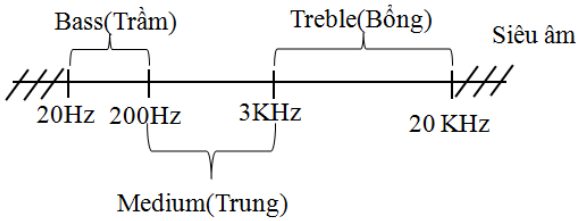
BÀI 1: PHÂN TÍCH PHỔ AUDIO TRÊN MATLAB

1. Tổng quan về Audio

1.1. Âm tần

Âm tần dược định nghĩa là khoảng tần số âm thanh mà tai người cảm nhận được 20Hz – 20 KHz. Phổ âm tần được chia làm 3 vùng Bass, Medium và Trebel:



Tai người cảm nhận cường độ âm thanh phụ thuộc vào cả biên độ và tần số, nghe thính nhất ở vùng từ 1 KHz – 3 KHz. Cường độ âm thanh lớn nhất mà tai người có thể chịu được là 120 dB.

1.2. Âm thanh số

Sampling rate: Tùy vào yêu cầu về chất lượng của tín hiệu mà ta có các tốc độ lấy mẫu khác nhau. Đối với chuẩn CD f_S = 44.1 KHz, DVD f_S = 48 KHz còn telephone là 8 KHz.

Bit depth: Số bit dùng để biểu diễn một mẫu.VD: Một tín hiệu âm thanh được lấy mẫu bởi một mạch ADC 16 bit với f_S = 48 KHz sẽ có sampling rate là 48 KHz và bit depth là 16 bit.

Uncompressed: Các định dạng tín hiệu âm thanh số không nén: .wav, .aiff. Tuy nhiên dung lượng của các file này khá lớn do đó gây khó khăn cho việc lưu trữ và xử lý, dẫn đến yêu cầu nén các tín hiệu âm thanh số.

Compressed: Có 2 loại nén là nén không mất dữ liệu (lossless) và nén mất dữ liệu (lossy).

Để đọc file .wav trong Matlab ta sử dụng lệnh wavread:

```
[data sr nbits] = wavread('sound.wav');
```

Trong đó sr là sampling rate, nbits là bit depth. Nếu file sound.wav là âm thanh 1 kênh (mono) thì data sẽ là mảng 1 chiều, nếu là âm thanh 2 kênh (stereo) thì data sẽ là mảng 2 chiều.

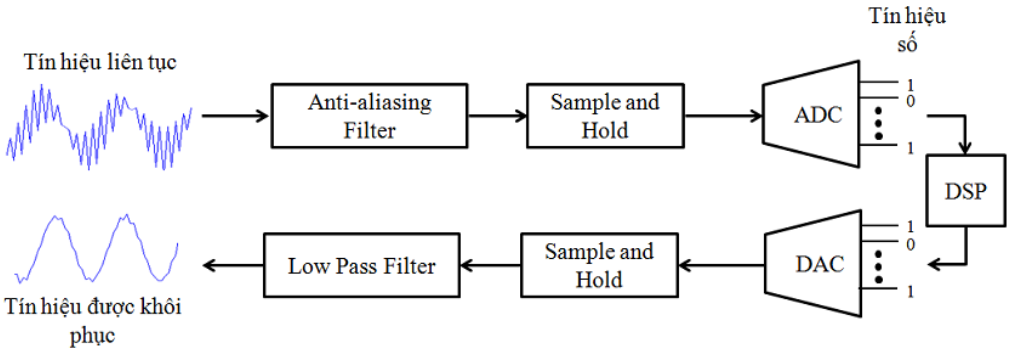
Để phát lại tín hiệu âm thanh số trong Matlab ta sử dụng lệnh sound:

```
sound(data,sr);
```

2. Phân tích phổ audio

Hình dưới là sơ đồ lấy mẫu và phát lại một tín hiệu audio đơn giản.Sau khi đã được chuyển sang tín hiệu số, tín hiệu audio sẽ được lưu lại, phân tích và xử lý bởi khối DSP. Sau đó ta có thể phát lại tín hiệu bằng cách khôi phục lại tín hiệu tương tự thông qua mạch DAC.

Lý thuyết về lấy mẫu và biến đổi FFT sinh viên đã được học trong môn DSP. Do đó, trọng tâm của bài này sẽ chỉ đi sâu vào việc phân tích phổ sử dụng biến đổi FFT trên Matlab.



Do biến đổi fft giả sử tín hiệu đưa vào là tuần hoàn, nên thông thường để tránh hiện tượng rò rỉ phổ ta phải đưa tín hiệu lấy mẫu một qua một cửa sổ (window) trước khi sử dụng biến đổi fft.

Để phân tích phổ fft trong Matlab ta sử dụng hàm fft(signal, N). Trong đó N là số điểm fft. Với f = fft(signal, N):

- Bin 1: (f(1))là thành phần DC / trung bình tín hiệu.
- Bin 2:(f(2)) là năng lượng tại tần số Fs/N .
- Bin K: lànăng lượng tạitần số (k-1)*Fs/N.
- Bin N/2 + 1:là năng lượng tại tần tố Nyquist Fs/2.
- Bin k > N/2 + 1:là ảnh (tần số âm) của tần số thực.
- Để coi phổ biên độ của tín hiệu theo giai decibel ta dùng lệnh sau: 20*log10(abs(f)).

Để chuyển tín hiệu từ miền tần số về miền thời gian ta dùng niến đổi fft ngược: signal_new = ifft(f, N_signal). Với N_signal là chiều dài của tín hiệu gốc.

Nhóm: ……..

- **……MSSV:……..**
- **……MSSV:……..**

----------o0o----------

BÀI 1: PHÂN TÍCH PHỔ AUDIO TRÊN MATLAB

Mục đích:

- Ứng dụng Matlab phân tích phổ tín hiệu âm thanh.

Bài 1: Phân tích phổ một tín hiệu sin đơn giản:

a) Đoạn script sau tạo một tín hiệu hình sin và dùng hàm fft để chuyển tín hiệu này sang miền tần số:

```
clear
clc

signalLength = 10240;
n=0:0.1:signalLength;
A=1;
xs  = A*sin(2*pi*0.4*n);

N = 10240;
xsFFT= abs(fft(xs,N));
y = 20*log10(xsFFT);
y = y(1:N/2);
f = (0:length(y)-1);
figure
plot(f,y);
axis([1 (N/2+1) -10 50])
grid
```

Sinh viên cho biết tần số lấy mẫu, biên độ và tần số của tín hiệu vào:

……

Chạy file script và quan xát tín hiệu. Cho biết bin có năng lượng cao nhất và giải thích:

……

……

b) Đoạn script sau được sửa lại để dùng cửa sổ hanning trước khi biến đổi fft. Chạy file script, so sánh với phổ tần của câu a và giảithích:

```
clear
clc

signalLength = 10240;
n=0:0.1:signalLength;
A=1;
xs  = A*sin(2*pi*0.4*n);

N = 10240;
window = hanning(N);
xsWindow = xs(1:N).*window';
xsFFT= abs(fft(xsWindow,N));
y = 20*log10(xsFFT);
y = y(1:N/2);
f = (0:length(y)-1);
figure
plot(f,y);
axis([1 (N/2+1) -10 50])
```

Giải thích:

……

……

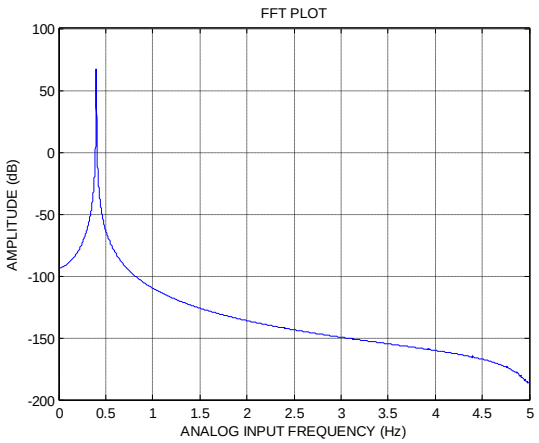
Bài 2: Phân tích phổ của 1 bài nhạc trên Matlab:

a) Dựa vào file script đã cho ở bài 1.b, tiến hành viết một file function trong Matlab để vẽ phổ tần tín hiệu:

```
function []=plotfft(signal,fs,N)
```

- signal: Tín hiệu vào.
- fs: Tần số lấy mẫu.
- N: Số điểm FFT

b) Viết file script sử dụng hàm vừa tạo để vẽ lại phổ tần của tín hiệu ở bài 1. Kết quả xuất ra phải có dạng như sau:



c) Lần lượt load file ‘hootie.wav’ và ‘road.wav’ rồi sử dụng hàm tạo ở câu a để vẽ phổ của các tín hiệu này, quan sát và so sánh phổ của tín hiệu:

……

……

BÀI 2: THỰC HIỆN LỌC AUDIO TRÊN MATLAB

3. Mạch lọc hữu hạn (FIR)

Công thức tổng quát:

$$y(n) = b_0x(n) + b_1x(n-1) + \ldots + b_Nx(n-N) = \Sigma_{i=0}^{N} b_i x(n-i)$$

Ưu điểm:

- Đơn giản, dễ thiết kế.
- Không có phần hồi tiếp nên mạch lọc luôn ổn định.
- Dễ thiết kế các mạch lọc tuyến tính pha nên có thể được ứng dụng trong các hệ thống yêu cầu cao về độ tuyến tính pha như truyền dữ liệu, lọc chéo …

Khuyết điểm:

- Để đạt được cùng độ dốc như mạch lọc IIR thì mạch lọc FIR cần bậc lọc cao hơn.

