filter coefficients with a high degree of precision. If the digital filter can be implemented using that same degree of precision, then the filter will behave as predicted by the filter design software.

In practice, only a finite number of bits can be used to represent the digital filter coefficients. This reduction in each coefficient's precision causes the frequency response of the filter to differ from the "ideal" response due to coefficient quantization errors.

When using B bits to represent the filter

$$y(n) = \sum_{i=0}^{n} b_{(i)}x(n-i) + \sum_{i=0}^{m} a_{(i)}y(m-i)$$

coefficients, the total number of possible values that the filter coefficients can take on is 2^B. Thus, instead of having an infinite range of values for the coefficients, they are instead constrained to one of the 2^B levels.

The location of the poles and zeros of the filter are also quantized. This is because they depend on the value of the filter coefficients. The quantization of the pole and zero locations will typically move the poles and zeros of the filter to

locations that are different from the "ideal" setting.
This can have drastic effects on the performance of the filter.

Internal Quantisation

A DSP function involves multiplication and addition/subtraction operations. However, there is bit growth due to these operations, and at some point, the bit widths have to be reduced. Operations like wrapping/saturation for the most significant bits and rounding/truncation for the least significant bits have to be used. The rounding process reduces bit width but is a source of noise and contributes to output round-off noise and, hence, affects the signal-to-noise ratio. The flexibility of FPGA

architecture allows for increasing the word length and reducing the round-off noise

Overflow

For a fixed-point implementation, there is a certain bit width and, hence, a range. As a result of calculations, the filter may exceed its maximum/minimum ranges. To minimize the effects of overflows, scaling can be used. Therefore, values can never overflow. There are different kinds of scaling and these tend to be used by DSP processors to fit within their fixed structure. However, this has an effect on the signal to noise ratios.

Stability

The roots of the denominator polynomial must be less than 1. It must be ensured that the quantisation effects do not shift the roots to be greater than 1 rendering the filter unstable.

5. Implementation Methodology

The following implementation methodology is identified. Various steps according to flow graph are:

Step 1: Design the Filter

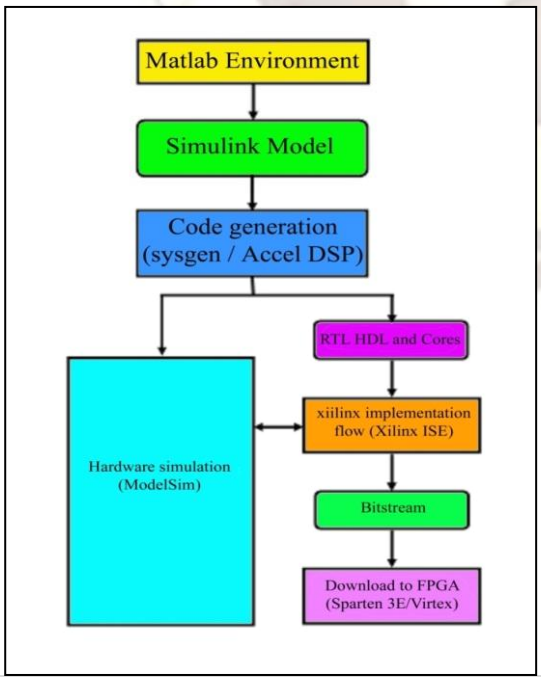
Step 2: Create Simulink Model.

Figure 4 - Implementation Methodology

Step 3: Implement Simulink model with the help of MATLAB.

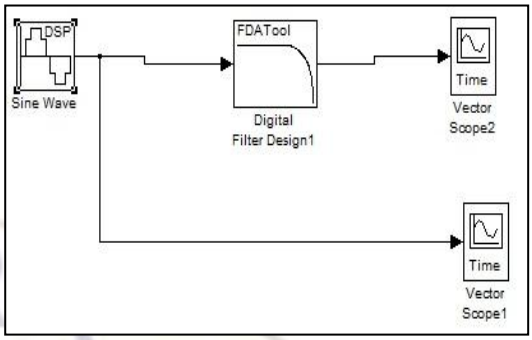
Step 4: Code generation using SysGen/Accel DSP.

Step 5: Simulate and debug the logic program and make necessary correction to design of Step 3.



Step 6: Simulate and generate RTL schematic and check for resource utilization

Step 7: Run an automatic place and route program.



