The designs about LMS and NLMS adaptive filters based on Matlab and FPGA

Pingxiuqi Chen

Emial：chen.148@wright.edu

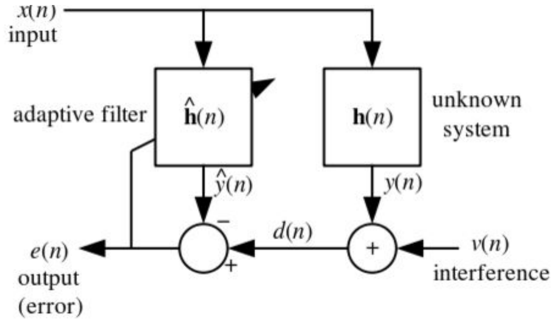
1/22/2017

1. Analysis:

Least mean square (LMS) and normalized least mean square (NLMS) adaptive filter are two kinds of special adaptive filters which combined LMS algorithm and NLMS algorithm with adaptive filters. LMS adaptive filter can mimic a desired filter by finding the filter coefficients (*w(n)*) that relate to producing the least mean square of error signal. LMS algorithm can make the filter be adapted on the error (*e(n)*) at the current time. NLMS algorithm is another widely used algorithm which is similar as LMS algorithm.

1.1. LMS adaptive filter

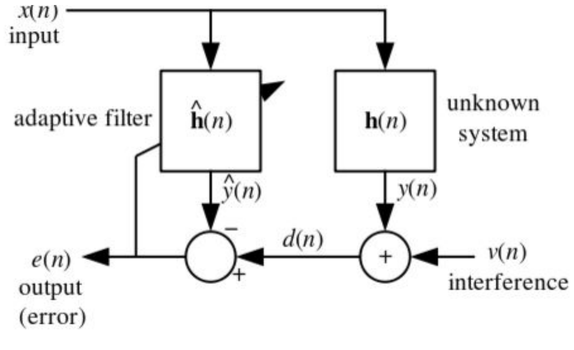
The structure of LMS adaptive filter is shown as follow [4]



LMS adaptive algorithm:
Input signal: x(n)=d(n)+noise
Desire signal: d(n)
Error signal: e(n)=d(n)-y(n)=d(n)-x'(n)w(n)=d(n)-w'(n)x(n)
Adaptive coefficient: w(n+1)=w(n)+2*μ*e(n)*x(n)
Output signal: y(n)=x'(n)w(n)=w'(n)x(n)
* From [2] we know μ is the step size parameter which is constant and does not depend on the input signal.

1.2. NLMS adaptive filter

The structure of NLMS adaptive filter is shown as follow [4], which is same as LMS. The only difference is NLMS adaptive filter use NLMS algorithm instead of LMS algorithm.



NLMS adaptive algorithm:
Input signal: x(n)=d(n)+noise
Desire signal: d(n)
Error signal: e(n)=d(n)-y(n)=d(n)-x'(n)w(n)=d(n)-w'(n)x(n)
Adaptive coefficient: w(n+1)=w(n)+μ*e(n)*x(n)
Step size parameter: μ(n)=1/(δ+x(n) . x'(n))
Output signal: y(n)=x'(n)w(n)=w'(n)x(n)
* From [2] we know δ is a small positive constant, used to avoid the division by zero. 0<μ<(2/λ_max), where λ_max is the largest variance of the input signal.

2. Solutions:

2.1 Question one:

First build a floating point MATLAB model for LMS and NLMS filter using reference [1] and without using any DSP toolbox commands. You can only use MATLAB basic commands.

Question analysis:

From reference [1], we know the desire signal is a sinusoid signal 1000 samples. So we need to build input signal first. The input signal is consisted by desire signal and noise. The input signal will be operated under adaptive and LMS algorithms. After several times of iterations, the output signal will be similar to desire signal. The only difference between LMS and NLMS algorithms is μ. The μ in LMS algorithm is constant and does not depend on the input signal, but in NLMS algorithm, μ will be changed when the input signal is changed.