4. Mạch lọc vô hạn (IIR)

Công thức tổng quát:

$$\begin{aligned} y(n) &= b_0x(n) + b_1x(n-1) + \ldots + b_Nx(n-N) \\ &\quad - a_1y(n-1) - a_2y(n-2) - \ldots - a_My(n-M) \\ &= \Sigma_{i=0}^{N} b_i x(n-i) - \Sigma_{j=1}^{M} a_j y(n-j) \end{aligned}$$

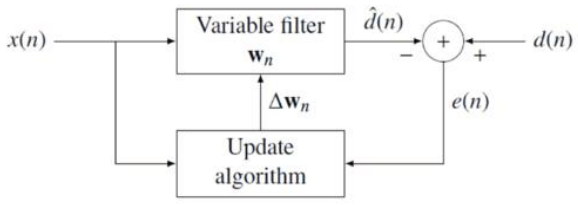
Ưu điểm:

- Để đạt được cùng độ dốc như mạch lọc FIR thì mạch lọc IIR cần bậc lọc thấp hơn.

Khuyết điểm:

- Phức tạp, khó thiết kế. Cần sử dụng các kỹ thuật như biến đổi lưỡng tính để đảm bảo tính ổn định của mạch lọc.
- Có phần hồi tiếp nên cần cẩn thận khi thiết kế để mạch lọc ổn định.
- Mạch có pha phi tuyến nên khó ứng dụng trong các hệ thống yêu cầu tuyến tính pha.

5. Mạch lọc thích nghi (Adaptive Filter)



Mạch lọc thích nghi là các mạch lọc có khả năng tự điều chỉnh các thông số trong hàm chuyển của mình dựa trên một thuật toán tối ưu nhất định. Quá trình hoạt động của một mạch lọc thích nghi như sau:

- Tín hiệu x(n) được đưa qua mạch lọc với các hệ số (w_n) khởi tạo. Tín hiệu ra khỏi mạch lọc là $\hat{d}(n)$.

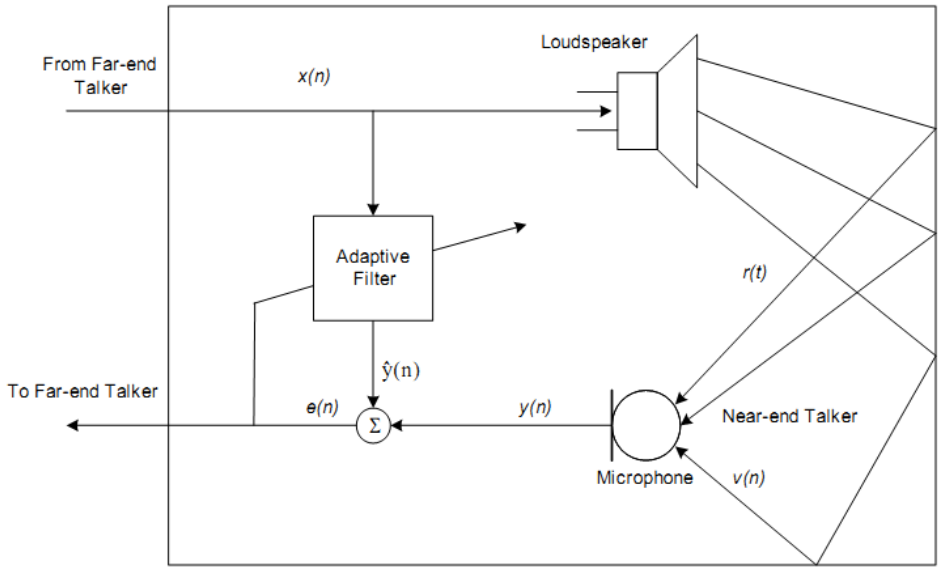
- Tín hiệu này sau đó được trừ đi bới tín hiệu mong muốn (d(n)). Sai số giữa hai tín hiệu này được gọi là tín hiệu lỗi e(n).
- Tín hiệu lỗi e(n) và tín hiệu vào x(n) sẽ được đưa vào thuật toán tối ưu để tính lại các hệ số (w_n) và cập nhật lại các hệ số này trong mạch lọc.
- Cứ tiếp tục như vậy, nếu điều kiện hội tụ của thuật toán tối ưu thỏa thì tín hiệu ra khỏi mạch lọc $\hat{d}(n)$ sẽ gần như tương đương với tín hiệu mong muốn d(n). Hay nói cách khác, tín hiệu lỗi e(n) sẽ tiến gần tới 0.

Ưu điểm:

- Linh động, được ứng dụng nhiều trong các mạch khử nhiễu trắng, khử echo trong micro (như mô tả ở hình dưới).

Khuyết điểm:

- Phức tạp, khó thiết kế. Tốn nhiều tài nguyên hơn các mạch lọc thông thường.
- Cần thời gian để mạch lọc hội tụ.



Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 2: THỰC HIỆN LỌC AUDIO TRÊN MATLAB

Mục đích:

- Thực hiện các loại mạch lọc khác nhau trên Matlab và ứng dụng vào lọc các dải tần của phổ âm thanh.

Bài 1: Thiết kế mạch lọc FIR, bậc N = 30, ứng với các loại cửa sổ và tần số cắt khác nhau để lọc file 'hootie.wav'. Sau đó dùng hàm plotfft ở bài 1 để quan sát phổ ra ở 2 kênh trái phải của tín hiệu này trước và sau khi qua mạch lọc.

a) Mạch lọc thấp qua tần số cắt 500Hz sử dụng cửa sổ tam giác và hanning.
b) Mạch lọc dải qua tần số cắt 1KHz – 3KHz sử dụng cửa sổ hanning và blackman.
c) Mạch lọc cao qua tần số cắt 4KHz sử dụng cửa sổ blackman và hamming.

Bài 2: Thiết kế mạch lọc chebyshev và butterworth với R_p = 0.5dB, R_s = 10dB để lọc file 'hootie.wav'. Sử dụng hàm plotfft để vẽ phổ pha của tín trước và sau khi lọc. Quan sát, so sánh kết quả so với bài 1 và giải thích

a) Mạch lọc thấp qua tần số cắt 500Hz.
b) Mạch lọc dải qua tần số cắt 1KHz – 3KHz.
c) Mạch lọc cao qua tần số cắt 4KHz.

Bài 3: Đoạn code sau tạo tín hiệu mới (x) bằng cách cộng nhiễu trắng vào tín hiệu gốc (d). Sau đó tín hiệu này được đưa qua một mạch lọc thích nghi để loại bớt nhiễu trắng. Sinh viên tiến thay đổi giá trị của "numberOftaps" với các giá trị: 5, 25, 50.

```
clc
clear

[d fs] = wavread('hootie.wav');

% Signal generation
noise = zeros(size(d));
numberOfChannels = min(size(d));
for i = 1:numberOfChannels
noise(:,i) = 0.25*sqrt(3)*(rand(length(d),1)-0.5);
end
x = d + noise;

% Adaptive Filter
u = 0.01;
y = zeros(size(d));
signaLength = length(x);
numberOfTaps = 5;
w = zeros(numberOfTaps,signaLength,numberOfChannels);
numberOfItterations = 1;
for j=1:numberOfItterations
for channel = 1:numberOfChannels
for i=numberOfTaps:length(x)
%Filter signal with FIR adaptive filter
            y(i,channel) = w(:,i,channel)'*x(i-
numberOfTaps+1:i,channel);

%Calculate error between desired signal and filtered signal
            e = d(i,channel) - y(i,channel);

%Update algorithm
            w(:,i+1,channel) = w(:,i,channel) + u*e*x(i-
numberOfTaps+1:i,channel);
end
end
end

%Plot weight of the filter
w = w(:,numberOfTaps:signaLength,:);
n = 1:signaLength-numberOfTaps + 1;
length(n);
length(w(1,:,1));
figure
plot(n,w(1,:,1))

%Play back the signals
sound(d,fs)
sound(x,fs)
sound(y,fs)
```

BÀI 3: XỬ LÝ ẢNH TRÊN MATLAB

Bài thực hành này giúp sinh viên ôn lại một số kiến thức về xử lý ảnh trên Matlab đã học qua ở môn thực hành Matlab & DSP và viết lại hàm lọc thay vì dùng hàm có sẵn trên Matlab để có thể ứng dụng lên FPGA.

Do phần lý thuyết ở môn thực hành trước và lý thuyết trên lớp đã khá đầy đủ, nên không cần phải nhắc lại ở bài thực hành này. Sinh viên cần đọc lại các khái niệm về histogram, lọc ảnh, tăng cường màu cho ảnh (bài 5B năm 2013 có một số thay đổi so với năm 2012).

Ở đây sẽ chỉ nhắc lại 2 lưu ý nhỏ khi xử lý ảnh trên Matlab:

1. Xử dụng hàm imread để đọc file ảnh vào Matlab. Nhưng tốt nhất luôn chuyển ảnh qua double ngay sau khi load. VD: image = im2double(imread('imagePath')). Nhằm tránh xảy ra lỗi khi dùng hàm imshow để vẽ lại hình trên Matlab.
2. Ảnh trắng đen là ảnh 2 chiều. Ảnh màu là ảnh 3 chiều và chiều thứ 3 có 3 giá trị tương ứng cho 3 màu R, G, B.

Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 3: XỬ LÝ ẢNH TRÊN MATLAB

Mục đích:

- Phân tích ảnh và tăng cường màu cho ảnh.
- Viết hàm lọc ảnh trên Matlab.

Bài 1: Phân tích histogram và tăng cường ảnh:

a) Sử dụng hàm imhist() để hiển thị Histogram cho 2 ảnh dark.png và bright.png. Nhận xét về độ sáng tối của 2 ảnh.

.....