This will place the logic block in appropriate places in FPGA and then route the interconnection between logic blocks.

Step 8: Run a program that will generate the bit pattern necessary to program FPGA.

Step 9: Download bit pattern into internal configuration, memory cells in FPGA and test operation of FPGA.

6. Filter Specifications

An IIR Butterworth Low Pass Filter with the following specifications is desired:

Attenuation in pass band, Ap = 1

Attenuation in stop band, A_s = 22

Pass band Frequency, f_p= 2.1Khz

Stob Band Frequency, f_s = 2.9 Khz

Sampling Frequency, f_{samp} = 6.5Khz

The transfer function of such a filter is obtained as

$$\therefore H(z) = \frac{0.3838 + 1.1514z^{-1} + 1.1514z^{-2} + 0.3838z^{-3}}{1 + 1.20287z^{-1} + 0.7224z^{-2} + 0.1451z^{-3}}$$

7. Simulation Results

The Simulink environment from Mathworks MATLAB is used for the simulation of the designed filter. Bandlimited white noise is given as an input to the FDA Tool and the output is obtained on the spectrum scope. The output on the scope verifies the stop band frequency of 2.9 Khz.

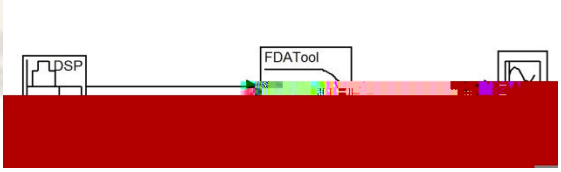


Figure 5 – set up for Frequency response of filter

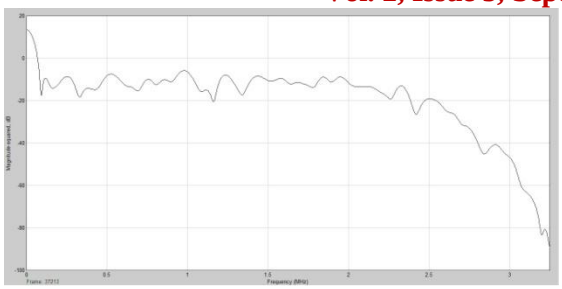


Figure 6 – frequency response of filter

Figure 7 – Illustration of filtering

We now illustrate the filtering action by giving the filter two sinusoidal inputs of frequency 2.1Khz and 2.9 Khz. The 2.1Khz input is passed to the scope as it falls within the cutoff range and the 2.9 Khz input is heavily attenuated as falls near the cutoff frequency.