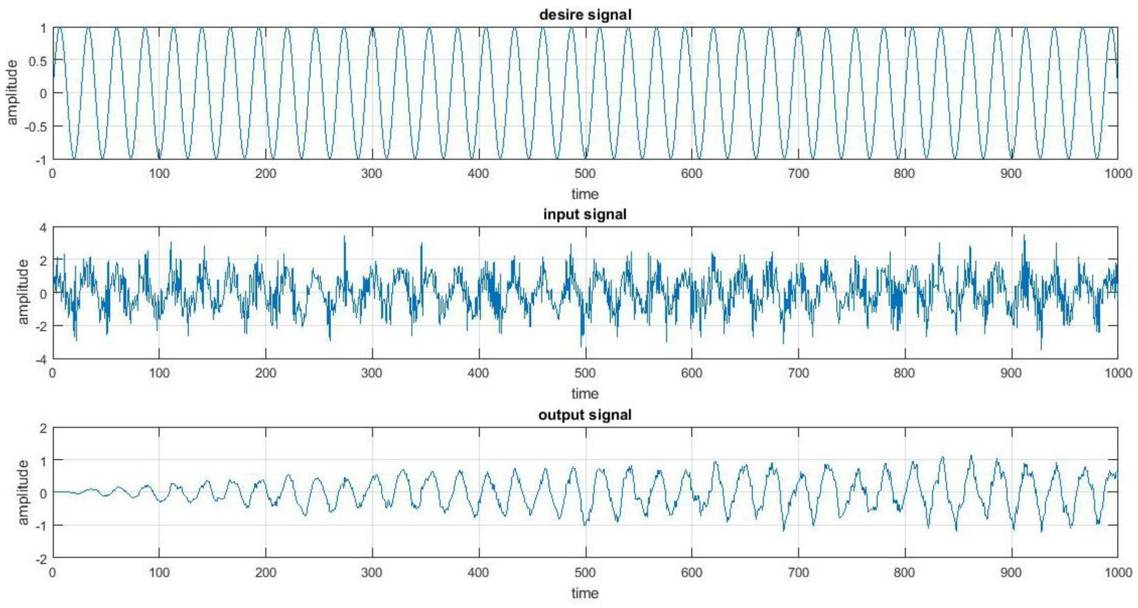
Code:

(1) LMS adaptive filter

```
%floting point matlab model for LMS filter

%by Pingxiuqi Chen

clear all; format short;

%desired signal

N=(1:1000)';

s=sin(0.075*pi*N);

%noise signal

v=0.8*randn(1000,1);%random noise part

ar=[1,1/2];                %autoregression coefficient

v1=filter(1,ar,v);    %noise signal, applied a 1-D digital filter

%noise-corrupted sinusoid xn

xn=s+v1;
```

```
%reference signal
ma=[1,-0.8,0.4,-0.2];
v2=filter(ma,1,v);
%expected output
d=s';
%step size
mu=rand()*(1/max(eig(xn'*xn)))
%LMS
sysorder=6;
w=zeros(6,1);
for n=sysorder:1000
    x=xn(n:-1:n-sysorder+1);
    y(n)=(w')*x;
    e(n)=d(n)-y(n);
    w=w+2*mu*e(n)*x ;
end
%plot(N,s,'b',N,xn,'r',N,y,'y')
%desire signal
subplot(3,1,1);
plot(N,s)
grid
ylabel('amplitude')
xlabel('time')
title('desire signal')
%input signal
subplot(3,1,2)
plot(N,xn)
grid
ylabel('amplitude')
xlabel('time')
```

```
title('input signal')
%output signal
subplot(3,1,3)
plot(N,y)
grid
ylabel('amplitude');
xlabel('time');
title('output signal');
```

