ISSN 2347 - 3983
Volume 8. No. 9, September 2020
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter44892020.pdf
https://doi.org/10.30534/ijeter/2020/44892020

Pipelined Digital Filters and Their Applications: FDATOOL Design and VERILOG HDL Verification

Phuong H. Lai[1], Tung V. Nguyen[2], Khoa D. Ta[2], Thien V. Truong[2], Duong B. Nguyen[2], Phong H. Nguyen[3]
[1]Dept. of Computing Fundamentals, FPT University, Hanoi, Vietnam, phuonglh17@fe.edu.vn
[2]ICT Department, FPT University, Hanoi, Vietnam, tungnvhe130151@fpt.edu.vn,khoatdhe130813@fpt.edu.vn, thientvse04522@fpt.edu.vn, duongnbhe130658@fpt.edu.vn
[3]Center of Machine Vision & Signal Analysis, University of Oulu, Finland; phong.nguyen@oulu.fi

ABSTRACT

This research will provide system on chip design for pipelined digital filters module.Two basic but important FIR and IIR filters are going to be discussed. At first, the position of digital filters in digital system is explained. Then, MATLAB *fdatool* and scripts are used for filter design. Finally ,the implementation and verification of proposed filter processor are performed VERILOG hardware description language (HDL).All scripts, algorithm is clearly given. We hope that the research will be a great reference and an intellectual property core for engineers and researcher students.

Key words: Application specific integrated design (ASIC), digital system design, digital filters, FIR, IIR, VerilogHDL, MATLAB fdatool,Model Simulation.

1. INTRODUCTION

Nowadays, it is no doubt about the important of digital signal on human beings [1]. Everywhere, every time, digital signal helps our living more comfortable and we cannot live without them. The system of generating, working with digital signals can be called digital signal processing system [2]. There are many sub-modules combined to DSP system and among them, digital filters are signal conditioners where their functions accept the raw signal to remove un-wanted parts and pass the remaining to output [3]. Filters can be found everywhere in our current digital age, for example a telephone line system which limits a range of human hearing; consequently, the quality of listening CD-music over the phone is not as good as we hear it directly [4]. In image processing, we also have the suitable filter for image such as RGB filter, image sharpening filter, image blur filter. Processing over time-domain signal are the common feature of digitals filters and the filters can be expressed as a convolution operator over time-domain signal [5].
The purpose of our study is to analysis and design the system on chip system of two popular types of digital filter: FIR and IIR. The study is flexible, open and useful to be the intellectual logic cores which can be reused in future digital system with suitable digital design [6]. All the source codes and data of the study are conducted by using Model Simulation 10.4a tool student version and MATLAB R2019b.We organize the study as follows.In Section 2, we explain the role and position of FFT block on digital system. The analysis and design of pipeline FFT processor is explained in Section 3. In Section 4, the implementation and verification of FFT processor design are discussed. Finally, conclusion is given in Section 5.

2. ROLE OF DIGITAL FILTERS MODULE ON DIGITAL SYSTEMS

A. Overview of digital filter

As classification via input signals, the filters can be divided into analogue filter and digital filter. An analogue filter combines many analogue components such as resistors, capacitors, and inductors which is widely used in many applications such as noise reduction, video signal enhancement, graphic equaliser. In contrast, a digital filter uses the discrete input signals. Therefore, it performs numerous calculations on sampled values of signal, the overall system model of digital filter can be seen in figure 1. In comparison with analogue filter, digital filter gives many advantages. The main advantages of digital filter are programable and easily to design, test, and implement. In this research, we focus on the digital filter, its design, and its application.

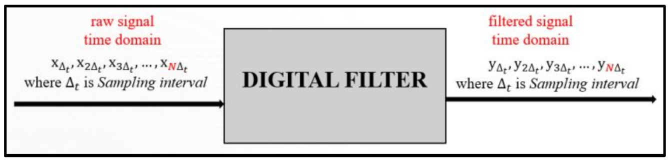

raw signal
time domain
$x_{\Delta_t}, x_{2\Delta_t}, x_{3\Delta_t}, \dots, x_{N\Delta_t}$
where Δ_t is *Sampling interval*
DIGITAL FILTER
filtered signal
time domain
$y_{\Delta_t}, y_{2\Delta_t}, y_{3\Delta_t}, \dots, y_{N\Delta_t}$
where Δ_t is *Sampling interval*

Figure 1:Overall model of a digital filter.

B. Digital filter module on digital system

Since digital filter performs the calculation on discrete signal, the analogue input signal must first be sampled and digitalized using an Analogue-Digital-converter (ADC). As a consequence, in an overall digital system as figure 2, the position of digital filter is located after an ADC, and digital filters output will be inputs to Fast Fourier transform (FFT) module or are controlled to be input of Digital-Analogue-converter to convert back to suitable analogue form. Digital filter is an important part of digital system such as a general-purpose PC, or a specialized DSP chip.

The most basic but popular digital filter is finite impulse response (FIR) which is usually implemented by a series of delays, multipliers, and adders, but there are no recursive parts. In contrast, we have infinite impulse response (IIR) which uses feed-back to keep more historical information active in the calculation. The processing of selecting filter length, filter coefficients are called digital filter design. *MATLAB fdatool* is a popular filter design tool, which is used for almost engineers to utilize digital filters, and it will be used through this research for the digital filter design.

