**MUSIC CONTROLLED LED LIGHTS USING FPGA**

**MEMBERS:**

* Aditya Sharma
* Dhriti Gupta
* Tanya Rampal
* Spandan Gera

**OBJECTIVE:**

To control switching of LED lights from the spectrum of the input music/audio signal.

**WHY THIS PROJECT:**

* To get acquainted with FPGA’s and their working.
* To improve our ability to code in Verilog.
* To understand signal processing and analysis.

**OVERVIEW:**

The main aim of this project is audio signal analysis. In overview this is the process that should be followed:

* Noise removal and amplification of input signal.
* Conversion of time domain signal into frequency domain.
* Driving the output pin from the intensity vs frequency info and lighting up the corresponding LED.

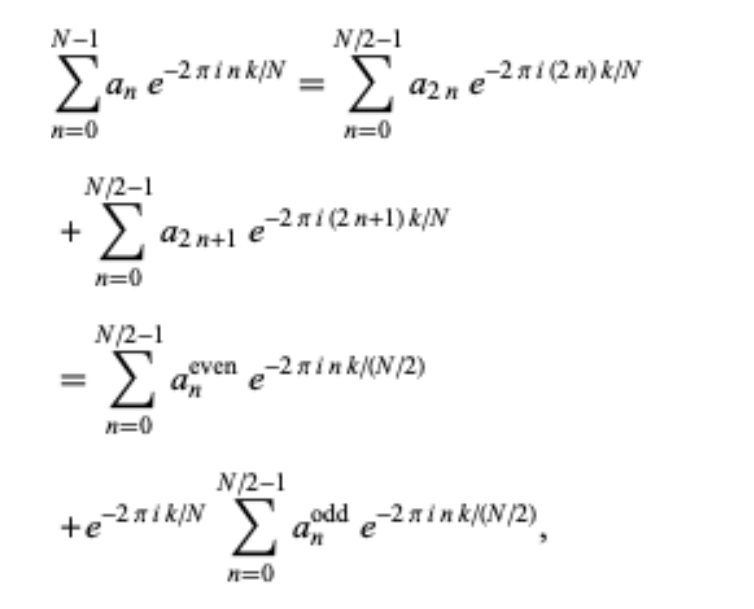
**CONCEPTS:**

* We initially started looking for efficient ways to undergo the conversion from time domain to frequency domain. This is explained in the points that follow:
* The fast Fourier transform algorithm [**FFT**] works really well to suit our purpose.
* For this we need an ADC to convert our input signal into a digital signal.
* After this, we have to code the **FFT** algorithm in Verilog to implement it on the FPGA.
* **The Fast Fourier Transform Algorithm:**

The FFT algorithm is basically a faster and efficient method to find the discrete Fourier transform.

For this project we have used the decimation in time Fourier transform algorithm.

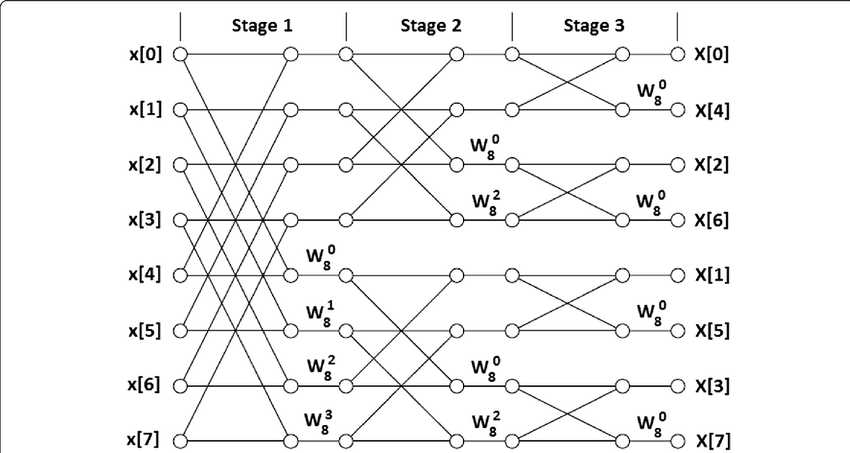
The basic idea is to break up a transform of N points into 2 transforms of length N/2 and further breaking it into 4 transforms of length N/4 and so on. Mathematically:



The image shown above shows how a general N-point Fourier transform is carried out using the FFT algorithm.