.....

b) Chỉ sử dụng vòng lặp for(), viết hàm histogram() có chức năng hiển thị Histogram của ảnh GRAY (GRAY có thể là ảnh 8-bit hoặc 16-bit grayscale).

```
function [count , bin] = histogram(GRAY)
```

`count`: vector trục dọc của histogram

`bin`: vector trục ngang của histogram

c) Xây dựng hàm tăng cường màu và kiểm tra với các giá trị K = 0, 0.5, 1, 2:

```
function Img_out = satu(Img, K)
```

Gợi ý: Để tăng cường màu cho ảnh, ta sử dụng công thức sau:

$$\begin{cases} R' = (0.299 + 0.701 * K) * R + \big(0.587 * (1-K)\big) * G + \big(0.114 * (1-K)\big) * B \\ G' = \big(0.299 * (1-K)\big) * R + (0.587 + 0.413 * K) * G + \big(0.114 * (1-K)\big) * B \\ B' = \big(0.299 * (1-K)\big) * R + \big(0.587 * (1-K)\big) * G + (0.114 + 0.886 * K) * B \end{cases}$$

Bài 2: Viết hàm để lọc ảnh sử dụng các ma trận sau (không dùng hàm có sẵn trong Matlab) theo phương pháp **same**. So sánh kết quả đạt được so với khi dùng hàm conv2 của Matlab.

```
function [newImage] = imageFilter(image,matrix)
```

1.
```
[0,   0.2,  0;
 0.2, 0.2,  0.2;
 0,   0.2,  0]
```
2.
```
[0,   0,    1/13, 0,    0;
 0,   1/13, 1/13, 1/13, 0;
 1/13,1/13, 1/13, 1/13, 1/13;
 0,   1/13, 1/13, 1/13, 0;
 0,   0,    1/13, 0,    0]
```
3.
```
[1/9, 1/9, 1/9;
 1/9, 1/9, 1/9;
 1/9, 1/9, 1/9]
```

BÀI 4+5+6:XỬ LÝ TÍN HIỆU ÂM THANH TRÊN FPGA

Bài Lab được thực hiện nhằm giúp sinh viên hiểu và thực hiện một số quy trình, thuật toán xử lý âm thanh đơn giản sử dụng bộ Audio CODEC trên board DE2. Sinh viên sẽ thiết kế mạch logic để nhận dữ liệu vào từ Microphone thông qua bộ Audio CODEC sau đó sẽ xử lý tín hiệu nhận được và đưa kết quả xử lý ra loa. Thiết bị cần thiết: Board DE2, Microphone và Loa (Headphone).

*__Chú ý__: Sinh viên **đem theo MicroPhone** và **HeadPhone** cho các bài thực tập.

Các yêu cầu cần đạt được sau 3 bài Lab:

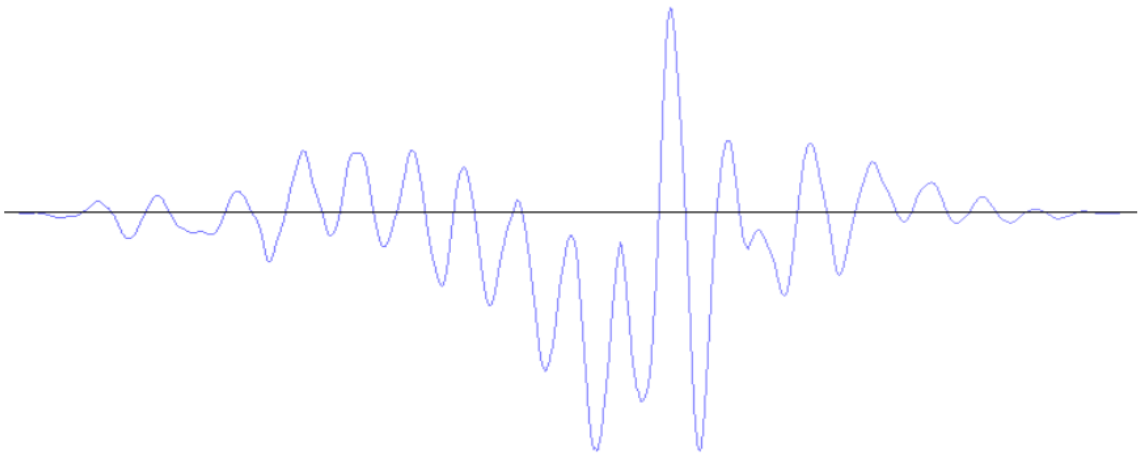
-Hiểu được tổng quan quá xử lý âm thanh.

-Biết cách sử dụng bộ Audio CODEC và board DE2 trong xử lý âm thanh.

-Xây dựng được các thành phần logic trong việc xử lý, thu nhận dữ liệu, lọc nhiễu, xuất âm thanh ra loa…

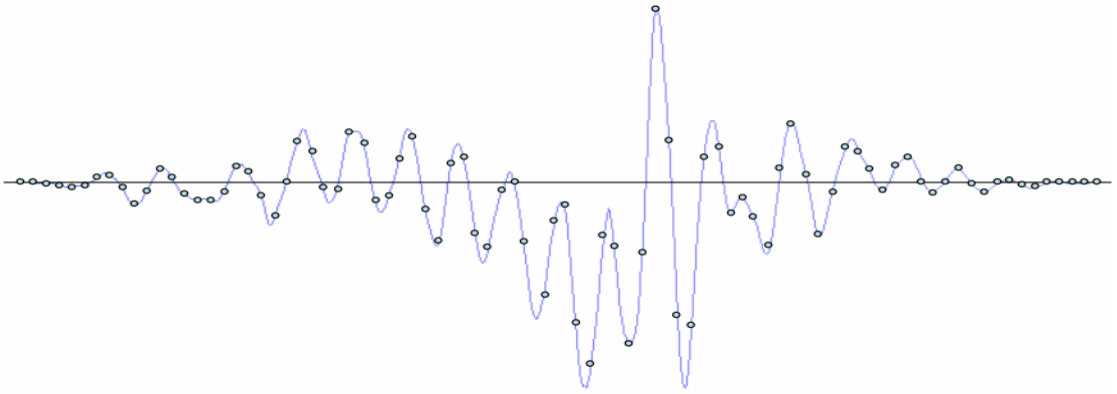
Lý thuyết

Âm thanh (tiếng nói, tiếng nhạc..) là tín hiệu có tần số và biên độ thay đổi liên tục liên tục theo thời gian. Hình 1 minh họa một tín hiệu âm thanh.



Hình 1: Tín hiệu âm thanh

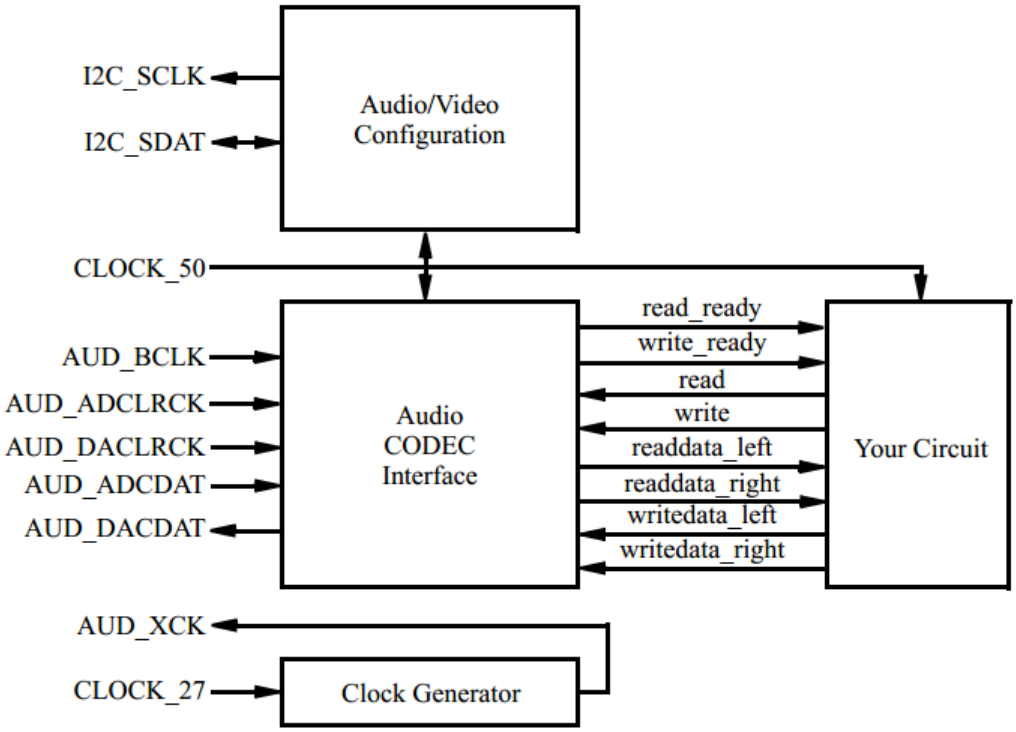
Đây là tín hiệu tương tự, liên tục theo thời gian. Trong các hệ thống số, nó được lưu trữ dưới dạng các mẫu rời rạc theo một khoảng thời gian nhất định gọi là quá trình lấy mẫu. Quá trình xử lý âm thanh được thực hiện trên các mẫu rời rạc này. Hình 2 minh họa việc lấy mẫu của một tín hiệu âm thanh.



Hình 2: Lấy mẫu tín hiệu âm thanh

Bộ Audio CODEC trên board DE2 có thể lấy mẫu âm thanh từ micro và truyền cho các mạch xử lý âm thanh. Mặc định, nó lấy mẫu ở tần số 48KHz (48000 mẫu/giây), tần số lấy mẫu này là đủ để biểu diễn chính xác các tín hiệu âm thanh trong dải nghe được.