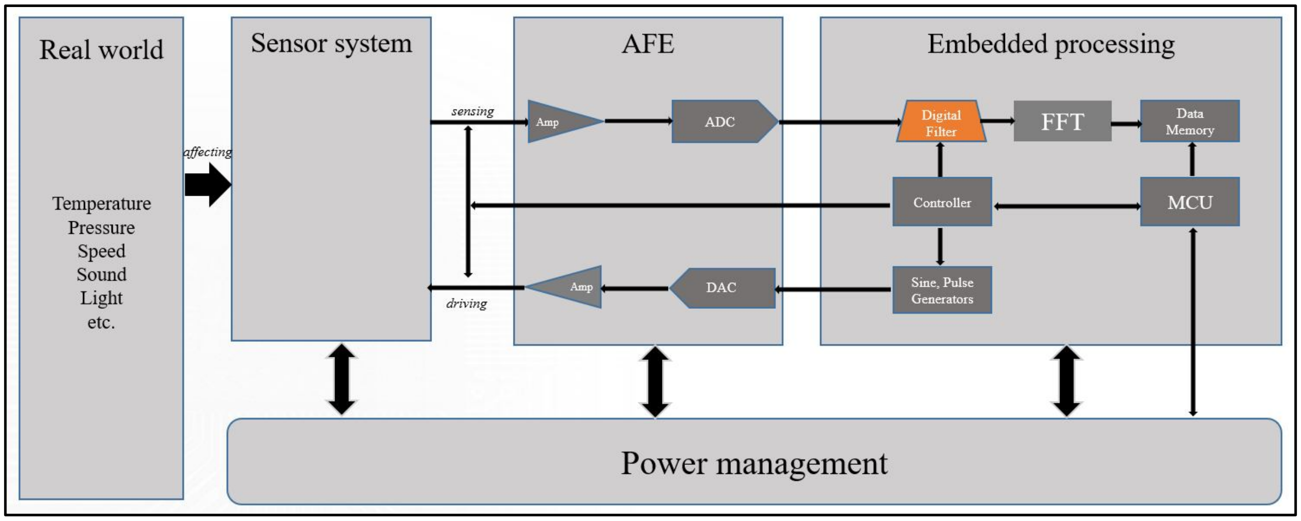
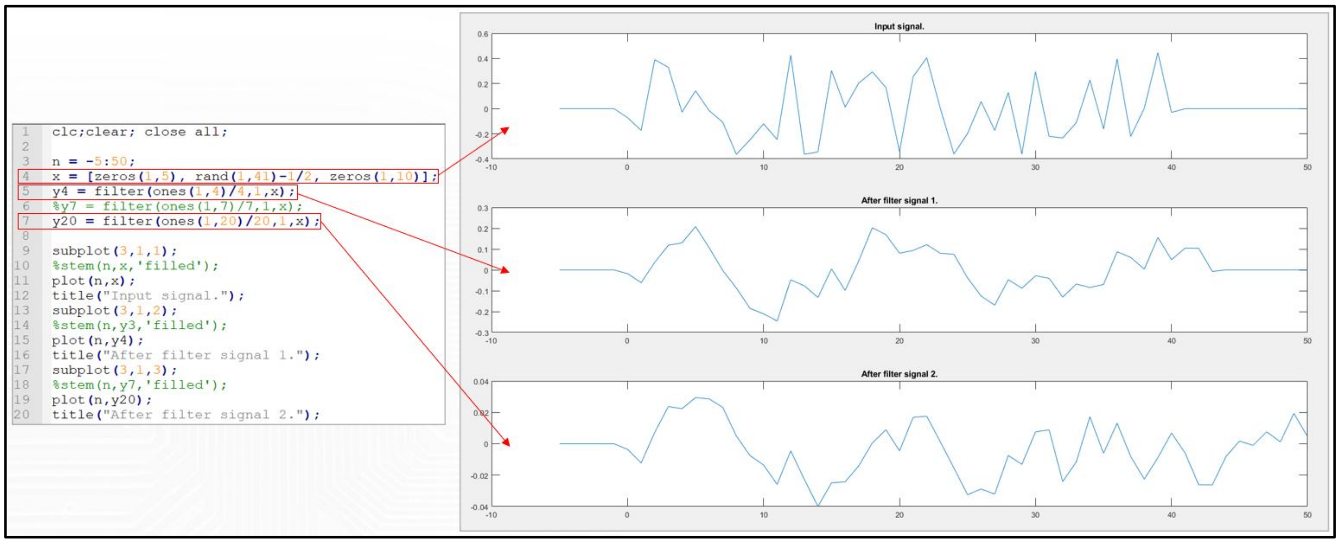


Figure 2: Position of a digital filter module on an overall digital system.



```
clc;clear; close all;

n = -5:50;
x = [zeros(1,5), rand(1,41)-1/2, zeros(1,10)];
y4 = filter(ones(1,4)/4,1,x);
%y7 = filter(ones(1,7)/7,1,x);
y20 = filter(ones(1,20)/20,1,x);

subplot(3,1,1);
%stem(n,x,'filled');
plot(n,x);
title("Input signal.");
subplot(3,1,2);
%stem(n,y3,'filled');
plot(n,y4);
title("After filter signal 1.");
subplot(3,1,3);
%stem(n,y7,'filled');
plot(n,y20);
title("After filter signal 2.");
```

Figure 3: Smoothing FIR filter simulation in MATLAB.

3. DESIGN AND IMPLEMENTATION OF PIPELINED FIR FILTER

A. Analysis of FIR filter

Finite impulse response filter (FIR) is defined by scaled by coefficients and time-delayed versions of inputs signal, which has a general difference equation as follows,

$$\mathrm{y}[n] = \sum_{i=0}^{N-1} h_i \times \mathrm{x}[n-i],$$

where x is input signal (raw signal), y is filtered output, N is filter order, and h is coefficient of filters or impulse response.The above computation is also called the discrete convolution. As a simple structure, FIR filter has numerous properties make it to be preferable. The block diagram for 4-orders digital FIR filter is given in figure 4 where D denotes the delay module.