To code this in Verilog, we can start with simpler modules like that of 2-point FFT and then using them write codes for 4-point, 8-point and 16-point FFT and so on as required. This results in a butterfly diagram which makes it really easy to visualize the whole thing.

Below is a butterfly diagram for 8-point FFT. Here denotes



* After getting the intensity Vs frequency relations, the output pins of FPGA will be either LOW or HIGH depending upon the intensity value at different frequencies and then the corresponding output will drive the LED’s.

**PROGRESS:**

* FPGA: We decided to use the Spartan 6 FPGA. It has all the features we would require for our project.
* After understanding the concepts, we simulated a model using MATLAB and Simulink to get an idea of what our inputs and outputs and the whole process would look like.
* For this, we took a sinusoidal signal as input, sampled it, analyzed the frequency spectrum and plotted the peak intensity values (amplitude) at different frequencies using peak finder.
* The Code is:

Fs = 1000; % samples per second

dt = 1/Fs; % seconds per sample

StopTime = 10; % seconds

t = (0:dt:2\*(StopTime-dt))'; % seconds

%%Sine wave:

Fc = 100; % hertz

x = (sin(2\*pi\*Fc\*t))+(sin(2\*pi\*2\*Fc\*t));

wave.signals.values = x;

wave.time = [];

model = 'hey.slx';

open\_system(model);

sablock = 'hey/Spectrum Analyzer';

cfg = get\_param(sablock,'ScopeConfiguration');

cfg.PeakFinder.Enable = true;

sim(model);

data = getMeasurementsData(cfg);

p = data.PeakFinder.Value;

f = data.PeakFinder.Frequency;

l=length(p);

int=[];

freq=[];

for i=1:l

if f(i)>-3

int(end+1)=p(i)

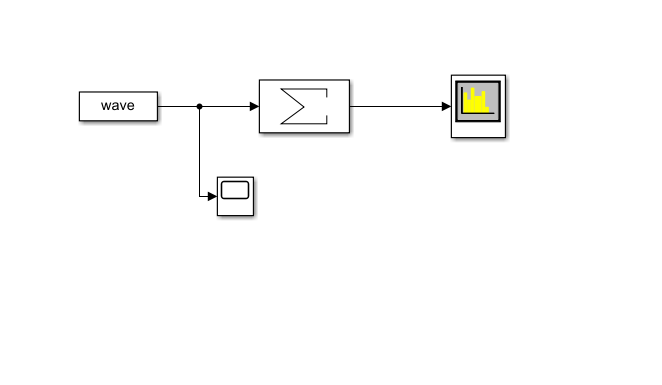
freq(end+1)=f(i);

end

end

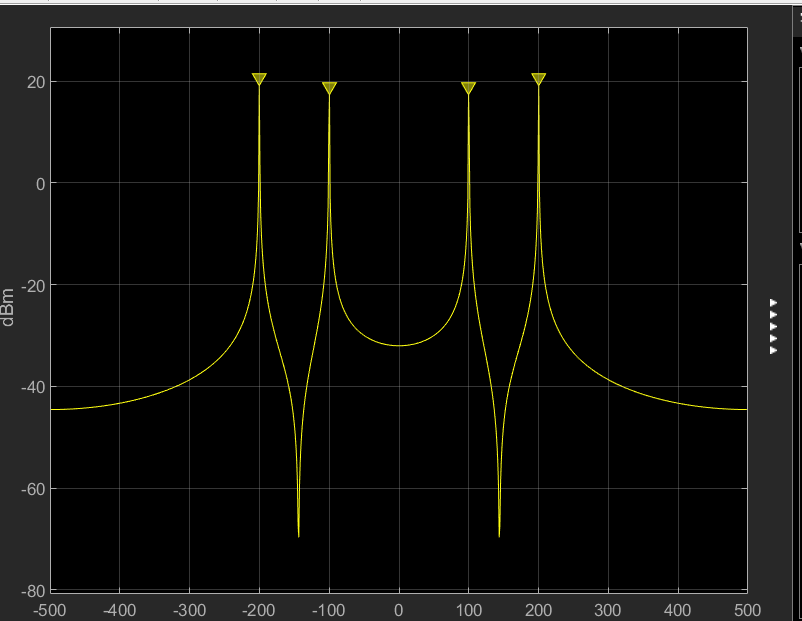
plot(freq,int,'o');

