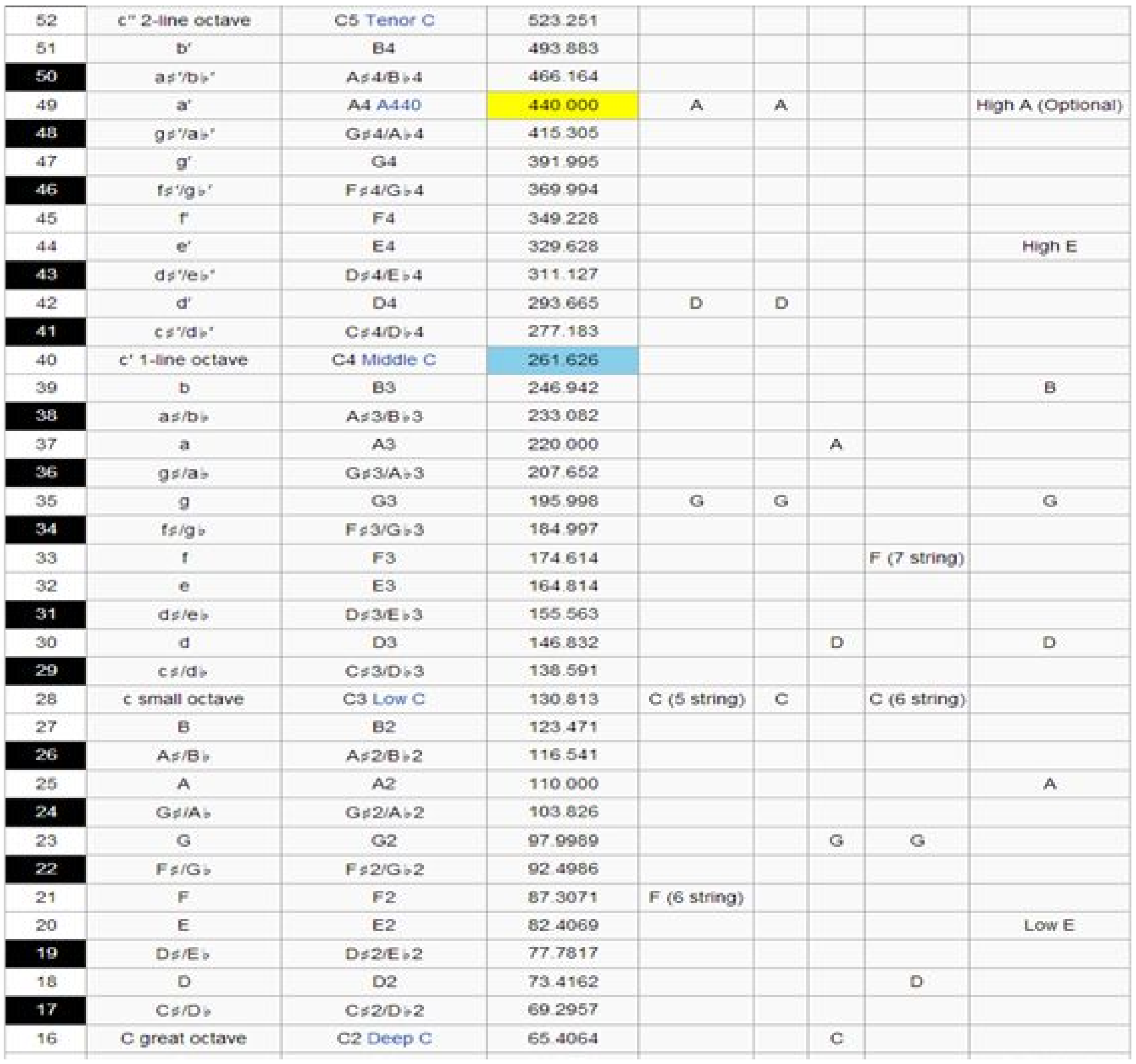
Audio Synthesizer

Prashanth H C (prashanth.c@iiitb.org)

Aim of the Project:

Aim of the Project is to send the Audio notes (Like Keyboard Instrument) to FPGA Using UART where each key represents a frequency. Map this note to an audio and send it to PC again using UART. And save the audio in a .wav file and play it.

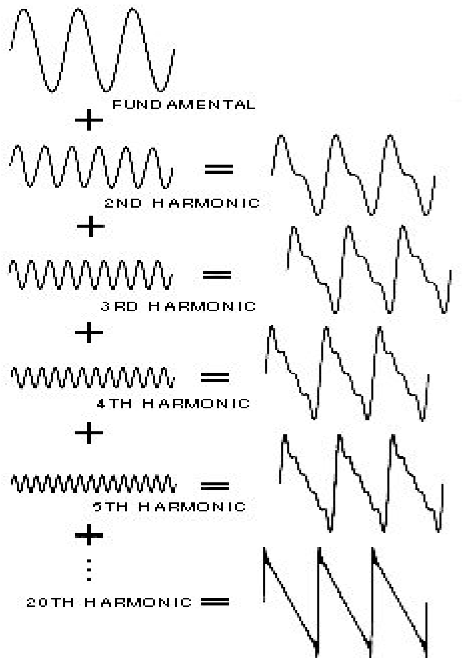
Frequencies of different Octave notes:



52	c'' 2-line octave	C5 Tenor C	523.251					
51	b'	B4	493.883					
50	a♯'/b♭'	A♯4/B♭4	466.164					
49	a'	A4 A440	440.000	A	A			High A (Optional)
48	g♯'/a♭'	G♯4/A♭4	415.305					
47	g'	G4	391.995					
46	f♯'/g♭'	F♯4/G♭4	369.994					
45	f'	F4	349.228					
44	e'	E4	329.628					High E
43	d♯'/e♭'	D♯4/E♭4	311.127					
42	d'	D4	293.665	D	D			
41	c♯'/d♭'	C♯4/D♭4	277.183					
40	c' 1-line octave	C4 Middle C	261.626					
39	b	B3	246.942					B
38	a♯/b♭	A♯3/B♭3	233.082					
37	a	A3	220.000			A		
36	g♯/a♭	G♯3/A♭3	207.652					
35	g	G3	195.998	G	G			G
34	f♯/g♭	F♯3/G♭3	184.997					
33	f	F3	174.614				F (7 string)	
32	e	E3	164.814					
31	d♯/e♭	D♯3/E♭3	155.563					
30	d	D3	146.832			D		D
29	c♯/d♭	C♯3/D♭3	138.591					
28	c small octave	C3 Low C	130.813	C (5 string)	C		C (6 string)	
27	B	B2	123.471					
26	A♯/B♭	A♯2/B♭2	116.541					
25	A	A2	110.000					A
24	G♯/A♭	G♯2/A♭2	103.826					
23	G	G2	97.9989			G	G	
22	F♯/G♭	F♯2/G♭2	92.4986					
21	F	F2	87.3071	F (6 string)				
20	E	E2	82.4069					Low E
19	D♯/E♭	D♯2/E♭2	77.7817					
18	D	D2	73.4162				D	
17	C♯/D♭	C♯2/D♭2	69.2957					
16	C great octave	C2 Deep C	65.4064			C		

Synthesis method :

To synthesize the waveform that replicates the sound produced by piano, additive synthesis is used. In additive synthesis we add the fundamental frequency (note frequency) with its harmonics.Harmonic additive synthesis is closely related to the concept of a Fourier series which is a way of expressing a periodic function as the sum of sinusoidal functions with frequencies equal to integer multiples of a common fundamental frequency.
In general, a Fourier series contains an infinite number of sinusoidal components, with no upper limit to the frequency of the sinusoidal functions and includes a DC component (one with frequency of 0 Hz). Frequencies outside of the human audible range can be omitted in additive synthesis.



As a result, only a finite number of sinusoidal terms with frequencies that lie within the audible range are modeled in additive synthesis.

Considering the limitations of hardware resources and to keep the implementation basic and simple we have considered only the first harmonic.