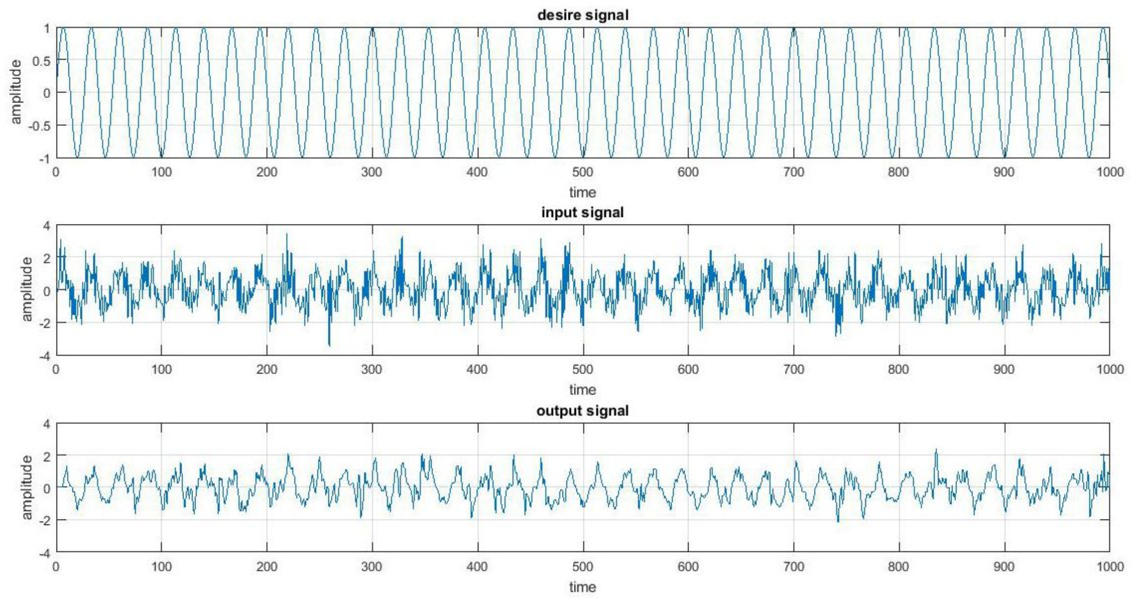


(2) NLMS adaptive filter

```
%floting point matlab model for NLMS filter
%by Pingxiuqi Chen
clear all; format short;
%desired signal
N=(1:1000)';
s=sin(0.075*pi*N);
%noise signal
v=0.8*randn(1000,1);%random noise part
ar=[1,1/2];                 %autoregression coefficient
```

```
v1=filter(1,ar,v);    %noise signal, applied a 1-D digital filter
%noise-corrupted sinusoid xn
xn=s+v1;
%reference signal
ma=[1,-0.8,0.4,-0.2];
v2=filter(ma,1,v);
%expected output
d=s';
%NLMS
sysorder=6;
w=zeros(6,1);
for n=sysorder:1000
        x=xn(n:-1:n-sysorder+1)
        mu=1/(0.01+(x'*x));
        y(n)=w'*x;
        e(n)=d(n)-y(n);
        w=w+mu*e(n)*x;
end
%desire signal
subplot(3,1,1);
plot(N,s)
grid
ylabel('amplitude')
xlabel('time')
title('desire signal')
%input signal
subplot(3,1,2)
plot(N,xn)
grid
ylabel('amplitude')
```

```
xlabel('time')
title('input signal')
%output signal
subplot(3,1,3)
plot(N,y)
grid
ylabel('amplitude');
xlabel('time');
title('output signal');
```



2.2 Question two:

Based on step 1, build a fixed point MATLAB model for reference [1] without using any DSP toolbox commands. Please use appropriate word lengths for different variables while making sure to keep 8-bit signed numbers at the interface and for the filter coefficients. You can only use MATLAB basic commands. The performance of the fixed point MATLAB model should be very close to that of the floating point model.