Trong các bài thực hành này, chúng ta sẽ tạo một vài thiết kế để lấy dữ liệu từ micro thông qua bộ Audio CODEC trên board DE2, các dữ liệu này được ghi lại, xử lý và xuất tín hiệu ra loa. Hệ thống đơn giản giúp ghi và phát âm thanh được cho sẵn và được minh họa ở hình 3, bao gồm: Bộ phát tần số (Clock Generator), giao diện bộ giải mã âm thanh (Audio CODEC Interface), bộ cấu hình Audio/Video (Audio/Video Configuration).



Hình 3: Hệ thống xử lý âm thanh đơn giản trên DE2

Các tín hiệu bên trái là các input và output của hệ thống. Các port I/O này nhằm để cung cấp clock cho hệ thống, kết nối với bộ Audio CODEC, kết nối các thiết bị ngoại vi trên board DE2.
Các tín hiệu ở giữa nhằm giao tiếp giữa các mạch mà ta thiết kế với giao diện Audio CODEC. Các tín hiệu này cho phép ta ghi ghi dữ liệu từ micro và xuất dữ liệu ra loa.

Hệ thống hoạt động như sau: Khi reset, bộ cấu hình Audio/Video tự tạo các tín hiệu để cấu hình và điều khiển bộ Audio CODEC để bắt đầu lấy mẫu các tín hiệu vào từ micro tại tần số 48KHz, và tạo tín hiệu ra tại cùng tần số lấy mẫu. Khi quá trình tự động khởi tạo hoàn thành, bộ Audio CODEC bắt đầu đọc dữ liệu từ mico và xuất dữ liệu tới bộ Audio CODEC Interace. Trong quá trình nhận, các mẫu được lưu trữ trong bộ đệm 128 phần tử của bộ Audio CODEC Interace. Phần tử đầu tiên luôn luôn có sẵn tại ***readdata_left*** và ***readdata right*** khi tín hiệu ***read_ready*** được kích hoạt. Các dữ liệu tiếp theo được đọc bằng cách bật tín hiệu ***read*** trong khi tín hiệu ***read_ready*** vẫn tích cực. Khi đó mẫu hiện tại được thay thế bằng mẫu tiếp theo trong 1 hay vài chu kì tiếp theo.

Quá trình xuất tín hiệu âm thanh ra loa được thực hiện tương tự như trên. Hệ thống hoạt động dựa vào tín hiệu ***write_ready***. Khi nó được bật, hệ thống sẽ ghi các mẫu tín hiệu tới bộ Audio CODEC Interface tại ***writedata_left*** và ***writedata_right*** với điều kiện tín hiệu ***write*** được bật. Quá trình này lưu trữ các mẫu vào bộ đệm nằm bên trong bộ Audio CODEC Interface và sau đó sẽ được gửi ra loa.

Nhiệm vụ của các bài thực hành là thiết kế **Your Circuit** (trên hệ thống ở hình 3) để xử lý tín hiệu âm thanh lấy từ micro (hay từ máy tính) sau đó xuất kết quả ra loa. Có 3 bài Lab tương ứng với 3 buổi thực tập trên lớp bao gồm: Bài 4, bài 5 và bài 6.

Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 4: XÂY DỰNG HỆ THỐNG ÂM THANH TRÊN DE2

Câu 1:Xây dựng hệ thống nhận tiếng nói từ micro và phát ra loa (headphone)

- Bài thực tập sử dụng các module có sẵn trong thư mục **DesignFiles**.
- Copy các file sang thư mục thực tập, đổi tên file *YourExercise.v* thành *Bai4Cau1.v.*
- Dựa vào sơ đồ hệ thống ở phần lý thuyết, xây dựng hệ thống xử lý âm thanh đơn giản bằng cách hoàn thành Your Circuit trong file *Bai4Cau1.v.*
- Cấu hình chân cho FPGA dựa trên file *DE2_pin_assignments.csv.*
- Biên dịch bằng Quartus và nạp lên kit DE2
- Kết nối micro vào cổng MIC và headphone vào LINE OUT. Reset mạch và nhận xét kết quả thu được? (Âm thanh tốt, giống thực tế, có nhiễu….)

……

……

Câu2: Tạo nguồn nhiễu vào tín hiệu âm thanh (chuẩn bị cho bài thực tập 5)

- Xây dựng hệ thống như ở câu 1 với top mô đun là *Bai4Cau2.v*
- Cộng vào tín hiệu âm thanh một nguồn nhiễu được tạo ra bởi counter sau:

```
module noise generator (clk, enable, Q);
    input clk, enable;
    output [23:0] Q;
    reg [2:0] counter;

always@(posedge clk)
     if (enable)
            counter = counter + 1'b1;

     assign Q = {{10{counter[2]}}, counter, 11'd0};
endmodule
```

- Biên dịch, nạp lên board DE2, kết nối micro và headphone sau đó nghe tín hiệu ra ở headphone

-So sánh kết quả thu được với kết quả ở câu 1

.....

.....

.....

Thay đổi độ rộng bit của counter thành 4bit, 5bit, thực hiện lại như trên. Nhận xét kết quả thu được?

.....

.....

.....

.....

.....

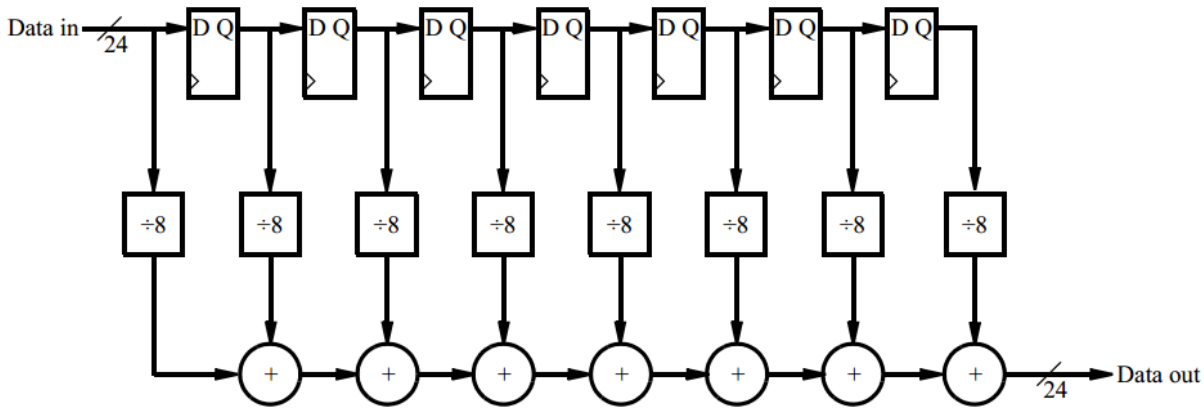
Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 5: LỌC NHIỄU ÂM THANH SỬ DỤNG BỘ LỌC TRUNG BÌNH FIR 8 BẬC

Bài thực hành này giới thiệu một kỹ thuật cơ bản trong xử lý tín hiệu số đó là lọc. Lọc là một quá trình làm biến đổi tín hiệu (ví dụ như là loại bỏ nhiễu trong tín hiệu…). Nhiễu trong âm thanh rất nhỏ nhưng làm thay đổi biên độ tín hiệu. Một bộ lọc nhiễu đơn giản đó là lọc trung bình có đáp ứng xung hữu hạn (FIR). Sơ đồ bộ lọc được chỉ ra ở hình vẽ.



Chức năng của bộ lọc là loại bỏ nhiễu bằng cách lấy trung bình của các mẫu tín hiệu liền kề nhau. Trong bài thực hành này, bộ lọc loại bỏ các độ lệch nhỏ trong tín hiệu âm thanh dựa vào sự thay đổi của 8 mẫu liên tiếp. Khi sử dụng micro chất lượng thấp, bộ lọc sẽ loại bỏ nhiễu khi ta nói vào micro và làm cho âm thanh rõ ràng hơn.

Nhiệm vụ của bài thực hành là thiết kế hệ thống lọc nhiễu đơn giản sử dụng bộ lọc trung bình FIR, lấy tín hiệu từ micro và xuất âm thanh ra loa.

Câu 1: Xây dựng hệ thống lọc nhiễu sử dụng lọc FIR 8 bậc

-Xây dựng hệ thống như bài thực hành 4 với Your Circuit là bộ lọc FIR 8 bậc như trên với tín hiệu nhiễu (3bit) được cộng thêm vào.

-Biên dịch và nạp xuống board DE2.

-Kết nối MIC trên FPGA với micro hoặc lấy tín hiệu từ máy tính.

Nhận xét kết quả thu được?

.....

.....

.....

.....

Lưu ý: Nếu sử dụng micro chất lượng cao có sẵn chức năng lọc nhiễu sẽ không nhận biết được sự sai khác trong kết quả.

Câu 2: Xây dựng mạch để chọn tín hiệu âm thanh xuất ra để so sánh.

-Thực hiện lại bộ lọc ở câu 1 với SW[1:0] để chọn tín hiệu cần phát ra loa theo yêu cầu sau:

SW = 2'b00: output tín hiệu gốc

SW = 2'b01: output tín hiệu gốc + nhiễu

SW = 2'b10, 2'b11: output tín hiệu sau khi qua bộ lọc

Kiểm nghiệm kết quả thu được khi thay đổi trạng thái của SW?

.....

.....

.....

Thay đổi độ rộng của bộ phát nhiễu thành 5bit. Nhận xét kết quả thu được?

.....

.....

.....

.....

.....

.....

Nhóm: ……..

