**CompE-470**

* I declare that all material in this assignment is my own work except where there is clear reference to the work of others.
* I have read, understood and agree to the SDSU Policy on Plagiarism and Cheating on the university website at <http://go.sdsu.edu/student_affairs/srr/cheating-plagiarism.aspx> , the syllabus and the student-teacher contract for the consequences of plagiarism, including both academic and punitive sanctions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Remark\*. By submitting this assignment report electronically, you are deemed to have signed the declaration above.*

12/3/2021

Project Final Report

[Direct Digital Synthesis in Verilog]

Rylan Bumbasi RedID#: 822563190

**Content**

1 (High level implementation)

My project is DDS (Direct Digital Synthesis). The main module for my project is a NCO (numerically controlled oscillator) which essentially is a phase accumulator hooked up to a LUT (look-up table) that converts the phase accumulator values to an amplitude. The phase to amplitude conversion depends on which waveform you are trying to generate. The advantages of using a phase accumulator instead of a basic numerical counter is that you can easily change the frequency of the waveform generated through the phase accumulator itself. Having this ability makes the NCO one of the most efficient ways to convert digital values to an analog wave form. There are other methods of converting phase to amplitude such as the “Cordic algorithm” however, when preforming DDS using digital logic it is more efficient to use the LUT method previously described.

The DDS module I have designed will allow 3 sine waves to be generated via the LUT method and all 3 of the wave’s generated are combines to make a major chord based off of the frequency of the root note.

2. (Block Diagram)

Diagram

Description automatically generated

3.

(Description of top-level module)

My DDS module has 3 inputs and 6 outputs. The 3 inputs are RST, CLK and fcw (frequency-controlled word). RST input is used to initialize count registers within my module to a value of 0. CLK input is used to control clocked logic within my module. An example of clocked logic within my module would be when the Phase Accumulator accumulates on the positive edge of the CLK event. Fcw input is used to control the amount of phase added to the phase accumulator for a single CLK. Fcw is a 32-bit input since the phase accumulator has a value of 32-bits. The 6 outputs in my module are sine, sine\_2, sine\_3, triangular wave, sawtooth Wave, and chord Wave. All 6 of the outputs for this module are used to generate different types of wave forms at varying frequencies.

(Description of logic)

My DDS module will first declare multiple registers and wires used. 2 Registers are used to act as an array to hold the sine and triangular values from a .mem file. The .mem file is instantiated and is utilized like a LUT. Next, multiple wires are declared to be used to hold the index values of the LUTs. The logic of my module consists of a RST state and a NON-RST states. When the RST state is active, then all accumulator registers are set the 0 at a positive CLK edge event. When the RST state is not active, then all accumulator registers are incremented by a value determined by the input frequency-controlled word. Since the frequency-controlled word is controlled by an input value, the frequency that the waves are generated can vary based off of what the fcw input is set too. After all accumulator values are incremented, wires are used to hold the current value stored inside of the accumulator registers. Since our DDS module will have a 10-bit resolution, the index wires will only store the 10 most significant bits stored in the 32-bit accumulator registers. After the index values are stored in wires, the module will go into the several LUTs stored in the memory section and properly assign an output value based on the current index. In order to generate a chord wave, 3 sine waves were generated and the sum of the outputs of all 3 waves were sent to an independent chord output. The independent chord output will represent the wave form off all 3 sine waves occurring at the same exact time.

4. (Source Code)

5. The only toolset used to create my project was Xilinx Vivado

(DDS MODULE SOURCE CODE)

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

//

//

//////////////////////////////////////////////////////////////////////////////////

module DDS( input CLK,

input RST,

input [31:0] fcw, // Frequency Controlled Word to control the frequency of the DDs output

output [15 : 0] sine, // output will use a 16-bit Reg to contain sin val

output [15 : 0] sine\_2, // output will use a 16-bit Reg to contain sin val

output [15 : 0] sine\_3, // output will use a 16-bit Reg to contain sin val

output [3 : 0] triangleWave,

output [31 : 0] sawtoothWave,

output [44 : 0] chordWave

);

parameter N = 1024; // values of sine will be spread out among N (2 ^ 10)

// Declare Memory Arrays

reg [15 : 0] rom\_memory [N - 1: 0]; // Declare a 16 bit array for rom\_sine

reg [15 : 0] rom\_memory\_2 [N - 1: 0]; // Declare a 16 bit rom\_squareWave

reg [31:0] accumulator\_val; // reg to store current value of accumulator

reg [31:0] accumulator\_val\_2; // reg to store current value of accumulator

reg [31:0] accumulator\_val\_3; // reg to store current value of accumulator

reg [31 : 0] sawtooth\_reg; // reg to store value for sawtooth wave

wire [9:0] lut\_index; // Define a wire to hold the current index of the look up table

wire [9:0] lut\_index\_2; // Define a wire to hold the current index of the look up table

wire [9:0] lut\_index\_3; // Define a wire to hold the current index of the look up table

wire [44 : 0] chordWave\_wire; // Define a wire to hold the value of the chordWave

initial begin

$readmemh("sine.mem", rom\_memory);