Question analysis:

From the solution of question one, we know all calculations are finished under floating point environment. Therefore all inputs, parameters, and outputs are floating pointing number. What we need to do is transfer all values to fixed point numbers.

If we use Matlab **'round'** function, we will change all signals to 0 value which is not correct. So I improved the desire and input signals' magnitude for 1000 times, and noise's magnitude 100 times. In question 2, we get to know we need to keep all interface signals and coefficients as 8-bit signed values. So we also need to use matlab

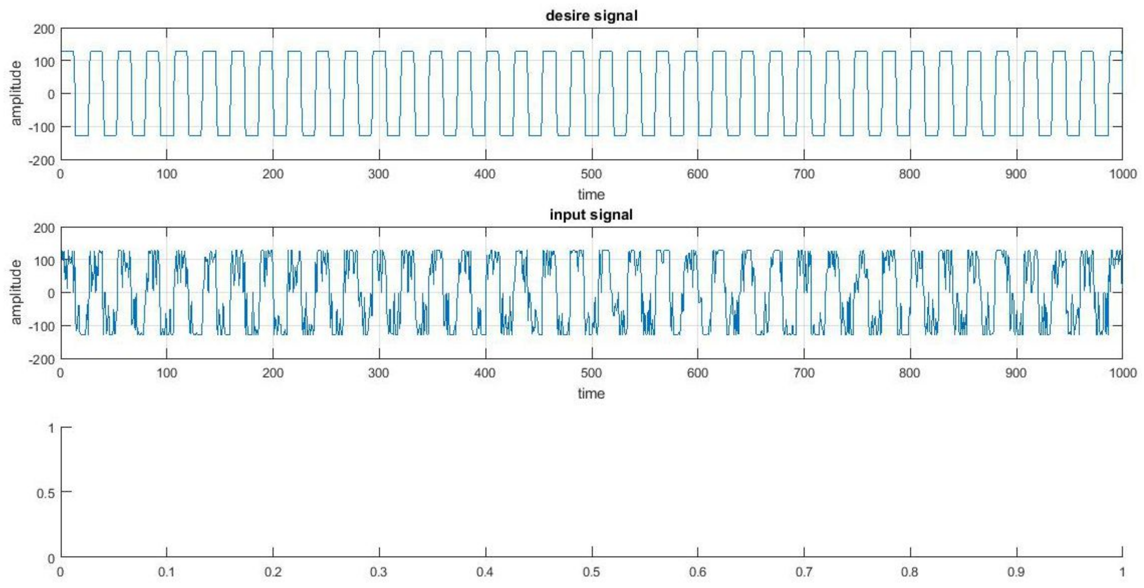
'int8' function for these values. So finally, all sin signals can be changed to square signals.

Problems I met:
(1). For this question, I just can get input and desire signals, I could't get output signals. I defined all interface signal and coefficients as 8-bit signed values. But when I run the code, there is error for *y(n)=int8(w'.*x).* In this equation, w and x are all 6x1 matrices, and all elements are integers. So I don't know how to solve this problem. Same problem also happened in NLMS adaptive filter code too.
(2). From the requirement, I know the noise I should use is AWGN noise. But after I applied matlab **'awgn'** function, I realized the waveform of awgn noise does not looks like a noise. So I still use the function **'randn'** to build the noise signal.