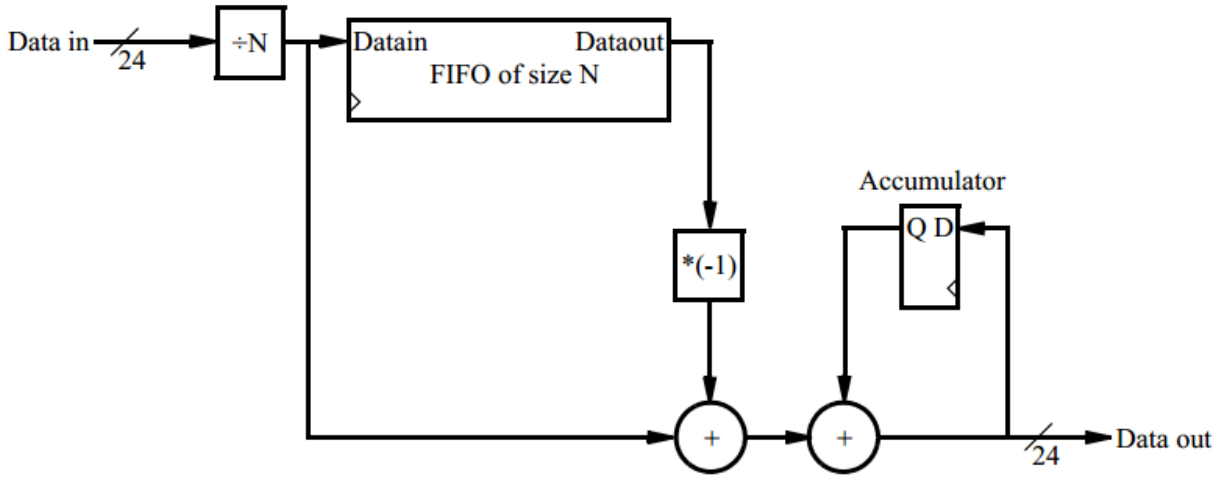
- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 6: CẢI TIẾN BỘ LỌC FIR

Việc sử dụng bộ lọc ở bài thực hành 5 có thể loại bỏ nhiễu từ microphone và loại bỏ nhiễu từ counter phát tín hiệu nhiễu. Tuy nhiên, nếu micro phone có chất lượng quá thấp hay độ rộng của counter trong bộ phát nhiễu lớn thì bộ lọc này không thể lọc hết nhiễu bởi vì bộ lọc chỉ xử lý trong 1 khung nhỏ. Do đó, ta cần xây dựng bộ lọc có bậc cao hơn, xử lý nhiều mẫu tín hiệu hơn trong 1 frame.

Trong bài thực hành này, ta sẽ thử nghiệm với bộ lọc FIR có bậc thay đổi để xác định số mẫu cần lấy trung bình ở tín hiệu vào để có thể loại bỏ nhiễu một cách tốt nhất. Để thực hiện ta sử dụng bộ lọc ở hình vẽ sau:



Để tính trung bình của N mẫu, đầu tiên bộ lọc chia tín hiệu cho N. Sau đó kết quả được lưu trữ trong bộ đệm First-In-First-Out (FIFO) có N phần tử và được cộng tích lũy ở thanh ghi Accumulator. Để đảm bảo giá trị trong thanh ghi tích lũy là trung bình của N mẫu thì ta cần phải trừ giá trị cuối cùng của bộ FIFO, vì giá trị này biểu thị mẫu thứ (N+1).

……

……

……

……

Câu 1: Xây dựng hệ thống lọc nhiễu sử dụng lọc FIR N bậc

-Xây dựng hệ thống lọc nhiễu với bộ phát nhiễu độ rộng 5bit và Your Circuit là bộ lọc FIR N=16 như hình vẽtrên.

-Biên dịch và nạp xuống board DE2.

-Kết nối audio với máy tính hay máy nghe nhạc để lấy tín hiệu.

Nhận xét kết quả thu được qua headphone?

.....

.....

.....

Câu 2: Xây dựng bộ lọc nhiễu với bậc N cao hơn

-Thực hiện lại bộ lọc ở câu 1 với N lần lượt bằng 8, 16, 32 và 64.

Chú ý: Để dễ so sánh kết quả, ta thêm vào tín hiệu SW[1:0] để chọn ngõ ra loa như sau:

SW = 2'b00: output tín hiệu gốc

SW = 2'b01: output tín hiệu gốc + nhiễu

SW = 2'b10, 2'b11: output tín hiệu sau khi qua bộ lọc

So sánh các kết quả thu được. Nhận xét?

.....

.....

.....

.....

Cho biết ưu điểm của việc chọn N = 8, 16, 32 hay 64 (là bội số của lũy thừa 2)?

.....

.....

.....

Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 7: ỨNG DỤNG LỌC MEDIAN TRONG LỌC NHIỄU ẢNH

1. Mục đích

Nhiễu đóng vai trò chủ yếu gây nên những khó khăn khi ta cần phân tích một tín hiệu ảnh. Giữa một ảnh thực và ảnh số hóa thu được có khác nhau do nhiều nguyên nhân như: nhiễu điện từ của máy thu, chất lượng kém của bộ số hóa,… Trên cơ sở đó, trong bài này thực hiện một hệ thống lọc nhiễu ảnh trên phần mềm Matlab sử dụng bô lọc trung vị (Median Filter) với mục đích nhằm cải thiện, phục hồi ảnh đã hư tổn bởi các yếu tố nhiễu gây ra như nhiễu muối tiêu (Salt and Pepper nosie) và nhiễu Gauss.

2. Phần lý thuyết

➢ Giới thiệu bộ lọc trung vị

Bộ lọc trung vị là bộ lọc làm mượt trong miền không gian. Là bộ lọc theo thống kê thứ tự. Nó thay thế giá trị của điểm ảnh bằng trung vị của các mức xám của điểm ảnh lân cận. Nó khá hiệu quả đối với hai loại nhiễu là nhiễu đốm (Speckle nosie) và nhiễu muối tiêu (Salt and Pepper nosie)

Bộ lọc trung vị được dùng phổ biến vì với một số loại nhiễu nhất định, nó có thể lọc nhiễu rất tốt với độ mờ thấp hơn so với bộ lọc làm mượt tuyến tính (xét cùng kích thước).

➢ Định nghĩa trung vị :

Cho dãy $\boldsymbol{x_1; x_2;\ldots;x_n}$đơn điệu tăng hoặc giảm. Khi đó trung vị của dãy ký hiệu là Med ([$\boldsymbol{x_n}$]), được định nghĩa :

- Nếu n lẻ : x [$\frac{\boldsymbol{n}}{\boldsymbol{2}} + \boldsymbol{1}$]
- Nếu n chẵn : x [$\frac{\boldsymbol{n}}{\boldsymbol{2}}$] hoặc x [$\frac{\boldsymbol{n}}{\boldsymbol{2}} + \boldsymbol{1}$]

➢ Toán tử cửa sổ:

Trong việc thực thi các thuật toán xử lý ảnh số cơ bản, người ta thường sử dụng một toán tử đặc biệt gọi là toán tử cửa sổ. Toán tử cửa sổ là một tập hợp có hình dạng nhất định, gồm các pixel có liên kết với một pixel trung tâm, là pixel đang được xử lý. Các phép toán

trên các pixel này sẽ có ảnh hưởng đến các pixel trung tâm cũng là các pixel đang được xử lý trong một thuật toán xử lý ảnh.

Toán tử cửa sổ có nhiều hình dạng, tùy thuộc vào thuật toán thực hiện. Tuy nhiên thường dùng nhất là các toán tử có dạng hình vuông với các cạnh là một số lẻ, ví dụ: 3x3, 5x5, 7x7...

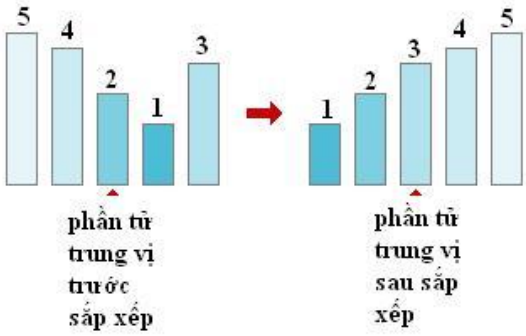
➢ **Ý tưởng :**

Ý tưởng chính của bộ lọc trung vị là như sau: ta dùng một cửa sổ lọc (ma trận 3x3) quét qua lần lượt từng điểm ảnh của ảnh đầu vào input. Tại vị trí của mỗi điểm ảnh lấy giá trị của điểm ảnh tương ứng trong vùng 3x3 của ảnh gốc lấp vào ma trận lọc. Sau đó sắp xếp các điểm ảnh trong cửa sổ này theo thứ tự (tăng dần hoặc giảm dần). Cuối cùng gán điểm ảnh nằm chính giữa (Trung vị) của dãy giá trị điểm ảnh đã được sắp xếp ở trên cho giá trị điểm ảnh đang xét của đầu ra output.