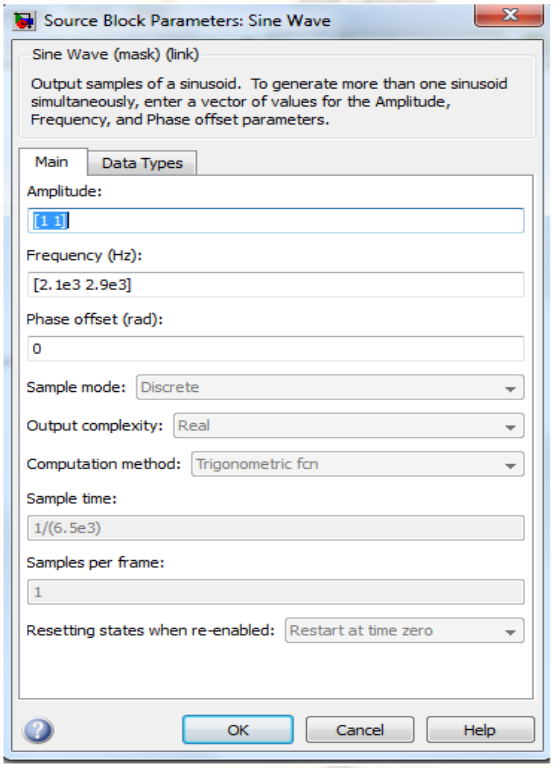


Figure 8 – inputs to the filter

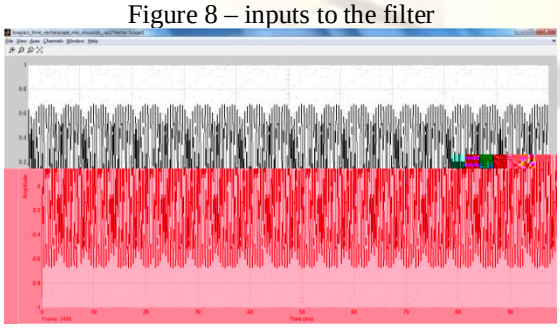


Figure 9 – Output without filter

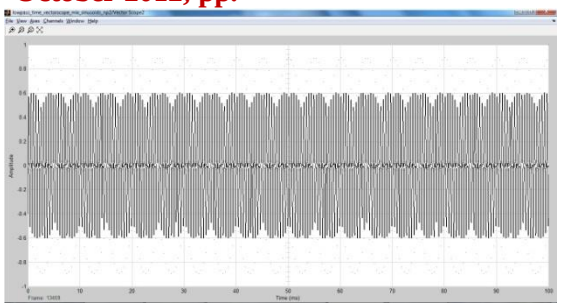


Figure 10 – output with filter

Synthesis

As seen, the simulation results have matched the specifications and Xilinx System Generator For DSP software is used to synthesise this filter. The following architecture is proposed for the implementing the designed filter using the basic available blocks in the Xilinx DSP blockset of System Generator like adders, multipliers and delays

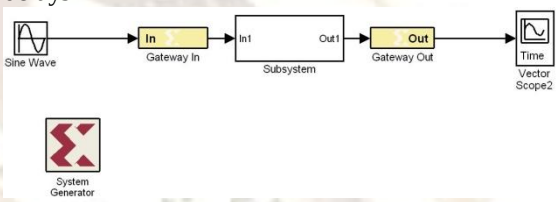


Figure 11 – Implementation of filter in System Generator

In the above figure the Third Order block is modelled as follows.

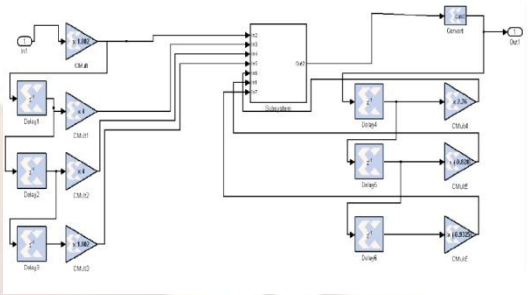


Figure 12 – Implementation of Filter block

In the above figure the adder block is modelled as,

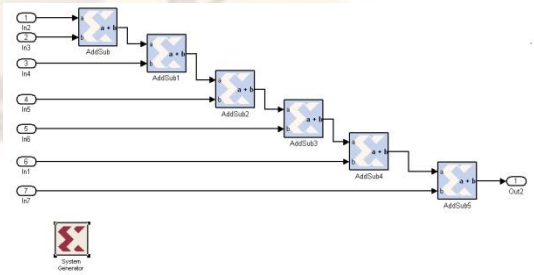


Figure 13 – Implementation of Adder block

Conclusion:

This Paper discusses the implementation of an IIR Digital filter on an FPGA. The parallel processing capability of the FPGA greatly increases the speed of operation in the implementation of the

Digital filter. Simulink is used to obtain the simulation of the designed filter. The filter may be rendered unstable because of quantisation effects and overflow errors. So, it must be carefully designed and guarded against them. Finally, a method is proposed to implement this filter using Xilinx System Generator.

References:

1. Xilinx white paper number 213, www.xilinx.com
2. Xilinx white paper number 330, www.xilinx.com
3. Chi-Jui Chou, Satish Mohanakrishnan, Joseph B. Evans, FPGA implementation of digital filters, Proc. of ICSPAT, 1993.
4. Juan J. Rodriguez, Andina Maria J. Moure, Maria D. Valdes, Features, design tools and application domains of FPGA, IEEE transactions on industrial electronics, 2007, pp: 1810-1823.
5. Louis Litwin, FIR and IIR digital filters, IEEE potentials, 2002.
6. Robert Esposito, Digital signal processing : A hardware based approach, proc of the 2007 middle Atlantic section fall conference of the American society for engineering education.
7. Sanjit K. Mitra, Digital signal processing : a computer based approach, McGraw Hill, 2006.
8. Xilinx system generator, basic tutorial, www.xilinx.com