$readmemh("squareWave.mem", rom\_memory\_2);

end

always@(posedge CLK) begin

// Initilize Count Values to 0 when RST = 1

if (RST) begin

accumulator\_val <= 0;

accumulator\_val\_2 <= 0;

accumulator\_val\_3 <= 0;

sawtooth\_reg <= 0;

end

else begin

// Incremenet acumulator value with fcw

sawtooth\_reg <= sawtooth\_reg + fcw;

accumulator\_val <= accumulator\_val + fcw;

accumulator\_val\_2 <= accumulator\_val\_2 + (fcw + (fcw \* 0.5));

accumulator\_val\_3 <= accumulator\_val\_3 + (fcw + (fcw \* 0.25));

end

end

// index sine lookup tables with accumulator values

assign lut\_index = accumulator\_val[31 : 22];

assign lut\_index\_2 = accumulator\_val\_2[31 : 22];

assign lut\_index\_3 = accumulator\_val\_3[31 : 22];

// assign current index in lut to sin outs

assign sine = rom\_memory[lut\_index];

assign sine\_2 = rom\_memory[lut\_index\_2];

assign sine\_3 = rom\_memory[lut\_index\_3];

// combine the output of all 3 sine waves to create a chord

assign chordWave\_wire = (sine) + (sine\_2) + (sine\_3);

// assign wave outputs to correct wires

assign chordWave = chordWave\_wire;

assign sawtoothWave = sawtooth\_reg;

assign triangleWave = rom\_memory\_2[lut\_index];

endmodule

**SINGLE WAVE TESTBENCH CODE**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

//

//

//////////////////////////////////////////////////////////////////////////////////

module tb\_DDS;

reg CLK, RST;

reg [31:0] fcw;

wire [15 : 0] sine\_out;

wire [15 : 0] sine\_out\_2;

wire [15 : 0] sine\_out\_3;

wire [44 : 0] chordWave\_out;

wire [31 : 0] sawtoothWave\_out;

wire [3 : 0] triangleWave\_out;

always #5 CLK <= ~CLK;

initial CLK <= 0;

DDS DUT (.CLK(CLK), .RST(RST), .fcw(fcw), .triangleWave(triangleWave\_out), .sawtoothWave(sawtoothWave\_out));

initial begin

RST = 1'b1; // Initilize values with RST input

fcw = 500000000;

#10 // 10 ns Later

RST = 1'b0;

#500 // 500 ns late

fcw = 250000000;

#500 // 500 ns Later

$finish;

end

endmodule

**CHORD WAVE TESTBENCH CODE**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

//

//

//////////////////////////////////////////////////////////////////////////////////

module tb\_DDS;

reg CLK, RST;

reg [31:0] fcw;

wire [15 : 0] sine\_out;

wire [15 : 0] sine\_out\_2;

wire [15 : 0] sine\_out\_3;

wire [44 : 0] chordWave\_out;

wire [31 : 0] sawtoothWave\_out;

wire [3 : 0] triangleWave\_out;

always #5 CLK <= ~CLK;

initial CLK <= 0;

DDS DUT (.CLK(CLK), .RST(RST), .fcw(fcw), .sine(sine\_out), .sine\_2(sine\_out\_2), .sine\_3(sine\_out\_3), .triangleWave(triangleWave\_out), .sawtoothWave(sawtoothWave\_out), .chordWave(chordWave\_out));

initial begin

RST = 1'b1; // Initilize values with RST input

fcw = 500000000;

#10 // 10 ns Later

RST = 1'b0;

$stop;