The most basic application of FIR filter is a moving average or smoothing filter. For example, when $h_1 = h_2 = h_3 = h_4$we have an averaging module with order 4. In figure 3, we simulate order4,20 FIR filters by using MATLAB. The length of signal is 56, inputs signal x is just a random number, its graph is shown in the first paragraph at right corner. The y4 and y20 are the FIR filters outputs for order-4 and order-20. Those paragraphs are shown. As we can see from the figure, the more order number we set, the more smoothing output we get. One of current application of FIR filter is in the pressure part in a communication between an active pen and touch screen where the 8 bits input is extended to 10-bits (1024 pressure level) with some suitable coefficients.

B. Implementation of FIR filter processor

As a block diagram shown in figure 4, we use Verilog HDL to implement FIR filter for 4-orders. The implementation is given as figure 8 where D-type Flip-flop module is used for one clock delay module, main module name is fir_4tap. The testbench is also given in figure 8 where MATLAB script to generate the inputs is attached. The example for testbench is same as the MATLAB simulation example in figure 3. The wave form of FIR design is shown in figure 9, the schematic view of FIR design also given in figure 9. As can be seen, result of proposed design is same as software simulation.

4. DESIGN AND IMPLEMENTATION OF PIPELINED IIR FILTER

A. Analysis of IIR filter

The Infinite impulse response (IIR) gets their name since their impulse response extends for infinite period by recursive or feedback. The equation for IIR is given as,

$$\sum_{i=0}^{N} a_i \times y[n-i] = \sum_{i=0}^{N} b_i \times x[n-i],$$

where x is input signal (raw signal), y is filtered output, N is filter order, and a_i, b_i are coefficient of filters, $a_0 = 1$. As the equation for IIR filter, the corresponding block diagram is explained as figure 5 where we can use several sessions in the filter design.

B. Implementation of IIR filter processor

In this research, we design IIR filter for low pass filter purpose. Firstly, MATLAB *fdatool* is used for filter design as given in figure 10 where we input the sample frequency F_s, pass frequency F_{pass}, stop frequency F_{stop}. Then, we will get the parameters of IIR filters such as number of sections (three) and numerators and denominators which are a_i, b_i (coefficient of filters). In next step, we need to convert floating point number into integer forms to be used in Verilog HDL. The figure 6 shows the MATLAB script and the results of conversion. Consequently, the Verilog module of IIR is given in figure 11. The testbench for verification is given as figure 7 where we consider the input as a mixed signal of three frequency f_1, f_2, f_3. The MATLAB script to generate mixed signal is also given in figure 3 as well as testbench. Due to the length of input signal, the main part of testbench script is shown in figure 7, the remaining parts is same as output of attached MATLAB script. The wave form result of proposed design is presented in figure 12 where x is plot of mixed signal, y is filtered signal which consists of only expected frequency.

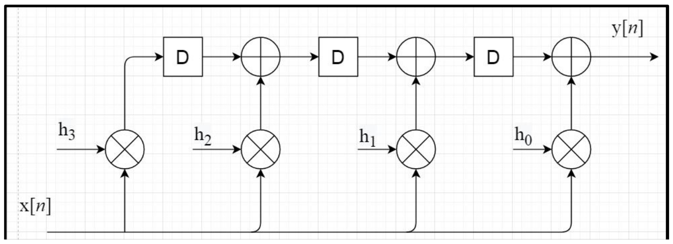


Figure 4: A FIR filter block diagram.

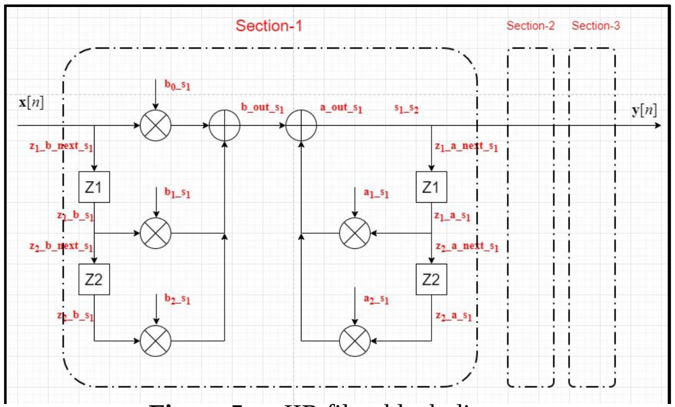
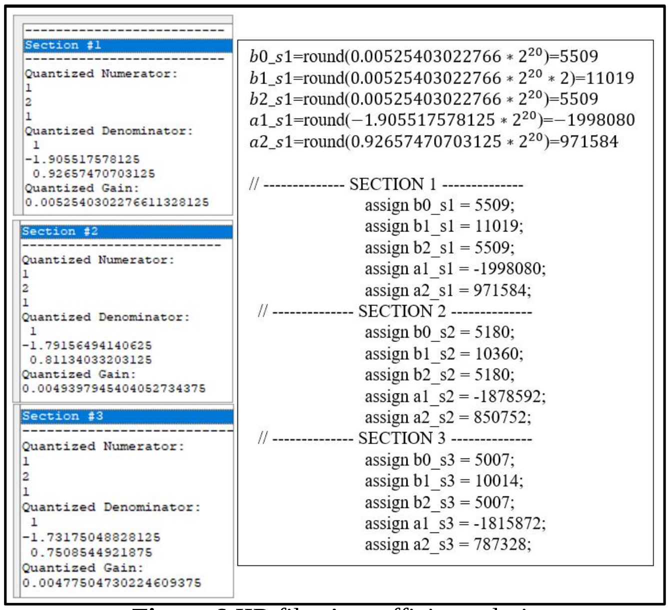