Code:
(1) LMS adaptive filter:

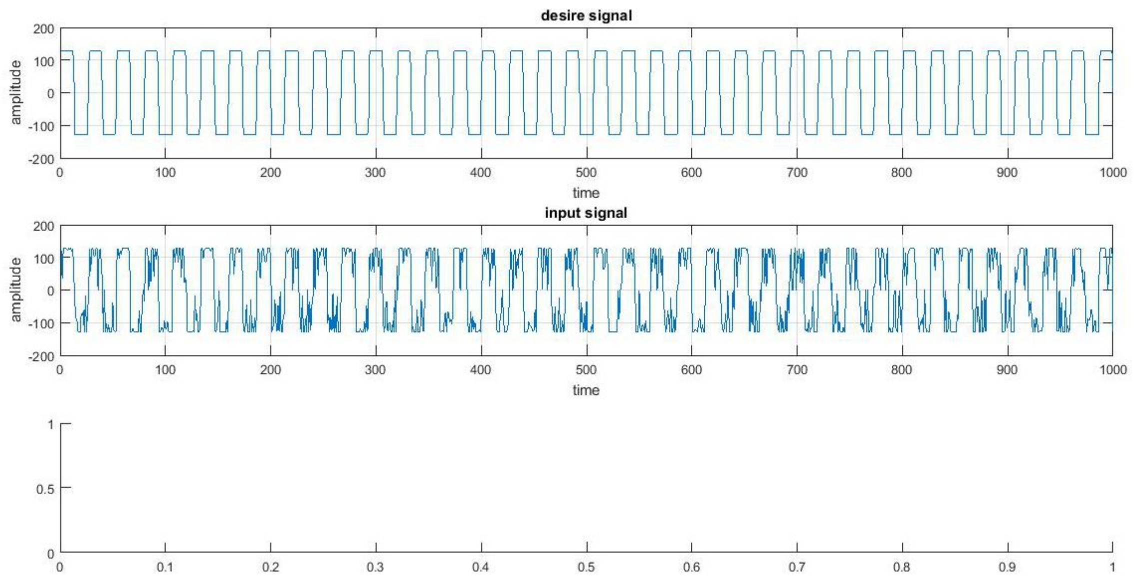
```
%fixed point matlab model for LMS filter

%by Pingxiuqi Chen

clear all;

%desired signal

N=(1:1000)';

s=int8(1000*sin(0.075*pi*N));

%noise signal

noise=int8(100*0.8*randn(1000,1));

%noise-corrupted input xn

xn=int8(s+noise);

%expected output

d=int8(s');

%step size

mu=int8(0.01*100);

%LMS

   sysorder=int8(6);

   w=int8(zeros(6,1));

   for n=sysorder:100

         x=xn(n:-1:n-sysorder+1)
```

```
        y(n)=int8(w'.*x);

        e(n)=int8(d(n)-y(n));

        w=int8(w+2*mu*e(n)*x);

  end


%desire signal

subplot(3,1,1);

plot(N,s)

grid

ylabel('amplitude')

xlabel('time')

title('desire signal')

%input signal

subplot(3,1,2)

plot(N,xn)

grid

ylabel('amplitude')

xlabel('time')

title('input signal')

%output signal

subplot(3,1,3)

plot(N,y)

grid

ylabel('amplitude');

xlabel('time');

title('output signal');
```



(2) NLMS code:

```
%floting point matlab model for NLMS filter

%by Pingxiuqi Chen

clear all; format short;

%desired signal

N=(1:1000)';

s=int8(1000*sin(0.075*pi*N));

%noise signal

v=int8(100*0.8*randn(1000,1));%random noise part

%noise-corrupted sinusoid x

xn=int8(s+v);

%expected output

d=int8(s');

%NLMS

sysorder=6;

w=zeros(6,1);

% for n=sysorder:1000

%         x=xn(n:-1:n-sysorder+1);
%         mu=1/(0.01+(x'.*x));
%         y(n)=w'.*x;
%         e(n)=d(n)-y(n);
%         w=w+mu*e(n)*x;
% end
%desire signal
subplot(3,1,1);
plot(N,s)
grid
ylabel('amplitude')
xlabel('time')
title('desire signal')
%input signal
subplot(3,1,2)
plot(N,xn)
grid
ylabel('amplitude')
xlabel('time')
title('input signal')
%output signal
subplot(3,1,3)
plot(N,y)
grid
ylabel('amplitude');
xlabel('time');
title('output signal');
```



2.3 Question three:

Based on step 2, design a Verilog RTL module for 6th order adaptive LMS filter in reference [1] and demonstrate the MODELSIM waveform matching similar functionality as reference [1]. Please do not use MATLAB coder though you can use the example as reference.