end

endmodule

**Memory Files**

**(SINE)**

8000

80c9

8192

825b

8324

83ed

84b6

857e

8647

8710

87d9

88a1

896a

8a32

8afb

8bc3

8c8b

8d53

8e1b

8ee3

8fab

9072

9139

9201

92c7

938e

9455

951b

95e1

96a7

976d

9833

98f8

99bd

9a82

9b47

9c0b

9ccf

9d93

9e56

9f19

9fdc

a09f

a161

a223

a2e5

a3a6

a467

a527

a5e8

a6a7

a767

a826

a8e5

a9a3

aa61

ab1f

abdc

ac98

ad55

ae10

aecc

af87

b041

b0fb

b1b5

b26e

b326

b3de

b496

b54d

b603

b6b9

b76f

b824

b8d8

b98c

ba3f

baf2

bba4

bc56

bd07

bdb7

be67

bf17

bfc5

c073

c121

c1cd

c279

c325

c3d0

c47a

c524

c5cc

c675

c71c

c7c3

c869

c90f

c9b3

ca57

cafb

cb9d

cc3f

cce0

cd81

ce20

cebf

cf5d

cffb

d097

d133

d1ce

d268

d302

d39a

d432

d4c9

d55f

d5f5

d689

d71d

d7b0

d842

d8d3

d964

d9f3

da82

db0f

db9c

dc28

dcb3

dd3d

ddc7

de4f

ded7

df5d

dfe3

e068

e0eb

e16e

e1f0

e271

e2f1

e370

e3ee

e46b

e4e8

e563

e5dd

e656

e6cf

e746

e7bc

e831

e8a6

e919

e98b

e9fc

ea6d

eadc

eb4a

ebb7

ec23

ec8e

ecf8

ed61

edc9

ee30

ee96

eefa

ef5e

efc1

f022

f083

f0e2

f140

f19d

f1f9

f254

f2ae

f307

f35e

f3b5

f40a

f45f

f4b2

f504

f555

f5a5

f5f3

f641

f68d

f6d8

f722

f76b

f7b3

f7fa

f83f

f884

f8c7

f909

f94a

f989

f9c8

fa05

fa41

fa7c

fab6

faee

fb26

fb5c

fb91

fbc5

fbf8

fc29

fc59

fc88

fcb6

fce3

fd0e

fd39

fd62

fd89

fdb0

fdd5

fdfa

fe1d

fe3e

fe5f

fe7e

fe9c

feb9

fed5

feef

ff09

ff21

ff37

ff4d

ff61

ff74

ff86

ff97

ffa6

ffb4

ffc1

ffcd

ffd8

ffe1

ffe9

fff0

fff5

fff9

fffd

fffe

ffff

fffe

fffd

fff9

fff5

fff0

ffe9

ffe1

ffd8

ffcd

ffc1

ffb4

ffa6

ff97

ff86

ff74

ff61

ff4d

ff37

ff21

ff09

feef

fed5

feb9

fe9c

fe7e

fe5f

fe3e

fe1d

fdfa

fdd5

fdb0

fd89

fd62

fd39

fd0e

fce3

fcb6

fc88

fc59

fc29

fbf8

fbc5

fb91

fb5c

fb26

faee

fab6

fa7c

fa41

fa05

f9c8

f989

f94a

f909

f8c7

f884

f83f

f7fa

f7b3

f76b

f722

f6d8

f68d

f641

f5f3

f5a5

f555

f504

f4b2

f45f

f40a

f3b5

f35e

f307

f2ae

f254

f1f9

f19d

f140

f0e2

f083

f022

efc1

ef5e

eefa

ee96

ee30

edc9

ed61

ecf8

ec8e

ec23

ebb7

eb4a

eadc

ea6d

e9fc

e98b

e919

e8a6

e831

e7bc

e746

e6cf

e656

e5dd

e563

e4e8

e46b

e3ee

e370

e2f1

e271

e1f0

e16e

e0eb

e068

dfe3

df5d

ded7

de4f

ddc7

dd3d

dcb3

dc28

db9c

db0f

da82

d9f3

d964

d8d3

d842

d7b0

d71d

d689

d5f5

d55f

d4c9

d432

d39a

d302

d268

d1ce

d133

d097

cffb

cf5d

cebf

ce20

cd81

cce0

cc3f

cb9d

cafb

ca57

c9b3

c90f

c869

c7c3

c71c

c675

c5cc

c524

c47a

c3d0

c325

c279

c1cd

c121

c073

bfc5

bf17

be67

bdb7

bd07

bc56

bba4

baf2

ba3f

b98c

b8d8

b824

b76f

b6b9

b603

b54d

b496

b3de

b326

b26e

b1b5

b0fb

b041

af87

aecc

ae10

ad55

ac98

abdc

ab1f

aa61

a9a3

a8e5

a826

a767

a6a7

a5e8

a527

a467

a3a6

a2e5

a223

a161

a09f

9fdc

9f19

9e56

9d93

9ccf

9c0b

9b47

9a82

99bd

98f8

9833

976d

96a7

95e1

951b

9455

938e

92c7

9201

9139

9072

8fab

8ee3

8e1b

8d53

8c8b

8bc3

8afb

8a32

896a

88a1

87d9

8710

8647

857e

84b6

83ed

8324

825b

8192

80c9

8000

7f36

7e6d

7da4

7cdb

7c12

7b49

7a81

79b8

78ef

7826

775e

7695

75cd

7504

743c

7374

72ac

71e4

711c

7054

6f8d

6ec6

6dfe

6d38

6c71

6baa

6ae4

6a1e

6958

6892

67cc

6707

6642

657d

64b8

63f4

6330

626c

61a9

60e6

6023

5f60

5e9e

5ddc

5d1a

5c59

5b98

5ad8

5a17

5958

5898

57d9

571a

565c

559e

54e0

5423

5367

52aa

51ef

5133

5078

4fbe