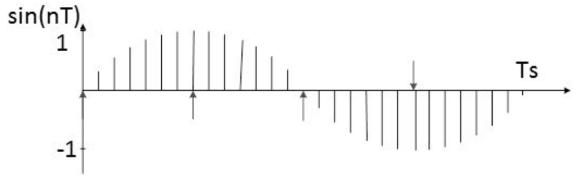
To generate discrete samples of audio using pure sinusoidal waves, the general form for additive synthesis can be given as ;

$$y[n] = \sum_{k=1}^{K} r_k[n] \cos(\theta_k[n])$$

where,

$$\theta_k[n] = \frac{2\pi}{f_s} \sum_{i=1}^{n} f_k[i] + \phi_k$$
$$= \theta_k[n-1] + \frac{2\pi}{f_s} f_k[n]$$

For the case with one harmonic frequency $k:\ fn,\ 2*fn$ for fundamental frequency fn. To generate a full cycle, we have to vary $\theta k[n]$ from 0 to 360°.



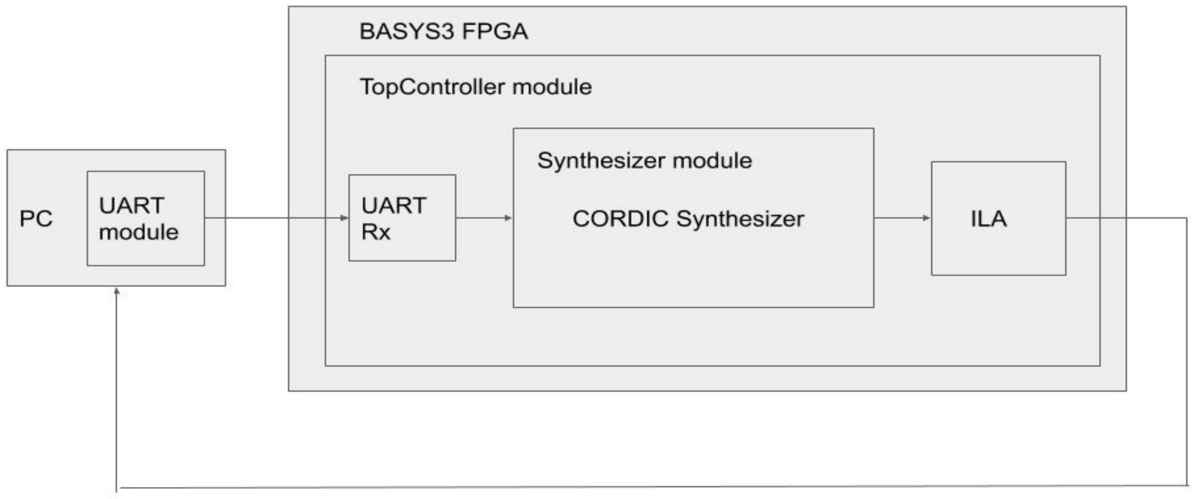
In the implementation with cordic core, we can vary the input phase from -pi to +pi to generate a full cycle. The rate at which we vary phase wrt to sample n determines the frequency of the sinusoid, with a fixed sampling rate.

Having the pure sinusoidal samples, they can be shaped using a shaping window to replicate the waveform of any musical instrument. In the fpga implementation, these waveform shaping coefficients are stored in a block RAM.

For replicating the tone produced by piano, this method is one of the simplest : Making music with maths. Our fpga implementation tries to perform this method, but without the self multiplication (to keep the number of harmonics fixed to two).

With a sampling rate of 8800, we can produce a 880Hz harmonic with 10 samples per cycle. Which implies a sinusoidal of 440Hz can be produced with at least 10x the minimum sampling rate. The tones of higher frequencies can also be produced but with very low samples per cycle.
The notes A4 (A440) to A0 can be guaranteed to have sufficient samples per cycle.

FPGA Implementation :



Some implementation details :

1) The sinusoidals are produced using CORDIC core of vivado IP by varying phase from -pi tp pi
2) The wave shaping window coefficients are pre calculated using matlab and stored in block RAM
3) The synthesized note/wave is stored in a block RAM.
4) The note that has to be synthesized along with other options are sent through UART module.
5) ILA IP is used to view the contents of synthesized samples stored and also to send the the scope data to PC.

Sending Audio notes from the PC to the FPGA:

As we know in order to provide different Audio notes to FPGA we have to specify the frequency of each note and send it through UART, so we checked for the frequency values for different audio notes on internet and found some frequencies of higher notes exceeds the 255Hz as we are sending 8 bit data and for 8 bit data maximum limit is 255 so we had to encode the frequencies of different notes to a 8 bit value and   sent it through the UART. In this project we used 21 keys in the keypad of the PC which represents 21audio notes of 3 octaves and sent those through UART.

3 octaves are:

1. c-small octave which consists of the piano keys A2, B2, C2, D2, E2, F2 and G2
2. C'1 line octave which consists of the piano keys A3, B3, C3, D3, E3, F3 and G3
3. C''2-line octave which consists of the piano keys A4, B4, C4, D4, E4, F4 and G4

In our experiment we are generating different kinds of waveforms and each waveform was synthesised to produce different kinds of tones.

1. Wave form which represents the piano audio.
2. Pure sine wave with chirping effect.
3. Pure sine wave without chirping effect.
4. Any other digital synthesized audio for which an envelope can be defined.

So, to provide encoded 8-bit value to the FPGA which represents both note and the wave form types coding was done in Python which first asks user to select one from the three waveforms i.e., piano wave, pure sine wave without chirping effect and pure sine wave with chirping effect.

Then after selection of waveform it will ask user to provide notes, they want to play which ranges in between the 3-octaves mentioned above by pressing different keys of PC.

And the print statement is mentioned below:

Enter the note you want to play:

c-small octave: for A2 press'z',B2 press'x',C2 press'c',D2 press'v',E2 press'b',F2 press'n',G2 press 'm': C'1 line octave: for A3 press'a',B3 press's',C3 press'd',D3 press'f',E3 press'g',F3 press'h',G3 press 'j'   : C''2 line octave: for A4 press'q',B4 press'w',C4 press'e',D4 press'r',E4 press't',F4 press'y',G4 press 'u' "

Encoded Value:

So according to the input some encoded 8-bit value is sent from the PC through the UART to the top controller module in FPGA.

And the 8-bit number encoding is mentioned something like this:

1. MSB bit(8th bit) represents whether the user has selected or Piano or Pure sine wave
2. 7th bit represents whether the user wants chirping effect or not
3. and other 6 bits represent the encoded value of notes (and the 6 bits starts from 000000 which represents A2 and end with 010100 which represents G4 note)

For Example.

1. 01000000 represents user has selected pure sinewave with chirping effect and the note is A2.
2. 10000011 represents user has selected piano tone output and the note he wants to listen is D2

Decoded Value:

So, this 8-bit value is sent to the top controller and in the top controller the 8-bit value is decoded the MSB 2 bits are used as select lines to generate piano waveform or pure sine wave form with chirping or pure sine waveform without chirping effect, and the last 5 bits are decoded to get the frequency which represents different octave notes.

Decoding of Audio Notes in Top Controller Module:

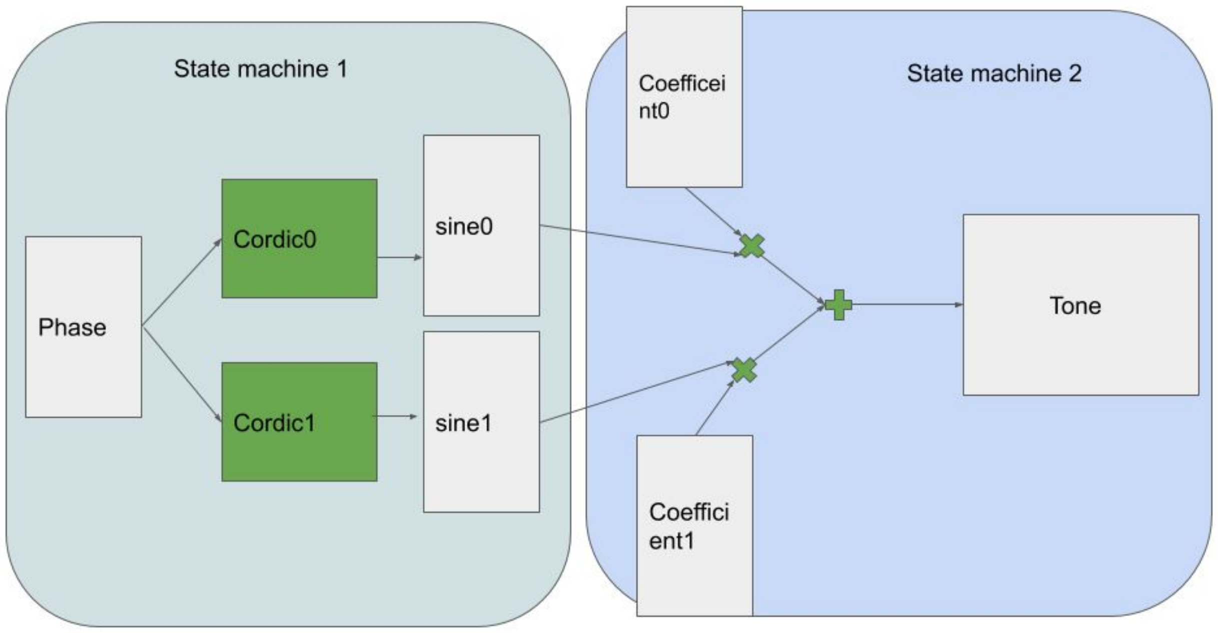
Top Controller module table (Table used to provide frequency input to the cordic module when certain key is pressed in the PC wjich indicates certain note):

```
case(freq_div)

 5'b00000: freq1 = 110;//A2 note

 5'b00001: freq1 = 124;//B2

 5'b00010: freq1 = 69;//C2

 5'b00011: freq1 = 74;//D2
```

```
        5'b00100: freq1 = 82;//E2

        5'b00101: freq1 = 88;//F2

        5'b00110: freq1 = 98;//G2

        5'b00111: freq1 = 220;//A3

        5'b01000: freq1 = 246;//B3

        5'b01001: freq1 = 130;//C3

        5'b01010: freq1 = 146;//D3

        5'b01011: freq1 = 164;//E3

        5'b01100: freq1 = 174;//F3

        5'b01101: freq1 = 196;//G3

        5'b01110: freq1 = 440;//A4

        5'b01111: freq1 = 493;//B4

        5'b10000: freq1 = 262;//C4

        5'b10001; freq1 = 294;//D4

        5'b10010: freq1 = 330;//E4

        5'b10011: freq1 = 350;//F4

        5'b10100: freq1 = 392;//G4
```

The encoded frequency value is decoded in topcontroller and actual frequency given to cordic synthesizer.

Cordic Synthesizer :



In the cordic synthesizer there are 6 block RAM's.
5 no.s of 8800x8 to store phase information(x1), calculated sine waves(x2), to store 8bit shaping window coefficients(x2)
The phase information memory is configures as simple dual port ROM,coefficient memory are configured as single port ROMs.
To store the synthesized tone, one 8800x16 RAM is used.

The vivado cordic IP takes phase value from the phaseROM, computes the corresponding values of sine and cos. This value is stored in the calculatedsineRAM.

This operation of calculating pure sine waves using cordic cores is controlled by a state machine. It takes two clock cycles to calculate one sample value using the state machine, giving sufficient time window for cordic cores.

The pure sine waves are then multiplied with window shaping coefficients stored in coefficientsROM. The shaped wave is then added with similarly shaped first harmonic. This result is stored in the tone RAM.
The operation of shaping the wave, store their sum in tone RAM is controlled by second state machine. The second state machine is run at a clock period twice that of first state machine.

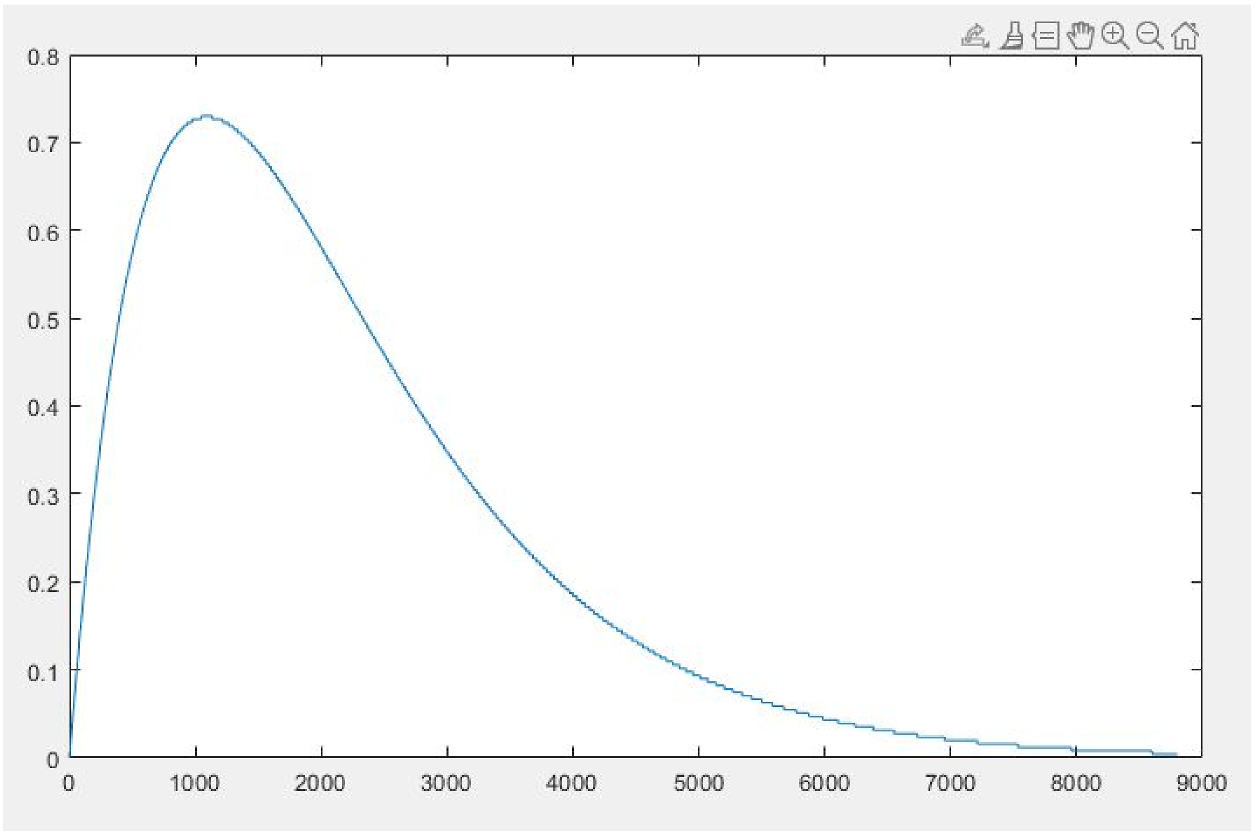
To generate a tone which replicates musical instruments, the corresponding shaping window coefficients are places in coefficientsROM. To generate pure sine wave, the coefficients0 can be made equal to one, coefficients1 made zero.

To control the frequency in a fixed sampling rate synthesizer, we can simple vary the phase according to the frequency needed.
Example for a frequency of 440Hz:
With a sampling rate of 8800, there are 8800 samples in one second.
440Hz sine wave has 440 cycles in one second → there are 8800/440 = 20 samples per cycle of 440Hz wave if sampled at 8800.
To generate this wave, we divide a cycle into 20 values of phase.

This phase can be stored in the phase block ROM, and read address increment according to the frequency we want to synthesize. For producing the chirp effect we can increment the frequency after every cycle, higher the increment value sharper the chirping effect.