Hình ảnh minh họa :

1	8	4
6	3	2
7	9	5

Hình 1: Ma trận cửa sổ 3x3



Hình 2: Tìm giá trị trung vị

➢ **Thuật toán thực hiện**

Lấy ý tưởng ở trên, chúng ta sử dụng một cửa sổ lọc (ma trận 3x3) quét qua lần lượt từng điểm ảnh của ảnh đầu vào input. Tại vị trí của mỗi điểm ảnh lấy các giá trị điểm ảnh tương ứng trong vùng 3x3 của ảnh gốc lấp vào ma trận lọc. Lúc này ta được một ma trận với tổng cộng là 9 giá trị pixel. Tiếp tục ta sử dụng các bộ so sánh giá trị của 3 pixel trên mỗi hàng của ma trận, với 9 pixel ta được 3 hàng và sau khi so sánh ta được 3 giá trị trung vị của 3 hàng, giá trị ta cần lấy là gia trị trung vị của 3 giá trị trên, như vậy với 4 lần so sánh ta sẽ có được giá trị trung vị cần tìm. Với cách làm này ta thấy việc thay thế các giá trị nhiễu sẽ làm

ảnh ítbị mờ hơn so với ý tưởng so sánh 9 giá trị pixel như nêu ở trên, vì ta thấy khoảng cách khác biệt

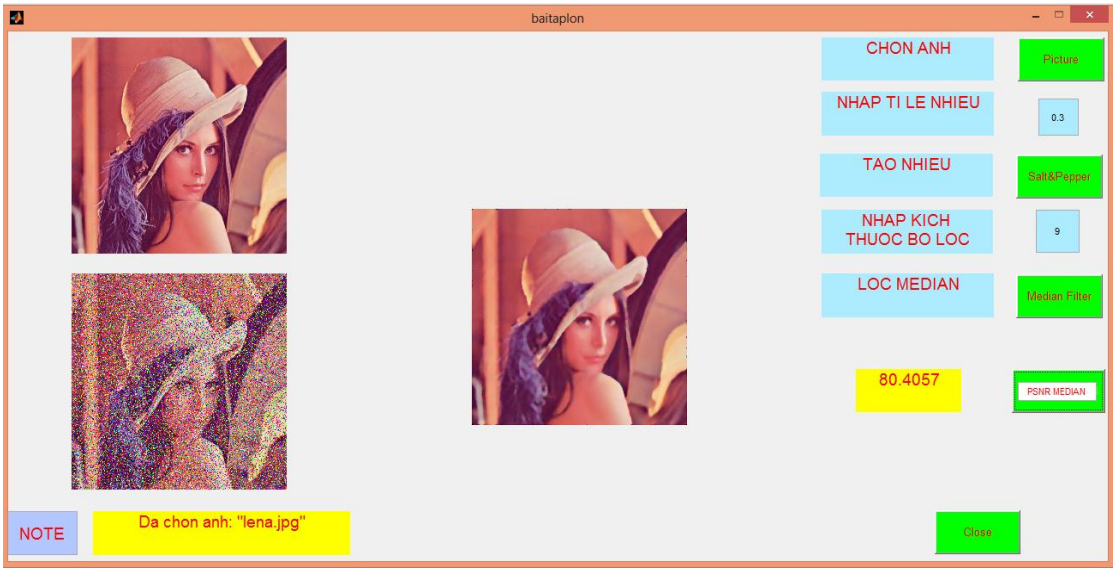
Kết quả được thể hiện qua ví dụ sau:

1	8	4
6	3	2
7	9	5

Áp dụng bộ so sánh thứ nhất đối với hàng đầu tiên ta tìm được các giá trị max1 = 8 median1 = 4 và min1 = 1, tương tự với hàng thứ hai ta được max2 = 6, median2 = 3, min2 = 2, và cuối cùng max3 = 9, median3 = 7, min3 = 5. Sau khi tìm được các giá trị này,nhiệm vụ tiếp theo là tìm giá trị trung vị của ba giá trị median1,median2 và median3 để gán cho đầu ra output, với các giá trị tìm được như trên thì sau khi qua bộ so sánh chúng ta cũng dễ dàng nhìn thấy giá trị trực quan cần tìm là 4. Vậy sau bốn bộ so sánh ta tìm được giá trị trung vị.

3. Thực nghiệm trên phần mềm Matlab

Giao diện thể hiện kết quả trên phần mềm Matlab



Hình 3: Giao diện kết quả trên Matlab

MSE(Mean Squared Error): Bình phương trung bình lỗi thể hiện sự sai lệch của ảnh khôi phục so với ảnh gốc.

$$\text{MSE} = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left[I(i,j) - K(i,j) \right]^2$$

Trong đó mxn là kích thước của ảnh; I, K lần lượt là ảnh gốc và ảnh được khôi phục

PSNR (Peak Signal-to-Noise Ratio): Tỉ số tín hiệu cực đại trên nhiễu được dùng để thể hiện chất lượng khôi phục hình ảnh so với ảnh gốc. Tỉ số PSNR càng cao thì chất lượng hình ảnh khôi phục càng gần giống với ảnh gốc ban đầu và nó được tính bởi công thức.

$$\text{PSNR} = 10\log 10\left(\frac{MAX^2}{MSE}\right)$$

MAX là giá trị tối đa của pixel trên ảnh.

4. Phần thực hành

a. Lọc trung vị Median là lọc tuyến tính hay phi tuyến?

b. Tóm tắt cách sử dụng lọc trung bình (Mean Filter) trong lọc nhiễu ảnh, so sánh với lọc trung vị (Median Filter)

c. Thực hiện lọc nhiễu cho ảnh Lena trong trường hợp ảnh bị nhiễu với các tỷ lệ khác nhau: 30%, 40%, 50%. Tính các giá trị MSE, PSNR tương ứng, nhận xét

d. Làm lại câu c trong trường hợp ảnh bị nhiễu Gauss, nhận xét.

Nhóm: ……..

- **……MSSV:…….**
- **……MSSV:……..**

----------o0o----------

BÀI 8: THIẾT KẾ HỆ THỐNG LỌC NHIỄU ẢNH TRÊN PHẦN CỨNG FPGA

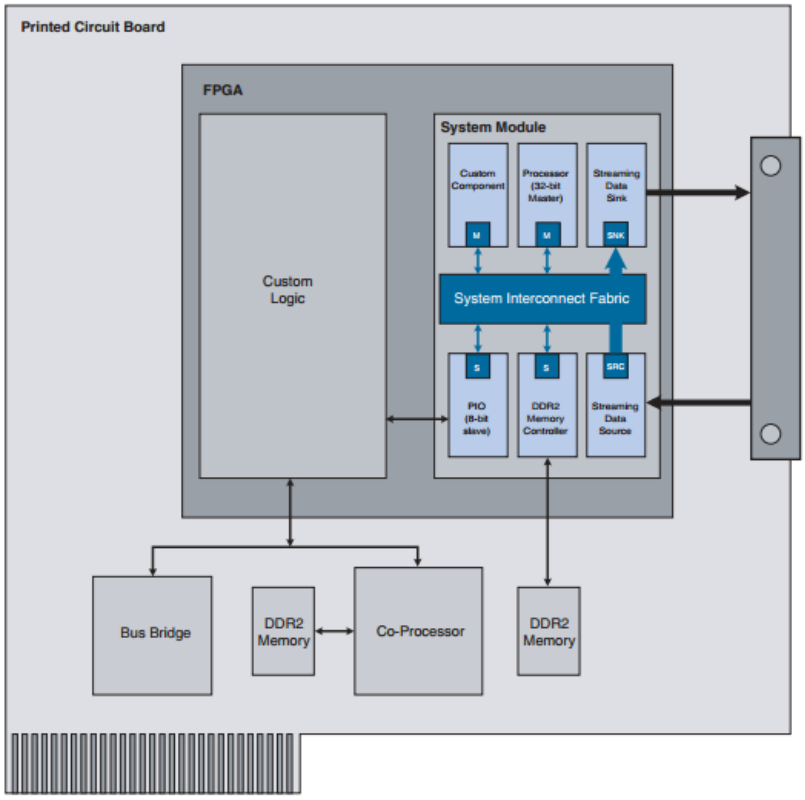
5. Mục đích

Thiết kế các hệ thống ứng dụng trên phần cứng FPGA sử dụng các tool hỗ trợ thiết kế phần cứng như: Quartus II, Modelsim, SOPC, Nios II,…

Tận dụng đặc tính của FPGA là linh hoạt và tốc độ xử lý nhanh, đặc biệt với khả năng xử lý song song, FPGA rất phù hợp với các bài toán xử lý ảnh đòi hỏi một khối lượng lớn tính toán phức tạp. Trên cơ sở đó, trong bài này thực hiện thiết kế một hệ thống lọc nhiễu ảnh trên phần cứng FPGA sử dụng phần mềm Quartus II và môi trường thiết kế hệ thống phần cứng SOPC Builder. Cụ thể ở đây là thiết kế bô lọc trung vị (Median Filter) với mục đích loại bỏ các nhiễu như: nhiễu muối tiêu (Salt and Pepper nosie) và nhiễu Gauss.Chương trình được thực hiện trên Kít DE2 của Altera, sử dụng chip FPGA họ Cyclone II.

6. Phần lý thuyết

❖ Phần mềm SoPC Builder



Hình 1: Cấu trúc SoPC được tạo ra bởi SoPC Builder

SoPC (System on Programmable Chip) là một dạng của SoC (System on Chip). Cùng với sự phát triển vượt bậc của ngành thiết kế IC (Integrated Circuit), khả năng tích hợp các linh kiện trên cùng một đế silicon ngày càng lớn, điều này dẫn đến việc tích hợp toàn bộ hệ thống (bao gồm các thành phần như: CPU, các khối tính toán, các khối DSP và các khối giao tiếp số, tương tự,...) trên cùng một chip đơn duy nhất là khả thi. Khi linh kiện logic khả trình FPGA ra đời với mật độ tích hợp ngày càng cao, các công nghệ mới ra đời tím cách đưa các thành phần khác như CPU, bộ nhớ, DSP,... vào trong linh kiện này tạo thành một hệ thống nhúng hoàn chỉnh. Cả một hệ thống được nhúng vào bên trong FPGA được gọi là một SoPC.

Hình 2: Giao diện SoPC Builder

SoPC được hỗ trợ bởi một giao diện người dùng rất thân thiện gồm các nhãn được để ở phía trên cửa sổ. Mỗi nhãn sẽ bao gồm các tác vụ được phân chia theo chức năng có liên quan với nhau.