Figure 5:an IIR filter block diagram.



```
---------------------------
Section #1
---------------------------
Quantized Numerator:
1
2
1
Quantized Denominator:
 1
-1.905517578125
 0.92657470703125
Quantized Gain:
0.0052540302276611328125
```

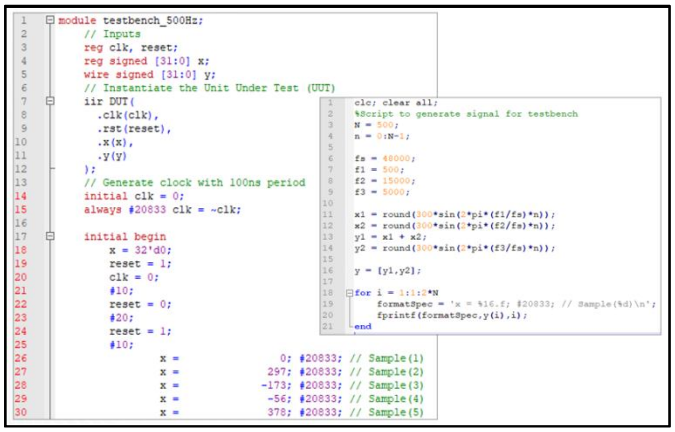
```
Section #2
---------------------------
Quantized Numerator:
1
2
1
Quantized Denominator:
 1
-1.79156494140625
 0.81134033203125
Quantized Gain:
0.0049397945404052734375
```

```
Section #3
---------------------------
Quantized Numerator:
1
2
1
Quantized Denominator:
 1
-1.73175048828125
 0.7508544921875
Quantized Gain:
0.00477504730224609375
```

$b0\_s1$=round(0.00525403022766 $*\ 2^{20}$)=5509
$b1\_s1$=round(0.00525403022766 $*\ 2^{20} *\ 2$)=11019
$b2\_s1$=round(0.00525403022766 $*\ 2^{20}$)=5509
$a1\_s1$=round(−1.905517578125 $*\ 2^{20}$)=−1998080
$a2\_s1$=round(0.92657470703125 $*\ 2^{20}$)=971584

```
// -------------- SECTION 1 --------------
        assign b0_s1 = 5509;
        assign b1_s1 = 11019;
        assign b2_s1 = 5509;
        assign a1_s1 = -1998080;
        assign a2_s1 = 971584;
 // -------------- SECTION 2 --------------
        assign b0_s2 = 5180;
        assign b1_s2 = 10360;
        assign b2_s2 = 5180;
        assign a1_s2 = -1878592;
        assign a2_s2 = 850752;
 // -------------- SECTION 3 --------------
        assign b0_s3 = 5007;
        assign b1_s3 = 10014;
        assign b2_s3 = 5007;
        assign a1_s3 = -1815872;
        assign a2_s3 = 787328;
```

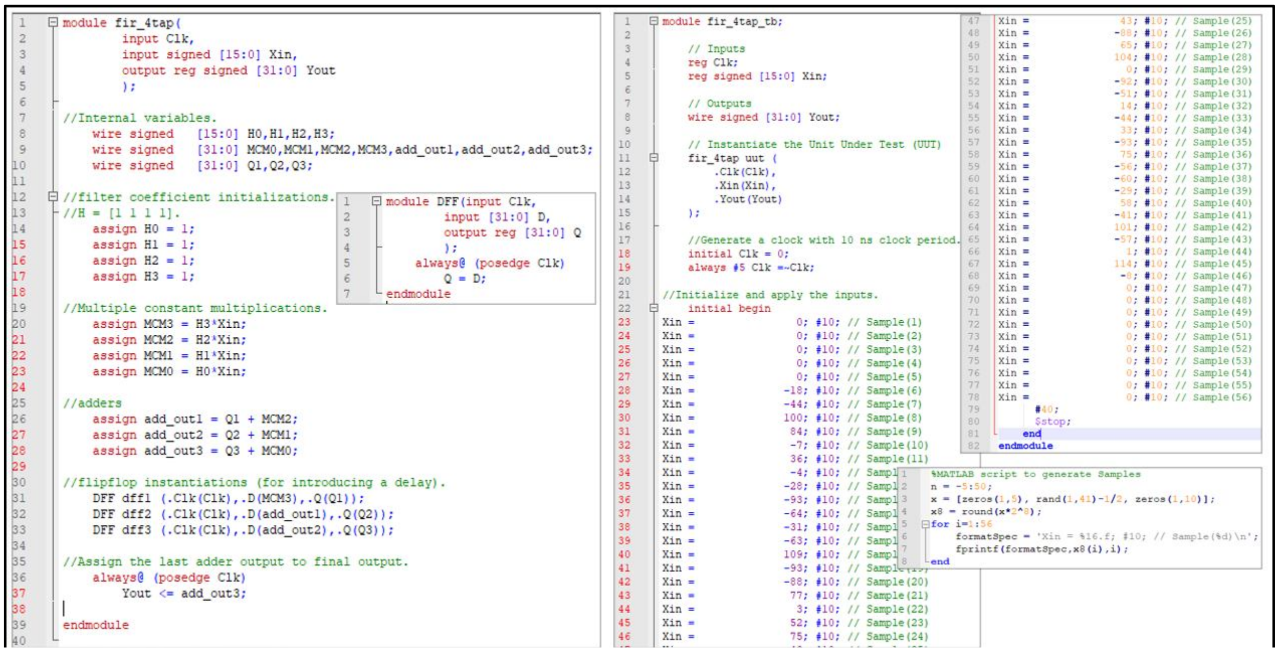
Figure 6:IIR filter's coefficients design.



```
module testbench_500Hz;
    // Inputs
    reg clk, reset;
    reg signed [31:0] x;
    wire signed [31:0] y;
    // Instantiate the Unit Under Test (UUT)
    iir DUT(
      .clk(clk),
      .rst(reset),
      .x(x),
      .y(y)
    );
    // Generate clock with 100ns period
    initial clk = 0;
    always #20833 clk = ~clk;

    initial begin
        x = 32'd0;
        reset = 1;
        clk = 0;
        #10;
        reset = 0;
        #20;
        reset = 1;
        #10;
                x =             0; #20833; // Sample(1)
                x =           297; #20833; // Sample(2)
                x =          -173; #20833; // Sample(3)
                x =           -56; #20833; // Sample(4)
                x =           378; #20833; // Sample(5)
```

```
clc; clear all;
%Script to generate signal for testbench
N = 500;
n = 0:N-1;

fs = 48000;
f1 = 500;
f2 = 15000;
f3 = 5000;

x1 = round(300*sin(2*pi*(f1/fs)*n));
x2 = round(300*sin(2*pi*(f2/fs)*n));
y1 = x1 + x2;
y2 = round(300*sin(2*pi*(f3/fs)*n));

y = [y1,y2];

for i = 1:1:2*N
    formatSpec = 'x = %16.f; #20833; // Sample(%d)\n';
    fprintf(formatSpec,y(i),i);
end
```