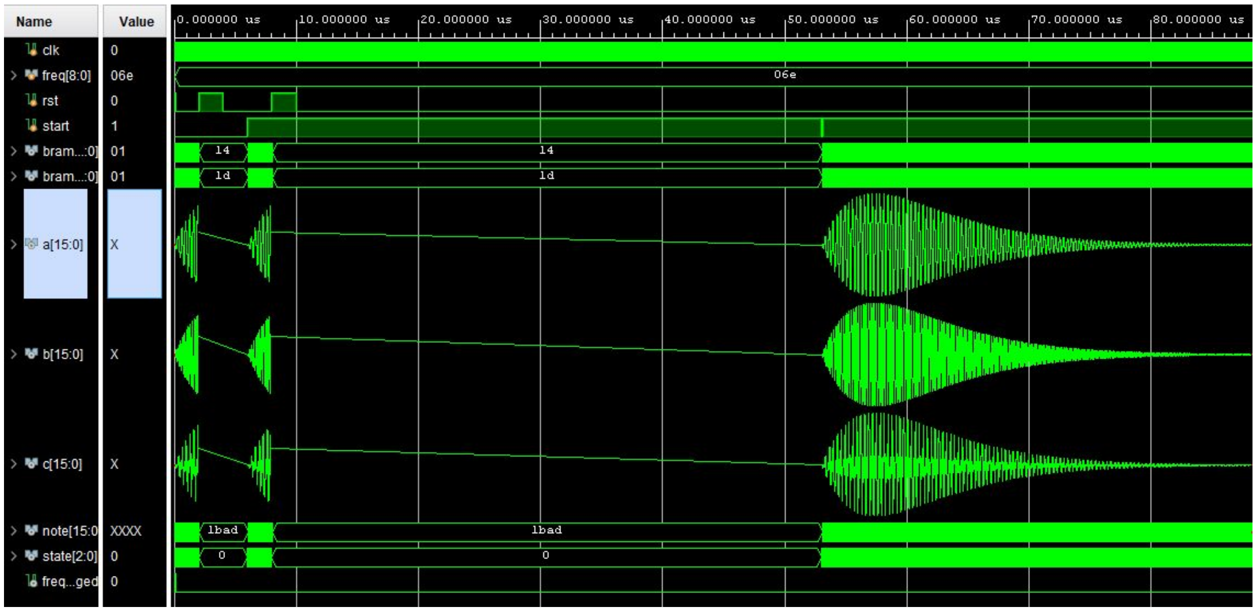
To increase the resolution of coefficients, all 8 bits can be used to represent the magnitude, with unsigned number of 8bits for fractional part. The sine wave generated by cordic module has one sign bit, one bit for magnitude, rest for fractional part.

Wave shaping window :



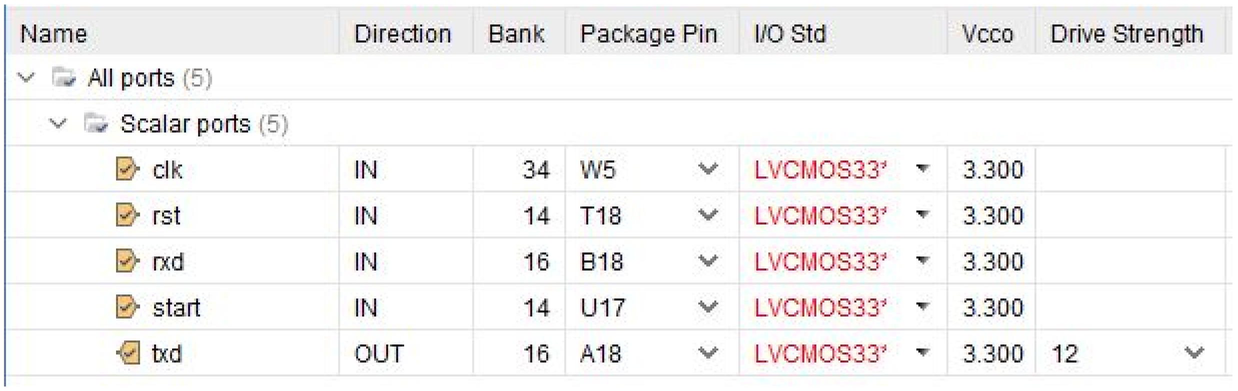
Behavioral Simulation result :

For piano tone:



The usage of inputs rst,start,freq is represented in the waveform.

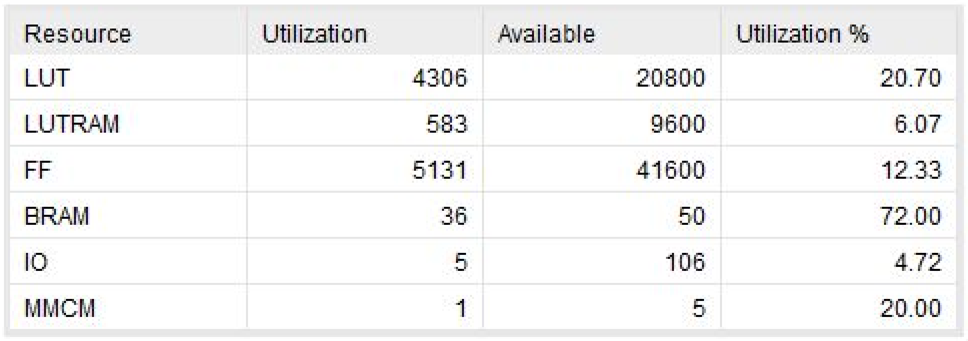
Constraints :



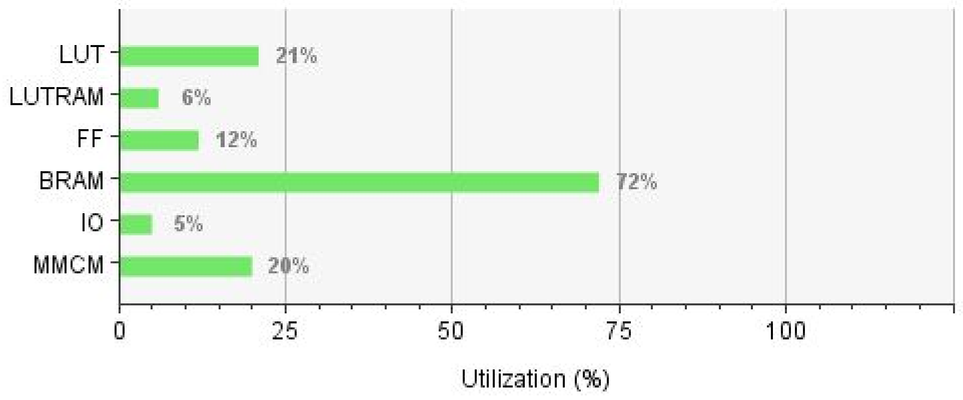
Name	Direction	Bank	Package Pin	I/O Std	Vcco	Drive Strength
All ports (5)						
Scalar ports (5)						
clk	IN	34	W5	LVCMOS33*	3.300	
rst	IN	14	T18	LVCMOS33*	3.300	
rxd	IN	16	B18	LVCMOS33*	3.300	
start	IN	14	U17	LVCMOS33*	3.300	
txd	OUT	16	A18	LVCMOS33*	3.300	12

Resource Utilization :

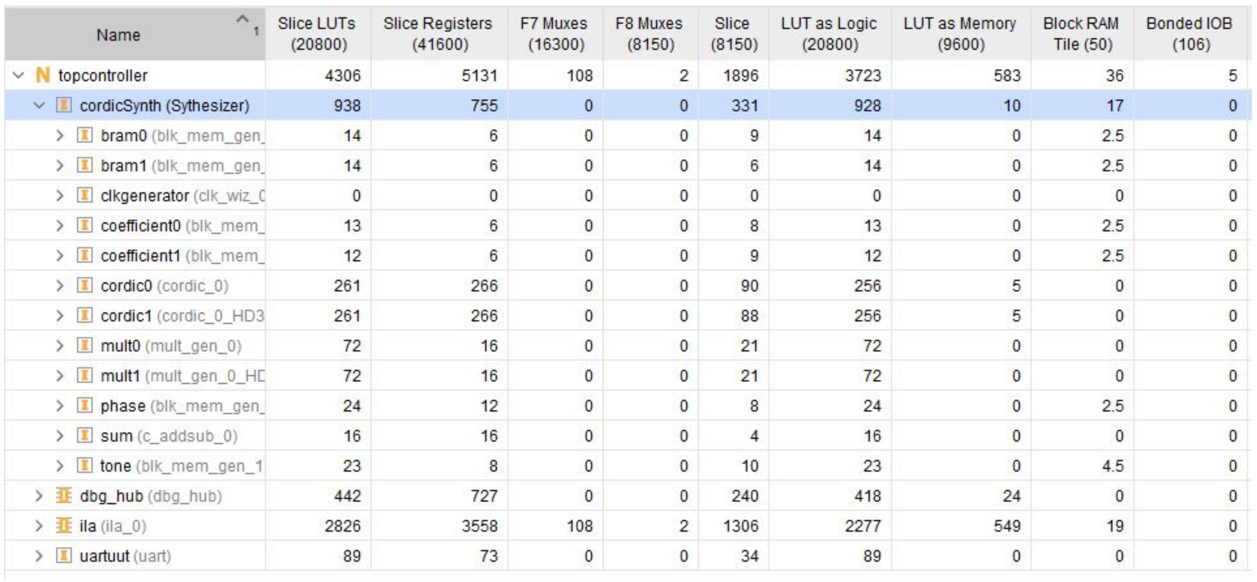
Summary



Resource	Utilization	Available	Utilization %
LUT	4306	20800	20.70
LUTRAM	583	9600	6.07
FF	5131	41600	12.33
BRAM	36	50	72.00
IO	5	106	4.72
MMCM	1	5	20.00