❖ **CPU Nios II**

CPU NiosIIlàmộtkiếntrúcvixửlýnhúng32-bitđượcthiếtkếchuyênbiệt chohọFPGAcủahãngAltera. NiosIIthíchhợpchonhữngứngdụngnhúng,từDSP chođếnđiều khiểnhệ thống.

❖ **Cấu trúc của hệ thống SoPC Builder**

SoPC Builder sẽ tạo ra các đường liên kết bên trong giữa các module với nhau và sắp xếp các đường liên kết này để đảm bảo sự kết nối giữa các module. Các module trong SoPC Builder đều sử dụng giao tiếp theo chuẩn Avalon, ví dụ như định địa chỉ các bộ nhớ, thiết lập các ngắt hay xây dựng các liên kết vật lý cho các thành phần. Người thiết kế có thể sử dụng SoPC Builder để kết nối bất kỳ linh kiện logic nào (cả on-chip lẫn off-chip) mà nó có giao tiếp Avalon. Một cách tổng quát, SoPC Builder bao gồm các thành phần sau:

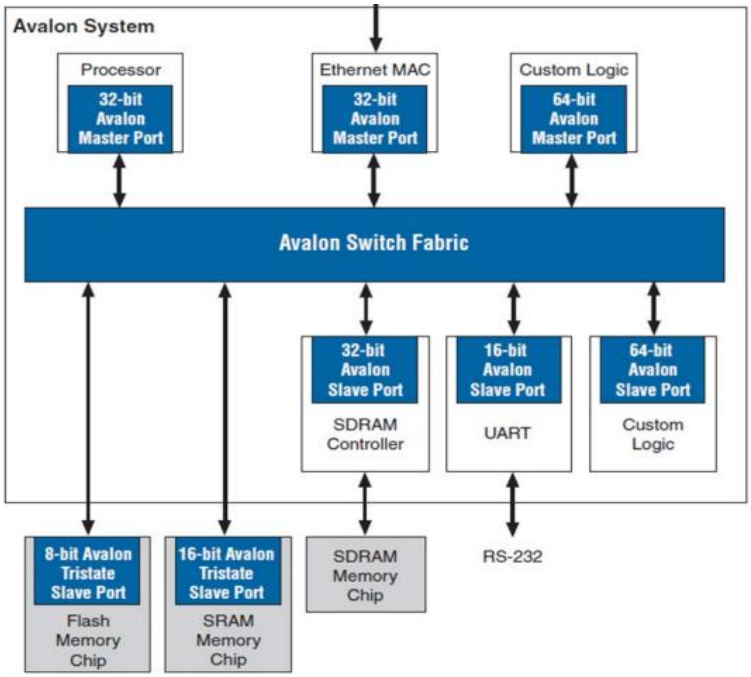
+ Vi xử lý Nios II
+ Các IP chuẩn và các thành phần giao tiếp ngoại vi
+ Các giao diện bộ nhớ
+ Các thành phần giao tiếp và bus, bao gồm cả Avalon Bus
+ Các bộ DSP

Người thiết kế có thể thêm vào hệ thống SoPC các thành phần tự thiết kế bằng cách sử dụng các ngôn ngữ mô tả phần cứng như Verilog HDL, VHDL chỉ cần chúng được thiết kế theo chuẩn giao tiếp Avalon. Sau khi đã tạo ra một SoPC, người thiết kế có thể biên dịch tất cả bằng phần mềm Quartus II trước khi cấu hình xuống chip FPGA.

❖ **Hệ thống Avalon bus**

Avalon Bus là một kiến trúc bus đơn giản được thiết kế cho kết nối giữa một vi xử lý on-chip và các thiết bị ngoại vi trong một SoPC. Avalon bus là một giao tiếp nhằm xác định kết nối cổng giữa các thành phần master và slave trên chip, đồng thời xác định thời gian (timing) giữa các thành phần khi chúng kết nối với nhau.

Đặcđiểmkỹthuậtcủagiaotiếp Avalonđượcthiếtkếchoviệctíchhợpcác thànhphầnthiếtbịngoạivitrongmôitrườngSoPC.Giaotiếpnàycungcấp chonhữngngườithiếtkếthiếtbịngoạivimộtcơsởchoviệcmôtảgiaotiếpđọc/ghi dựatrênđịachỉcủanhữngthiếtbịngoạivimaster-slavenhưvixửlý,bộnhớ,UART, timer,... Nóđịnhnghĩaviệctruyềntínhiệugiữathiếtbịngoạivivớicấu trúcliênkết Avalon switch fabric.Liênkếtnàychophépngườithiếtkếhệthốngkếtnốibất cứthiếtbịngoạivimastertớibấtcứthiếtbịngoạivislave nào.GiaotiếpAvalonmôtảliênkết cóthểcấuhìnhmànóchophépngườithiếtkếthiếtbịngoạivigiớihạnkiểutínhiệu để hỗ trợ những kiểu tín hiệu cần choviệc truyền tínhiệu.

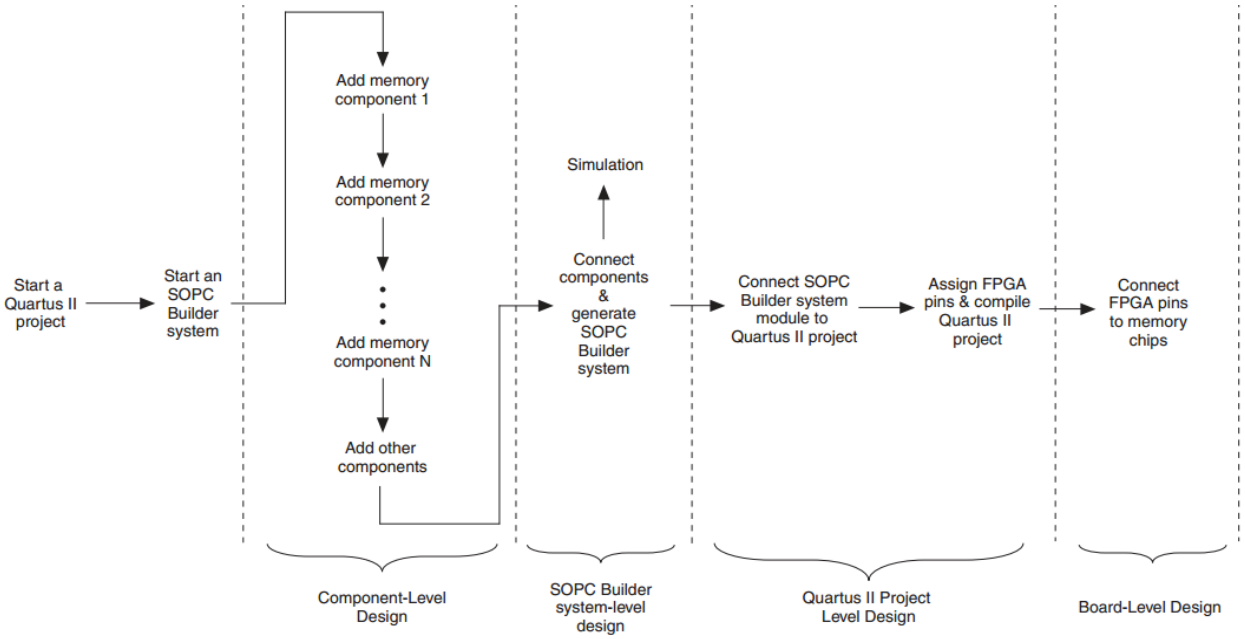


Hình 3: Giaodiện Avalon bus

Avalon bus được thiết kế đặc biệt để thích nghi với môi trường SoPC. Do đó nó là một kiến trúc bus được tích hợp sẵn rất năng động trên SoPC, bao gồm các tài nguyên logic và các đường kết nối bên trong một thiết bị logic khả trình.

❖ **Các bước thiết kế SoPC**

Các bước thiết kế một hệ thống SoPC được trình bày trong hình 4



Hình 4: Các bước thiết kế SoPC

+ Khởi tạo một SoPC với Nios II làm trung tâm

+ Tích hợp các bộ điều khiển bộ nhớ (memory controller)