Figure 7:A testbench for IIR filter processor.



```verilog
module fir_4tap(
        input Clk,
        input signed [15:0] Xin,
        output reg signed [31:0] Yout
        );

//Internal variables.
    wire signed   [15:0] H0,H1,H2,H3;
    wire signed   [31:0] MCM0,MCM1,MCM2,MCM3,add_out1,add_out2,add_out3;
    wire signed   [31:0] Q1,Q2,Q3;

//filter coefficient initializations.
//H = [1 1 1 1].
    assign H0 = 1;
    assign H1 = 1;
    assign H2 = 1;
    assign H3 = 1;

//Multiple constant multiplications.
    assign MCM3 = H3*Xin;
    assign MCM2 = H2*Xin;
    assign MCM1 = H1*Xin;
    assign MCM0 = H0*Xin;

//adders
    assign add_out1 = Q1 + MCM2;
    assign add_out2 = Q2 + MCM1;
    assign add_out3 = Q3 + MCM0;

//flipflop instantiations (for introducing a delay).
    DFF dff1 (.Clk(Clk),.D(MCM3),.Q(Q1));
    DFF dff2 (.Clk(Clk),.D(add_out1),.Q(Q2));
    DFF dff3 (.Clk(Clk),.D(add_out2),.Q(Q3));

//Assign the last adder output to final output.
    always@ (posedge Clk)
        Yout <= add_out3;

endmodule
```

```verilog
module DFF(input Clk,
        input [31:0] D,
        output reg [31:0] Q
        );
    always@ (posedge Clk)
        Q = D;
endmodule
```

```verilog
module fir_4tap_tb;

    // Inputs
    reg Clk;
    reg signed [15:0] Xin;

    // Outputs
    wire signed [31:0] Yout;

    // Instantiate the Unit Under Test (UUT)
    fir_4tap uut (
        .Clk(Clk),
        .Xin(Xin),
        .Yout(Yout)
    );

    //Generate a clock with 10 ns clock period.
    initial Clk = 0;
    always #5 Clk =~Clk;

//Initialize and apply the inputs.
    initial begin
Xin =                   0; #10; // Sample(1)
Xin =                   0; #10; // Sample(2)
Xin =                   0; #10; // Sample(3)
Xin =                   0; #10; // Sample(4)
Xin =                   0; #10; // Sample(5)
Xin =                 -18; #10; // Sample(6)
Xin =                 -44; #10; // Sample(7)
Xin =                 100; #10; // Sample(8)
Xin =                  84; #10; // Sample(9)
Xin =                  -7; #10; // Sample(10)
Xin =                  36; #10; // Sample(11)
Xin =                  -4; #10; // Sampl
Xin =                 -28; #10; // Sampl
Xin =                 -93; #10; // Sampl
Xin =                 -64; #10; // Sampl
Xin =                 -31; #10; // Sampl
Xin =                 -63; #10; // Sampl
Xin =                 109; #10; // Sampl
Xin =                 -93; #10; // Sample
Xin =                 -88; #10; // Sample(20)
Xin =                  77; #10; // Sample(21)
Xin =                   3; #10; // Sample(22)
Xin =                  52; #10; // Sample(23)
Xin =                  75; #10; // Sample(24)
Xin =                  43; #10; // Sample(25)
Xin =                 -88; #10; // Sample(26)
Xin =                  65; #10; // Sample(27)
Xin =                 104; #10; // Sample(28)
Xin =                   0; #10; // Sample(29)
Xin =                 -92; #10; // Sample(30)
Xin =                 -51; #10; // Sample(31)
Xin =                  14; #10; // Sample(32)
Xin =                 -44; #10; // Sample(33)
Xin =                  33; #10; // Sample(34)
Xin =                 -93; #10; // Sample(35)
Xin =                  75; #10; // Sample(36)
Xin =                 -56; #10; // Sample(37)
Xin =                 -60; #10; // Sample(38)
Xin =                 -29; #10; // Sample(39)
Xin =                  58; #10; // Sample(40)
Xin =                 -41; #10; // Sample(41)
Xin =                 101; #10; // Sample(42)
Xin =                 -57; #10; // Sample(43)
Xin =                   1; #10; // Sample(44)
Xin =                 114; #10; // Sample(45)
Xin =                  -8; #10; // Sample(46)
Xin =                   0; #10; // Sample(47)
Xin =                   0; #10; // Sample(48)
Xin =                   0; #10; // Sample(49)
Xin =                   0; #10; // Sample(50)
Xin =                   0; #10; // Sample(51)
Xin =                   0; #10; // Sample(52)
Xin =                   0; #10; // Sample(53)
Xin =                   0; #10; // Sample(54)
Xin =                   0; #10; // Sample(55)
Xin =                   0; #10; // Sample(56)
        #40;
        $stop;
    end
endmodule
```

```matlab
%MATLAB script to generate Samples
n = -5:50;
x = [zeros(1,5), rand(1,41)-1/2, zeros(1,10)];
x8 = round(x*2^8);
for i=1:56
    formatSpec = 'Xin = %16.f; #10; // Sample(%d)\n';
    fprintf(formatSpec,x8(i),i);
end
```

Figure 8: A FIR filter implementation via VERILOG HDL.