* Simulink model:

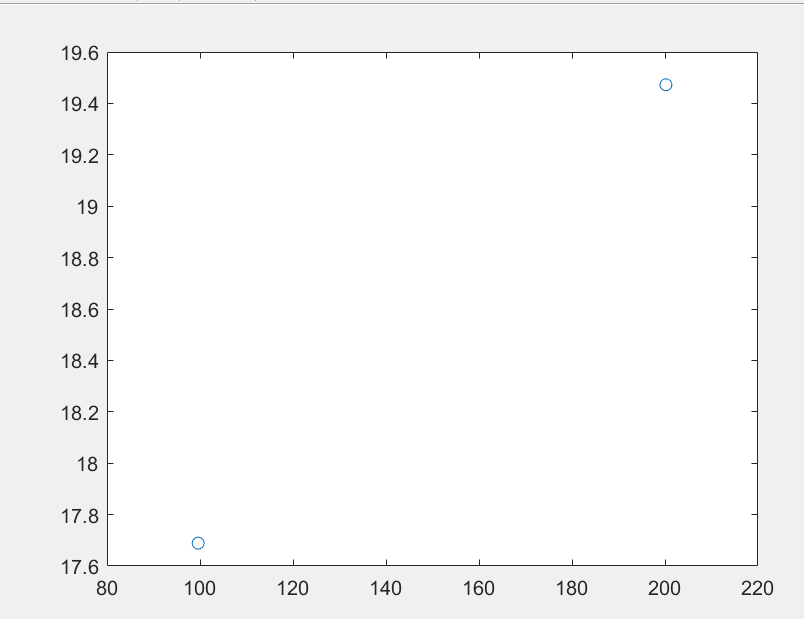


* We got the output for the sine waves and these were identical to what we had expected:

Output from the spectrum analyser:



Output from the peak finder:



* Spartan 6 does not have an inbuilt ADC, so we will be passing the input signal through an ADC and then use that as FPGA input.
* We are working on the Verilog code for FFT.

We started with a behavioural code for 2-point FFT and then moved further to a 16- point FFT.

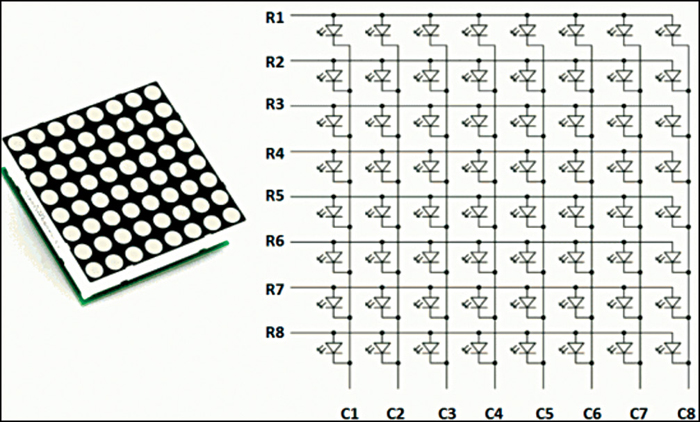
The code is being improved to make it more robust and have better synthesis of the final schematic on the FPGA.