+ Lập trình các module với ngôn ngữ Verilog HDL/VHDL để tạo các thành phần logic

+ Thêm các thành phần vào SoPC builder

+ Kết nối các thành phần với nhau dùng giao tiếp Avalon

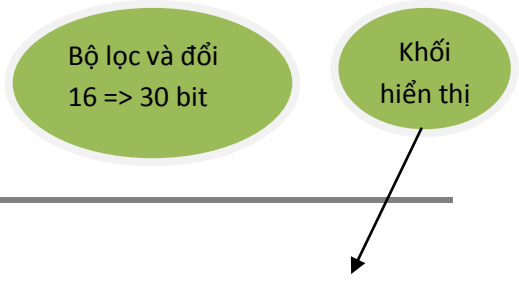
+ Biên dịch SoPC

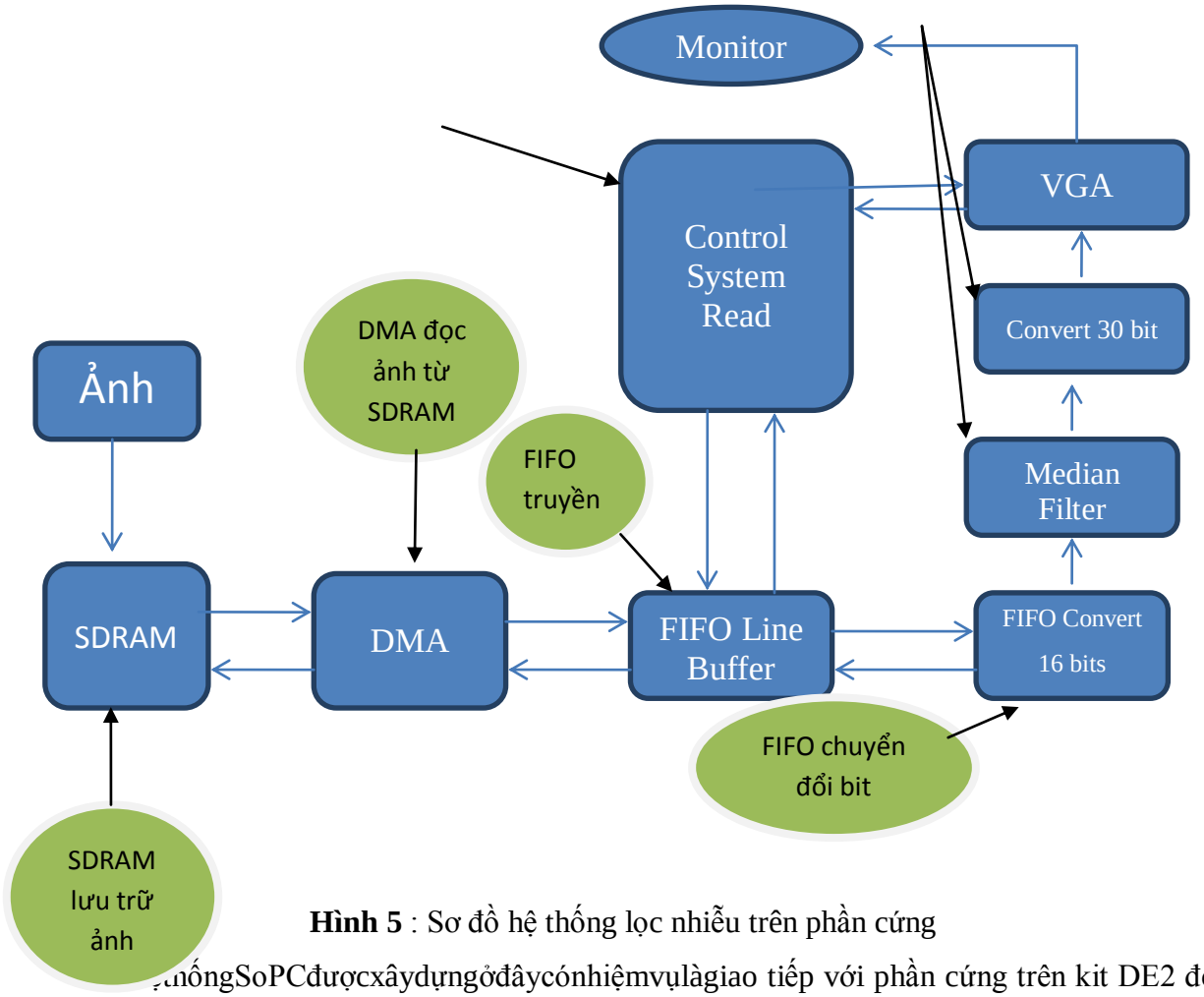
+ Lựa chọn chip trên FPGA, nối chân, biên dịch quartus II

+ Cấu hình xuống FPGA

Hầu hết các hệ thống được thiết kế với SoPC builder đều yêu cầu cần phải có bộ nhớ. Chẳng hạn, một vi xử lý nhúng bất kỳ đều cần phải có bộ nhớ để lưu trữ mã lệnh và dữ liệu. Hầu như mỗi hệ thống không chỉ có một loại bộ nhớ mà sử dụng nhiều loại bộ nhớ khác nhau trong thiết kế nhằm phục vụ cho nhiều mục đích khác nhau.

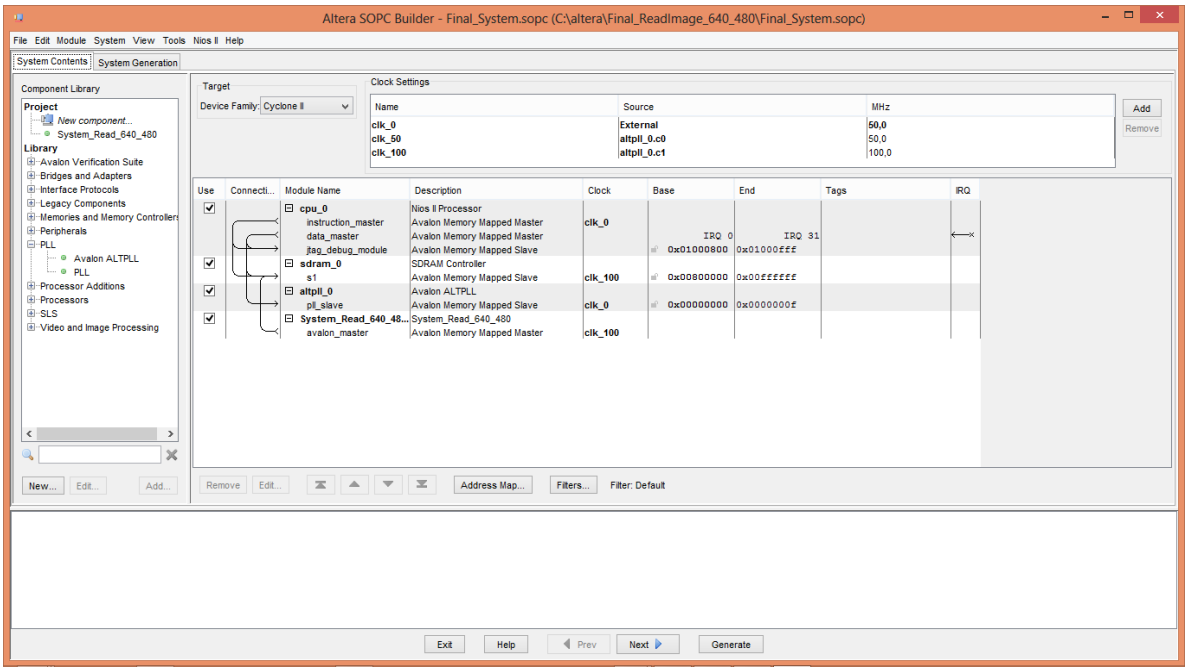
7. Thực nghiệm thiết kế hệ thống trên FPGA





Hình 5 : Sơ đồ hệ thống lọc nhiễu trên phần cứng

ệthốngSoPCđượcxâydựngởđâycónhiệmvụlàgiao tiếp với phần cứng trên kit DE2 để đọc ảnh từ SDRAM và hiển thị lên màn hình sau khi quá trình nạp file ảnh từ NIOS II xuống bộ nhớ.



Hình 6: Giao diện thiết kế hệ thống SOPC.

Qua sơ đồ trên ta thấy để hiển thị ảnh sau khi lọc nhiễu thì ta cần phải thiết kế hệ thống bao gồm các khối như trên và chức năng của từng khối như sau:

- **Ảnh 640x480:** là kích cỡ ảnh được xử lý để hiển thị lên màn hình, ảnh sau khi được chuyển đổi từ định dạng JPEG sang dạng RAW(ảnh dạng thô) thì được nạp lên SDRAM thông qua chương trình NIOS.
- **SDRAM:** là vùng nhớ để chứa dữ liệu ảnh cần xử lý, với dung lượng nhớ của SDRAM trên board DE2 của Altera lên đến 8MB thì nó là một vùng nhớ được xem là khá thuận lợi để lưu trữ ảnh với kích cỡ 640x480 được thực hiện trong đề tài.
- **DMA (DIRECT MEMORY ACCESS) Master Read:** DMA đọc data(pixel) 32bit từ bộ nhớ SDRAM để truyền dữ liệu là một cơ chếtruyền dữliệu trực tiếp giữa hai hay nhiều thành phần trong hệthống không thông qua CPU. Nhờvậy, quá trình truyền nhận giữa các thành phầncó thểthực hiện songsong nhauvà đạt đượctốc độrất cao.
- **FIFO Line Buffer:** đóng vai trò như bộ đệm để truyền dữ liệu, nó sẽ có các tín hiều cần thiết để yêu cầu khi nào khối DMA phải đọc hoặc truyền dữ liệu.
- **FIFO_Convert 16 bits:** chuyển đổi data 32 bit thành 16 bit để thực hiện khối xử lý ảnh Median_Filter.
- **Median_Filter:** bộ lọc trung vị có chức năng lọc nhiễu ảnh, sau khi qua xử lý ảnh hiển thị lên màn hình sẽ rõ ràng hơn.
- **Convert 30 bit:** là bộ chuyển đổi nhằm mục đích đồng bộ tín hiệu với VGA.
- **VGA (Video Graphics Array):** đồng bộ thời gian để quét hang ngang cũng như hàng dọc cho một file ảnh hiển thị lên màn hình không bị lỗi.
- **Control System Read:** là khối điều khiển để việc truyền ảnh và hiển thị ảnh được đồng bộ.
- **Monitor:** hiển thị kết quả.

8. Phần thực hành

e. Nguyên lý thiết kế hệ thống trên phần cứng với SoPC Builder, Avalon bus, Nios II?

f. Thiết kế hệ thống SoPC xuất ảnh ra màn hình VGA sử dụng kit FPGA DE2 của Altera.

g. Sử dụng ngôn ngữ Verilog thiết kế bộ lọc Median với kích thước cửa sổ lọc 3x3. Mô phỏng, kiểm tra chức năng của bộ lọc.

h. Thiết kế hệ thống lọc nhiễu ảnh hoàn chỉnh cho ảnh Lena theo sơ đồ trong hình 5. Kiểm tra chức năng của từng khối trên SignalTap.

i. Làm lại câu d trong trường hợp ảnh bị nhiễu với các tỷ lệ khác nhau: 30%, 40%, 50%. Tính các giá trị MSE, PSNR tương ứng, nhận xét.