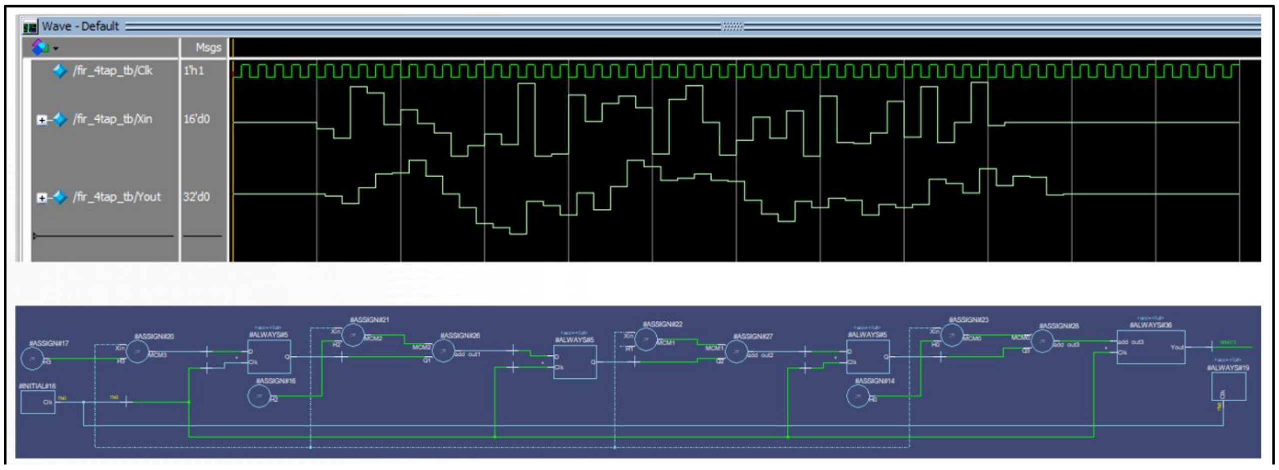
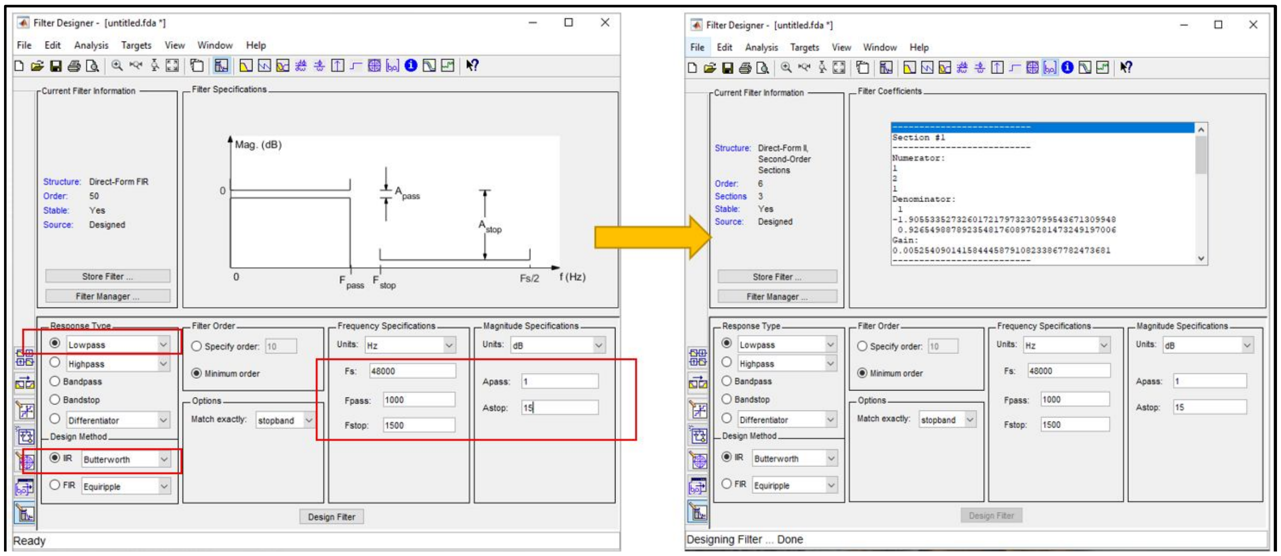


Figure 9: Wave form result and Schematic view of FIR processor.