* Another method that we are working on is using MATLAB to convert the domain of the input signal. MATLAB and Simulink have an add-on called HDL coder which converts the functioning block into an RTL HDL code. We are using the same MATLAB code and Simulink model shown on the previous pages to do this. This will make the conversion possible without writing a Verilog code for FFT.
* Along with this we are working on the last part of our project that is the portion of code that will be used to drive the output pins of the FPGA. The plan is to use an 8\*8 LED matrix for display. 16 output pins of the FPGA will be used. Each pin corresponds to a row or a column of the LED matrix.

The rows indicate the frequency and the columns indicate the intensity at that frequency. More the intensity more no. of LED’s will glow. Thus we have modelled a mini-qualizer using the matrix.

For a particular frequency band B the **row corresponding to it will get a HIGH signal and the column/s corresponding to the amount of intensity will get a LOW signal.**

Circuit diagram of LED matrix:



Each row and column is a vector of 8 bits as shown below, and 10 corresponds to ON whereas 01 corresponds to OFF.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1=pin[0] | 0th bit of row 0  7th bit of column8 | 1st bit of row 0  7th bit of column 9 |  |  |  |  |  |  |
| R2=pin[1] |  |  |  |  |  |  |  |  |
| R3=pin[2] |  |  |  |  |  |  |  |  |
| R4=pin[3] |  |  |  |  |  |  |  |  |
| R5=pin[4] |  |  |  |  |  |  |  |  |
| R6=pin[5] |  |  |  |  |  |  |  |  |
| R7=pin[6] |  |  |  |  |  |  |  |  |
| R8=pin[7] |  |  |  |  |  |  |  |  |
|  | C1=pin[8] | C2=pin[9] | C3=pin[10] | C4=pin[11] | C5=pin[12] | C6=pin[13] | C7=pin[14] | C8=pin[15] |

Verilog code for frequency and intensity mapping:

// Code your design here

module pinmap(freq,in,pin);

input [12:0] freq [7:0];

input [12:0] in [7:0];

output reg [0:7] pin [15:0];

int i,j,k,l;

reg [12:0]band;

//range from 200-3000Hz

always @(\*)

begin

for(i=0; i<8; i=i+1)

pin[i]=8'b00000000;

for(i=8; i<16; i=i+1)

pin[i]=8'b11111111;

for (j=0;j<8;j=j+1)

begin

band=200+350\*(j+1);

for (i=0;i<8;i=i+1) begin

if(freq[i]<=band & freq[i]>(band-350)) begin

for(k=0; k<8; k=k+1) //rows

begin

if(in[i]>60+3\*k)

pin[j][k]=1;

else

pin[j][k]=0;

end

for(k=0; k<8; k=k+1) //columns

pin[k+8][7-j]=!pin[j][k];

end

end

end

end

endmodule

Testbench:

// Code your testbench here

// or browse Examples

module test;

reg [12:0] freq[7:0];

reg [12:0] in [7:0];

wire [0:7] pin [15:0];

pinmap uut(freq,in,pin);

initial begin

#5 freq[0]=256; in[0]=76;

freq[1]=1000; in[1]=65;

freq[2]=2900; in[2]=83;

#5 for(int i=0;i<16;i=i+1)

$display(i," %b",pin[i]);

#5 freq[0]=700; in[0]=60;

freq[1]=1900; in[1]=62;

freq[2]=2300; in[2]=91;

#5 for(int i=0;i<16;i=i+1)

$display(i," %b",pin[i]);

end

endmodule

**PROBLEMS:**

* The major problem/ bottleneck for this project is the amount of info we can gather from our approach. Practically, there will always be a demand for more information, so that is what we are focusing on by manipulating our Verilog codes and using more LED’s while also taking care of the amount of hardware that we should ideally use.
* Extracting the frequency and intensities from the analog signal is has an easier implementation in MATLAB, however the conversion of this model into HDL format is proving to be difficult.
* Another issue that might occur is the presence of noise in the input signal. Filtering the noisy components before the ADC operates on the signal is thus important.