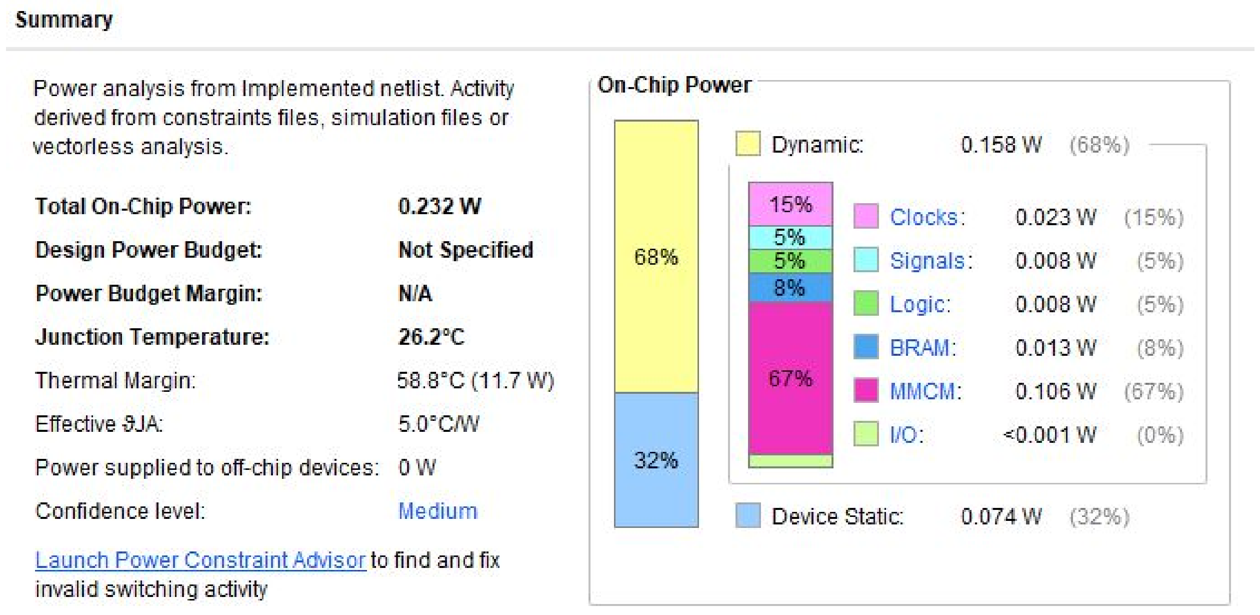


Distribution :



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (106)
topcontroller	4306	5131	108	2	1896	3723	583	36	5
cordicSynth (Sythesizer)	938	755	0	0	331	928	10	17	0
bram0 (blk_mem_gen_	14	6	0	0	9	14	0	2.5	0
bram1 (blk_mem_gen_	14	6	0	0	6	14	0	2.5	0
clkgenerator (clk_wiz_(	0	0	0	0	0	0	0	0	0
coefficient0 (blk_mem_	13	6	0	0	8	13	0	2.5	0
coefficient1 (blk_mem_	12	6	0	0	9	12	0	2.5	0
cordic0 (cordic_0)	261	266	0	0	90	256	5	0	0
cordic1 (cordic_0_HD3	261	266	0	0	88	256	5	0	0
mult0 (mult_gen_0)	72	16	0	0	21	72	0	0	0
mult1 (mult_gen_0_HD	72	16	0	0	21	72	0	0	0
phase (blk_mem_gen_	24	12	0	0	8	24	0	2.5	0
sum (c_addsub_0)	16	16	0	0	4	16	0	0	0
tone (blk_mem_gen_1	23	8	0	0	10	23	0	4.5	0
dbg_hub (dbg_hub)	442	727	0	0	240	418	24	0	0
ila (ila_0)	2826	3558	108	2	1306	2277	549	19	0
uartuut (uart)	89	73	0	0	34	89	0	0	0

Power estimate :



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

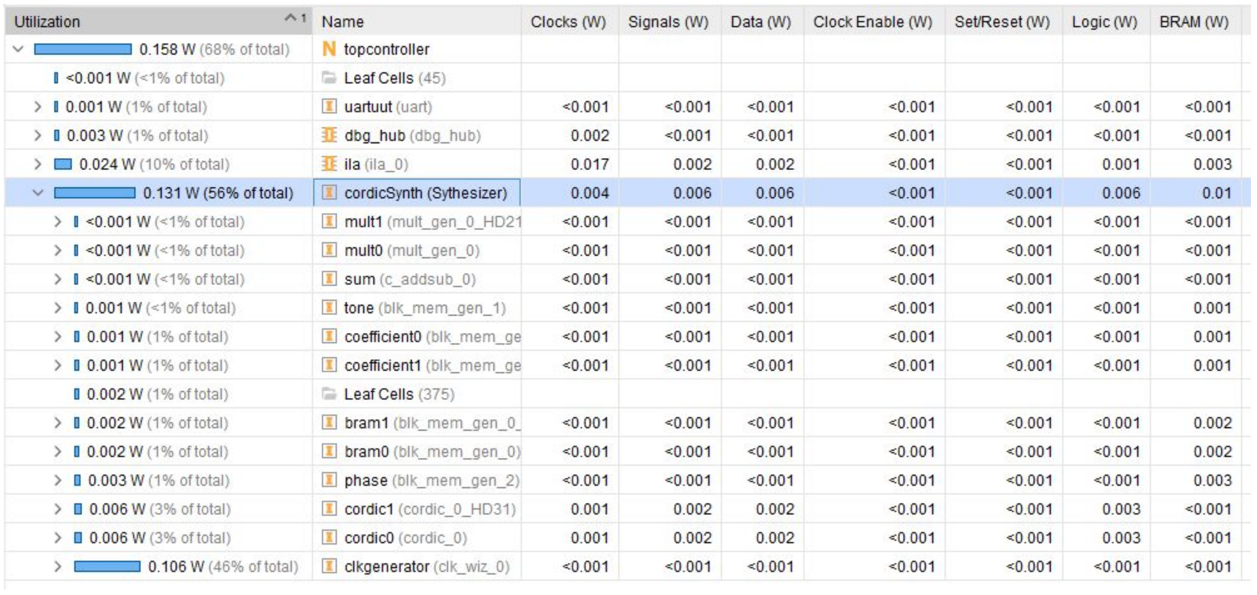
Total On-Chip Power:	**0.232 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26.2°C**
Thermal Margin:	58.8°C (11.7 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