Question analysis:

For this question, I need to use verilog to design the LMS adaptive filter. From reference [2], we know the whole working process of filter can be described by 9 states FSM. The inputs of circuit are clk, reset, x(n), d(n),e(n),y(n). When clk signal comes to rising edge, all signals will be operated and shifted to next state. When reset signal equal to '1', the system comes to original state.

S0 is the initial state of the system, when new data is available, x(n) and d(n) will be shifted to next state S1, and x(n) will be converted in SM (sign-and-magnitude) form. In S1 state, the system will do the operation of y(n)=x'(n) *w(n). The result of y(n) will be sent to next state S2, which will convert y(n) to 2's complement form. All y(n)s will be added together which will happen in next state S3. For next state S4, the system will get the error e(n) between desire signals d(n) and y(n). After S4, we can choose to update coefficient w(n) or not. If we choose no, the system will go back to S0, if we choose yes, the system will go to next state S5 to do the operation of e*μ. After we get all values, the system will go to next state S6, which will finish the operation w(n+1)=w(n)+2*μ*e(n)*x(n). For next state S7, the w(n+1) will be converted to 2's complement form. In next state S8, the w(n) will be updated. The last state S9 will convert w(n+1) to SM form for multiplication.

Problems I met:

I'm stuck at how to convert floating point number to fixed point number. So I just figure out part of the code.

Code:

```
module adaptive_filter(clk, reset, xin, din, eout, yout);
parameter N1 = 8, N2 = 16, L = 6;//L is the order of LMS adaptive filter

input clk;
input reset;
input [N1-1:0] xin, din;
output [N2-1:0] eout, yout;

reg [N1-1:0] x0,x1,x2,x3,x4,x5,f0,f1,f2,f3,f4,f5,d,mu;
wire [N2-1:0] p0,p1,p2,p3,p4,p5,xmu0,xmu1,xmu2,xmu3,xmu4,xmu5;
wire [N2-1:0] y, sxty, e, sxtd;
wire [N2-1:0] sum;

always @(posedge clk)
begin
if(reset)
begin
x0 <= 0;
x1 <= 0;
x2 <= 0;
x3 <= 0;
x4 <= 0;
x5 <= 0;
f0 <=0;
f1 <=0;
f2 <=0;
f3 <=0;
f4 <=0;
f5 <=0;
d <=0;
```

```
end
else begin
d <= din;
x0 <= xin;
x1 <= x0;
x2 <= x1;
x3 <= x2;
x4 <= x3;
x5 <= x4;
f0 <= f0 + xmu0[15:7];
f1 <= f1 + xmu1[15:7];
f2 <= f2 + xmu2[15:7];
f3 <= f3 + xmu3[15:7];
f4 <= f4 + xmu4[15:7];
f5 <= f5 + xmu5[15:7];
end
end
(This part should describe the algorithm)


assign yout = y;
assign eout = e;


endmodule
```

3. Reference:

[1]https://www.mathworks.com/help/dsp/ug/enhance-a-signal-using-lms-and-normalized-lms-algorithms.html#brdowiw

[2] I. Homana, I. Muresan, M. Topa, and C. Contan, "FPGA implementation of LMS and NLMS adaptive filters for acoustic echo cancellation," ACTA Technica Napocensis-Electron. Telecommun., vol. 52, no. 4, pp. 13–16, 2011.

[3]https://www.mathworks.com/examples/matlab-hdl-coder/mw/hdlcoder_productmlhdlc_tutorial_dsp_lms_nc-lms-filter-noise-cancellation

[4]https://en.wikipedia.org/wiki/Least_mean_squares_filter