Current Filter Information
Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: [✓ IIR Butterworth] FIR Equiripple
Filter Order: Specify order: 10 [✓ Minimum order]
Options: Match exactly: stopband
Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 1000; Fstop: 1500
Magnitude Specifications: Units: dB; Apass: 1; Astop: 15

Ready

Current Filter Information
Structure: Direct-Form II, Second-Order Sections
Order: 6
Sections 3
Stable: Yes
Source: Designed

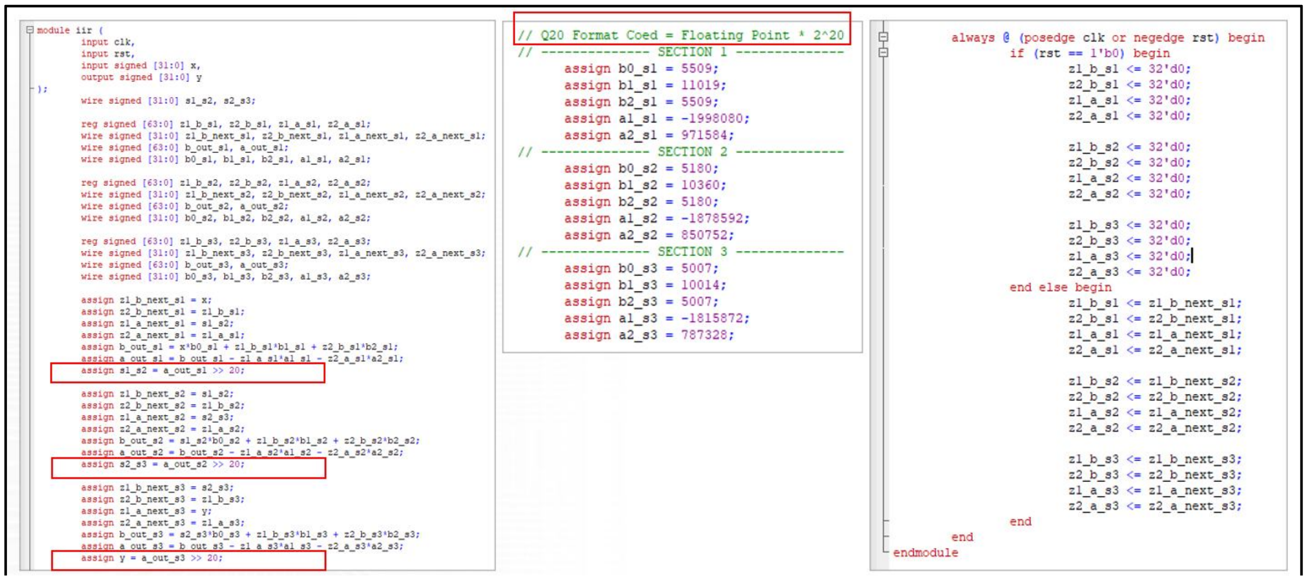
Filter Coefficients

```
--------------------------
Section #1
--------------------------
Numerator:
1
2
1
Denominator:
 1
-1.905533527326017217973230799543671309948
 0.926549887892354817608975281473249197006
Gain:
0.005254090141584445879108233867782473681
--------------------------
```

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: [✓ IIR Butterworth] FIR Equiripple
Filter Order: Specify order: 10 [✓ Minimum order]
Options: Match exactly: stopband
Frequency Specifications: Units: Hz; Fs: 48000; Fpass: 1000; Fstop: 1500
Magnitude Specifications: Units: dB; Apass: 1; Astop: 15

Designing Filter ... Done

Figure 10: IIR design by a filter designer‘*MATLAB fdatool*’.



```
module iir (
        input clk,
        input rst,
        input signed [31:0] x,
        output signed [31:0] y
);
        wire signed [31:0] s1_s2, s2_s3;

        reg signed [63:0] z1_b_s1, z2_b_s1, z1_a_s1, z2_a_s1;
        wire signed [31:0] z1_b_next_s1, z2_b_next_s1, z1_a_next_s1, z2_a_next_s1;
        wire signed [63:0] b_out_s1, a_out_s1;
        wire signed [31:0] b0_s1, b1_s1, b2_s1, a1_s1, a2_s1;

        reg signed [63:0] z1_b_s2, z2_b_s2, z1_a_s2, z2_a_s2;
        wire signed [31:0] z1_b_next_s2, z2_b_next_s2, z1_a_next_s2, z2_a_next_s2;
        wire signed [63:0] b_out_s2, a_out_s2;
        wire signed [31:0] b0_s2, b1_s2, b2_s2, a1_s2, a2_s2;

        reg signed [63:0] z1_b_s3, z2_b_s3, z1_a_s3, z2_a_s3;
        wire signed [31:0] z1_b_next_s3, z2_b_next_s3, z1_a_next_s3, z2_a_next_s3;
        wire signed [63:0] b_out_s3, a_out_s3;
        wire signed [31:0] b0_s3, b1_s3, b2_s3, a1_s3, a2_s3;

        assign z1_b_next_s1 = x;
        assign z2_b_next_s1 = z1_b_s1;
        assign z1_a_next_s1 = s1_s2;
        assign z2_a_next_s1 = z1_a_s1;
        assign b_out_s1 = x*b0_s1 + z1_b_s1*b1_s1 + z2_b_s1*b2_s1;
        assign a_out_s1 = b_out_s1 - z1_a_s1*a1_s1 - z2_a_s1*a2_s1;
        assign s1_s2 = a_out_s1 >> 20;

        assign z1_b_next_s2 = s1_s2;
        assign z2_b_next_s2 = z1_b_s2;
        assign z1_a_next_s2 = s2_s3;
        assign z2_a_next_s2 = z1_a_s2;
        assign b_out_s2 = s1_s2*b0_s2 + z1_b_s2*b1_s2 + z2_b_s2*b2_s2;
        assign a_out_s2 = b_out_s2 - z1_a_s2*a1_s2 - z2_a_s2*a2_s2;
        assign s2_s3 = a_out_s2 >> 20;

        assign z1_b_next_s3 = s2_s3;
        assign z2_b_next_s3 = z1_b_s3;
        assign z1_a_next_s3 = y;
        assign z2_a_next_s3 = z1_a_s3;
        assign b_out_s3 = s2_s3*b0_s3 + z1_b_s3*b1_s3 + z2_b_s3*b2_s3;
        assign a_out_s3 = b_out_s3 - z1_a_s3*a1_s3 - z2_a_s3*a2_s3;
        assign y = a_out_s3 >> 20;
```

```
// Q20 Format Coed = Floating Point * 2^20
// -------------- SECTION 1 --------------
        assign b0_s1 = 5509;
        assign b1_s1 = 11019;
        assign b2_s1 = 5509;
        assign a1_s1 = -1998080;
        assign a2_s1 = 971584;
// -------------- SECTION 2 --------------
        assign b0_s2 = 5180;
        assign b1_s2 = 10360;
        assign b2_s2 = 5180;
        assign a1_s2 = -1878592;
        assign a2_s2 = 850752;
// -------------- SECTION 3 --------------
        assign b0_s3 = 5007;
        assign b1_s3 = 10014;
        assign b2_s3 = 5007;
        assign a1_s3 = -1815872;
        assign a2_s3 = 787328;
```

```
        always @ (posedge clk or negedge rst) begin
            if (rst == 1'b0) begin
                z1_b_s1 <= 32'd0;
                z2_b_s1 <= 32'd0;
                z1_a_s1 <= 32'd0;
                z2_a_s1 <= 32'd0;

                z1_b_s2 <= 32'd0;
                z2_b_s2 <= 32'd0;
                z1_a_s2 <= 32'd0;
                z2_a_s2 <= 32'd0;

                z1_b_s3 <= 32'd0;
                z2_b_s3 <= 32'd0;
                z1_a_s3 <= 32'd0;
                z2_a_s3 <= 32'd0;
            end else begin
                z1_b_s1 <= z1_b_next_s1;
                z2_b_s1 <= z2_b_next_s1;
                z1_a_s1 <= z1_a_next_s1;
                z2_a_s1 <= z2_a_next_s1;

                z1_b_s2 <= z1_b_next_s2;
                z2_b_s2 <= z2_b_next_s2;
                z1_a_s2 <= z1_a_next_s2;
                z2_a_s2 <= z2_a_next_s2;

                z1_b_s3 <= z1_b_next_s3;
                z2_b_s3 <= z2_b_next_s3;
                z1_a_s3 <= z1_a_next_s3;
                z2_a_s3 <= z2_a_next_s3;
            end
        end
endmodule
```