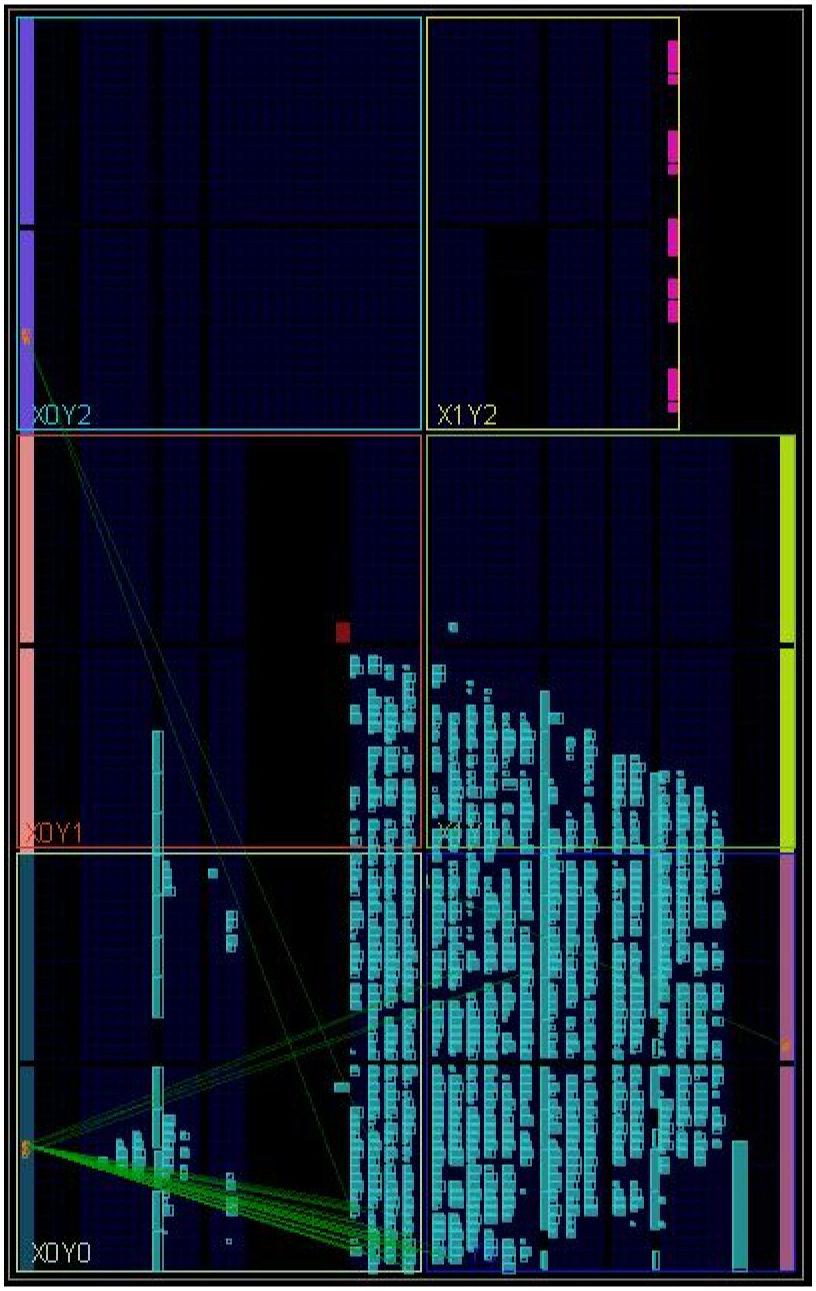
Dynamic:	0.158 W	(68%)
Clocks:	0.023 W	(15%)
Signals:	0.008 W	(5%)
Logic:	0.008 W	(5%)
BRAM:	0.013 W	(8%)
MMCM:	0.106 W	(67%)
I/O:	<0.001 W	(0%)
Device Static:	0.074 W	(32%)

Distribution :



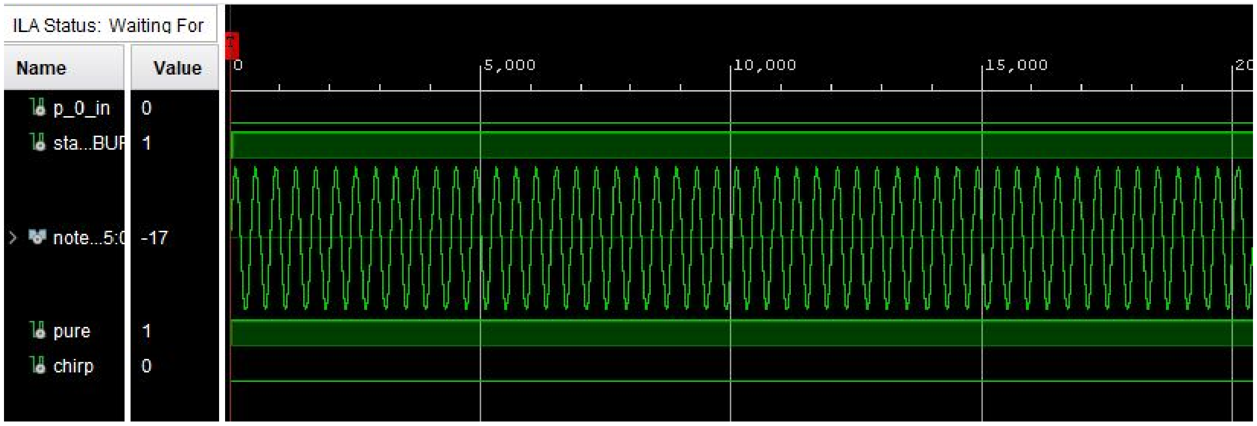
Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)
0.158 W (68% of total)	topcontroller							
<0.001 W (<1% of total)	Leaf Cells (45)							
0.001 W (1% of total)	uartuut (uart)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.003 W (1% of total)	dbg_hub (dbg_hub)	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.024 W (10% of total)	ila (ila_0)	0.017	0.002	0.002	<0.001	<0.001	0.001	0.003
0.131 W (56% of total)	cordicSynth (Sythesizer)	0.004	0.006	0.006	<0.001	<0.001	0.006	0.01
<0.001 W (<1% of total)	mult1 (mult_gen_0_HD21	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	mult0 (mult_gen_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	sum (c_addsub_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.001 W (<1% of total)	tone (blk_mem_gen_1)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.001
0.001 W (1% of total)	coefficient0 (blk_mem_ge	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.001
0.001 W (1% of total)	coefficient1 (blk_mem_ge	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.001
0.002 W (1% of total)	Leaf Cells (375)							
0.002 W (1% of total)	bram1 (blk_mem_gen_0_	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.002
0.002 W (1% of total)	bram0 (blk_mem_gen_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.002
0.003 W (1% of total)	phase (blk_mem_gen_2)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.003
0.006 W (3% of total)	cordic1 (cordic_0_HD31)	0.001	0.002	0.002	<0.001	<0.001	0.003	<0.001
0.006 W (3% of total)	cordic0 (cordic_0)	0.001	0.002	0.002	<0.001	<0.001	0.003	<0.001
0.106 W (46% of total)	clkgenerator (clk_wiz_0)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001

Placement of the logic :

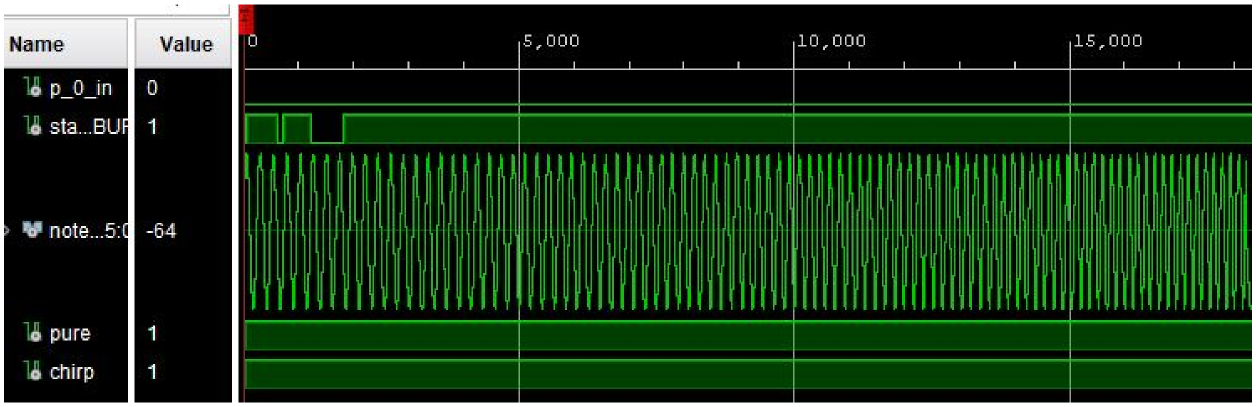


ILA outputs:

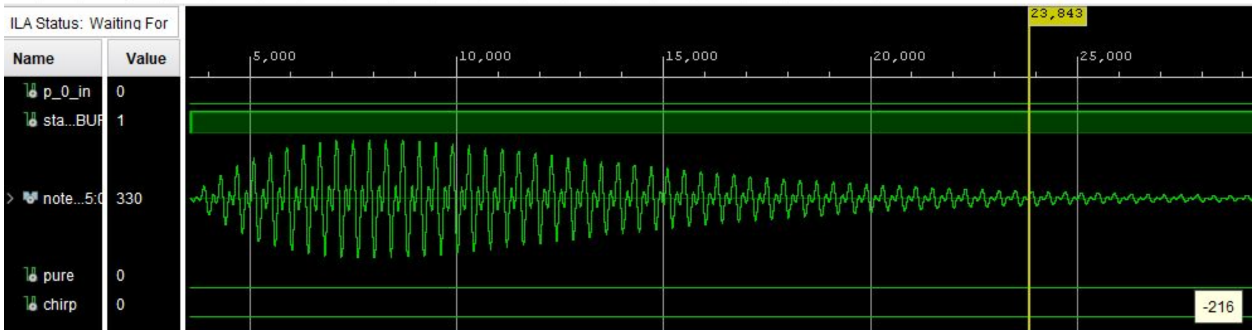
Pure sine wave of 82Hz :



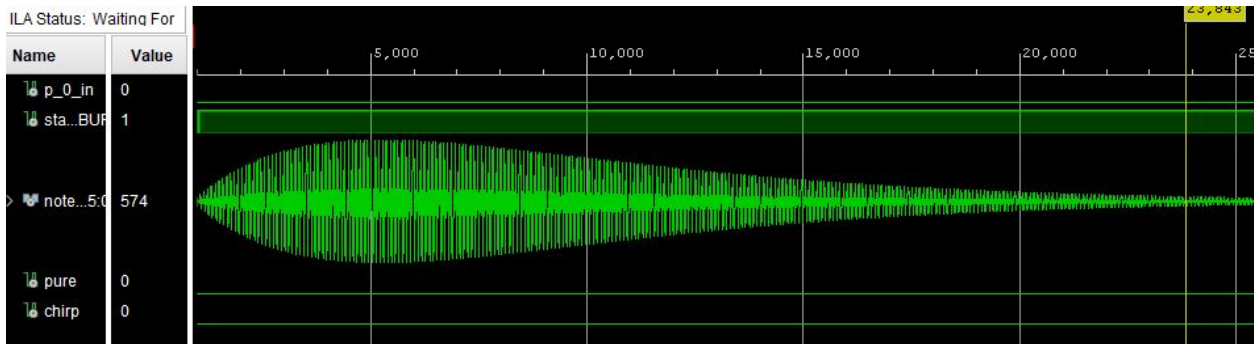
Chirp signal from 82Hz :



Lowest piano note implemented E2 :

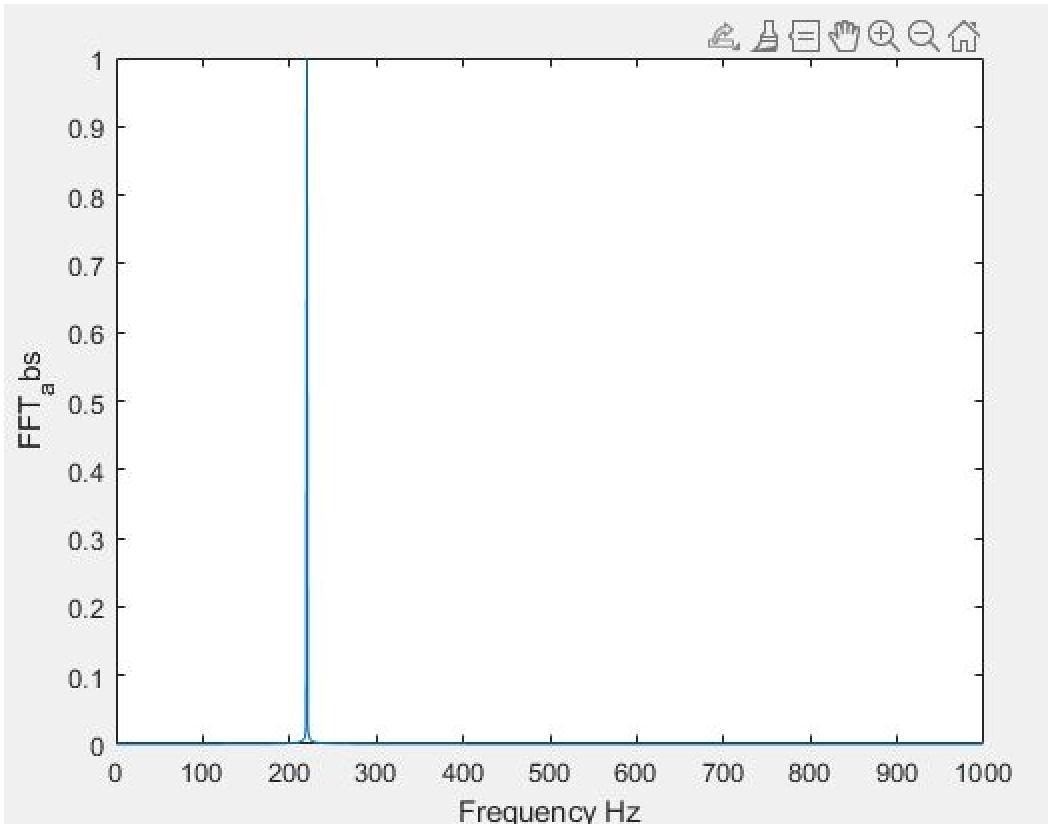


Highest piano note implemented B4 :

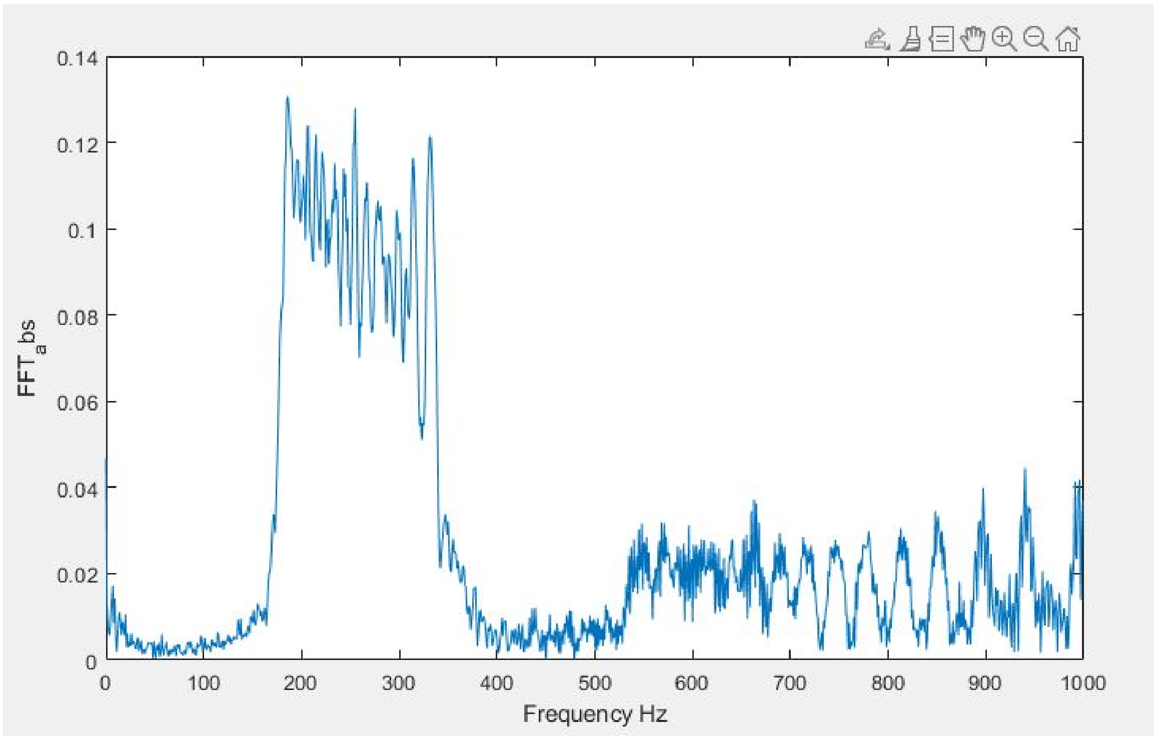


FFT of generated audio files :