Figure11: IIR filter implementation via VERILOG HDL.

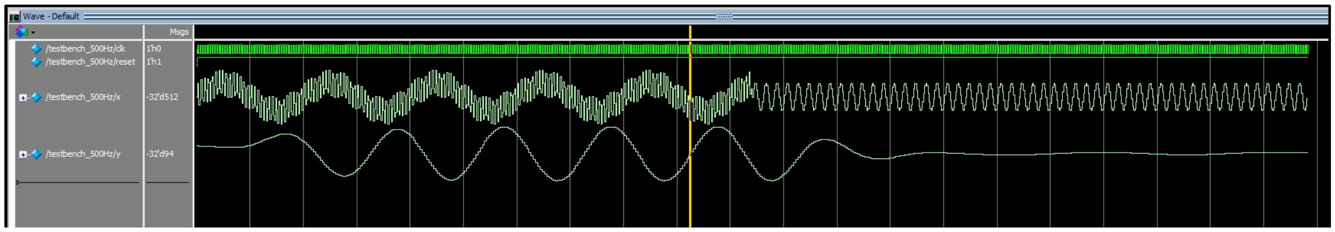


Figure 12: Wave form result of IIR processor.

5. CONCLUSION

The research has analyzed pipelined digital filter design. The MATLAB *fdatool* and scripts to filter design and calculation are shown clearly. In addition, those filter design parameters have been used in hardware implementation by Verilog HDL. A visual testbench is discussed to show the novel of proposed digital system design. All source codes and scripts are available upon reasonable requests.

ACKNOWLEDGEMENT

This work is supported byFPT University, Hanoi, Vietnam. We would like to thank Dr. Nguyen Manh Duc (G2Touch Company, Republic of Korea) for his useful suggestions, and Dr. Nguyen Ha Phong (Center of Machine Vision & Signal Analysis, University of Oulu, Finland) for his advices.

Conflict of Interest

On behalf of all authors, the corresponding author declares that there is no conflict of interest.

REFERENCES

1. Nguyen, D.M., Kim, S. “**The fog on Generalized teleportation by means of discrete-time quantum walks on N-lines and N-cycles**”, Modern Physics Letters B 33 (23), 1950270, 2019.
2. S. Chakraborty. "**Design and Realization of IIR Digital Band Stop Filter Using Modified Analog to Digital Mapping Technique**". Int. J. of Science, Engineering and Technology Research (IJSETR), vol. 2(3), 2013.
3. M. H. E. yousfiAlaou1, et. al. "**ECG denoising by EMD and EEMD improved with an adaptive RLS filter**". Int. J. of Advanced Trends in Computer Science and Engineering, vol. 9(3), 2020.
4. Nguyen, D.M., Kim, S. “**A novel construction for quantum stabilizer codes based on binary formalism**”. Int. J. of Modern Phys. B. vol. 33(8), 2020.
5. R. Dhannawat1, et. al. "**A New Faster, Better Pixels Weighted Don’t Care Filter for Image Denoising and Deblurring**". Int. J. of Advanced Trends in Computer Science and Engineering, vol. 9(2), 2020.
6. Hieu V.D.et. al. “**Design and verification of novel classical error control codes using VERILOG Hardware Description Language**”. Int. J. of Advanced Trends in Computer Science and Engineering, vol. 9(4), 2020.