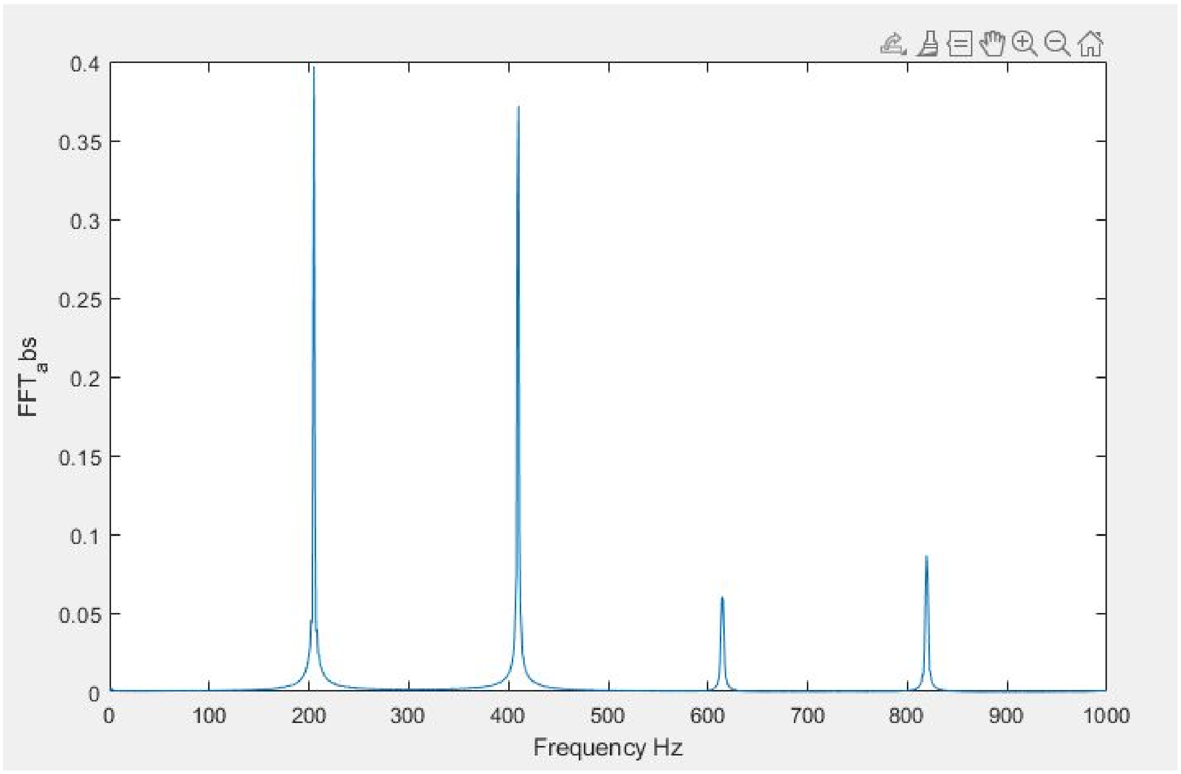
Pure sine wave of 220Hz :



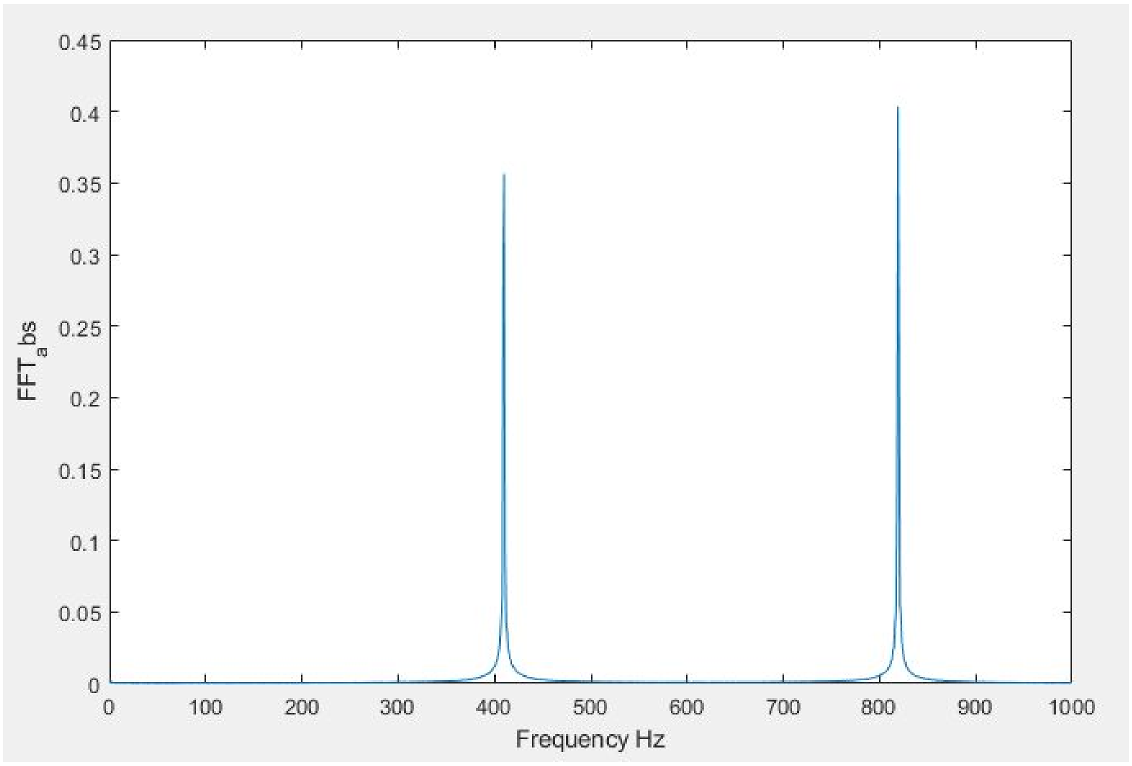
Chirp starting from 220:



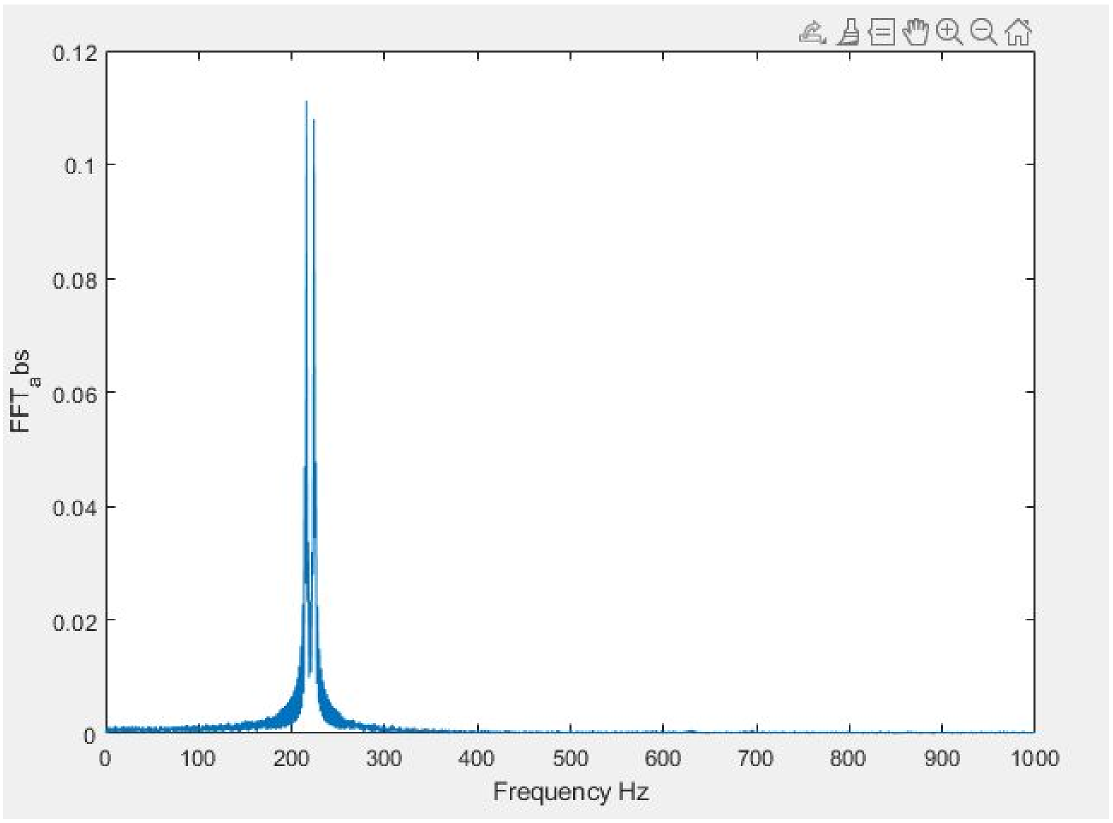
Piano note A3 :



Piano note with 400 Hz fundamental :



Beats of 220Hz :



References :

https://dsp.stackexchange.com/questions/46598/mathematical-equation-for-the-sound-wave-that-a-piano-makes

https://www.youtube.com/watch?t=254&v=ogFAHvYatWs&feature=youtu.be

https://en.wikipedia.org/wiki/Additive_synthesis#:~:text=Additive%20synthesis%20is%20a%20sound,or%20inharmonic%20partials%20or%20overtones.

https://amath.colorado.edu/pub/matlab/music/

https://www.mikroe.com/ebooks/programming-dspic-microcontrollers-in-pascal/introduction-uart-module

https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

https://pages.mtu.edu/~suits/notefreqs.html