4f04

4e4a

4d91

4cd9

4c21

4b69

4ab2

49fc

4946

4890

47db

4727

4673

45c0

450d

445b

43a9

42f8

4248

4198

40e8

403a

3f8c

3ede

3e32

3d86

3cda

3c2f

3b85

3adb

3a33

398a

38e3

383c

3796

36f0

364c

35a8

3504

3462

33c0

331f

327e

31df

3140

30a2

3004

2f68

2ecc

2e31

2d97

2cfd

2c65

2bcd

2b36

2aa0

2a0a

2976

28e2

284f

27bd

272c

269b

260c

257d

24f0

2463

23d7

234c

22c2

2238

21b0

2128

20a2

201c

1f97

1f14

1e91

1e0f

1d8e

1d0e

1c8f

1c11

1b94

1b17

1a9c

1a22

19a9

1930

18b9

1843

17ce

1759

16e6

1674

1603

1592

1523

14b5

1448

13dc

1371

1307

129e

1236

11cf

1169

1105

10a1

103e

fdd

f7c

f1d

ebf

e62

e06

dab

d51

cf8

ca1

c4a

bf5

ba0

b4d

afb

aaa

a5a

a0c

9be

972

927

8dd

894

84c

805

7c0

77b

738

6f6

6b5

676

637

5fa

5be

583

549

511

4d9

4a3

46e

43a

407

3d6

3a6

377

349

31c

2f1

2c6

29d

276

24f

22a

205

1e2

1c1

1a0

181

163

146

12a

110

f6

de

c8

b2

9e

8b

79

68

59

4b

3e

32

27

1e

16

f

a

6

2

1

0

1

2

6

a

f

16

1e

27

32

3e

4b

59

68

79

8b

9e

b2

c8

de

f6

110

12a

146

163

181

1a0

1c1

1e2

205

22a

24f

276

29d

2c6

2f1

31c

349

377

3a6

3d6

407

43a

46e

4a3

4d9

511

549

583

5be

5fa

637

676

6b5

6f6

738

77b

7c0

805

84c

894

8dd

927

972

9be

a0c

a5a

aaa

afb

b4d

ba0

bf5

c4a

ca1

cf8

d51

dab

e06

e62

ebf

f1d

f7c

fdd

103e

10a1

1105

1169

11cf

1236

129e

1307

1371

13dc

1448

14b5

1523

1592

1603

1674

16e6

1759

17ce

1843

18b9

1930

19a9

1a22

1a9c

1b17

1b94

1c11

1c8f

1d0e

1d8e

1e0f

1e91

1f14

1f97

201c

20a2

2128

21b0

2238

22c2

234c

23d7

2463

24f0

257d

260c

269b

272c

27bd

284f

28e2

2976

2a0a

2aa0

2b36

2bcd

2c65

2cfd

2d97

2e31

2ecc

2f68

3004

30a2

3140

31df

327e

331f

33c0

3462

3504

35a8

364c

36f0

3796

383c

38e3

398a

3a33

3adb

3b85

3c2f

3cda

3d86

3e32

3ede

3f8c

403a

40e8

4198

4248

42f8

43a9

445b

450d

45c0

4673

4727

47db

4890

4946

49fc

4ab2

4b69

4c21

4cd9

4d91

4e4a

4f04

4fbe

5078

5133

51ef

52aa

5367

5423

54e0

559e

565c

571a

57d9

5898

5958

5a17

5ad8

5b98

5c59

5d1a

5ddc

5e9e

5f60

6023

60e6

61a9

626c

6330

63f4

64b8

657d

6642

6707

67cc

6892

6958

6a1e

6ae4

6baa

6c71

6d38

6dfe

6ec6

6f8d

7054

711c

71e4

72ac

7374

743c

7504

75cd

7695

775e

7826

78ef

79b8

7a81

7b49

7c12

7cdb

7da4

7e6d

7f36

**Triangular Wave**0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0000

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

0001

7. (Simulation Results)

Chart

Description automatically generated

The objective of my first simulation was to ensure that I was able to change the frequency of the generated waves by changing the input value of the frequency-controlled word inputted into my module. At time 500ns, the value of the frequency-controlled word was halved and you can observe that the frequency that the waves were being generated got halved at the time instance the value of the fcw changed. This indicates that the fcw is successfully varying the frequency of the generated waves.

A picture containing graphical user interface

Description automatically generated(Simulation Results Cont.)

The result of this testbench simulation was to test to see of my module was successfully generating a waveform for a chord using 3 existing generated sine waves. As you can observe in the chord wave output, this simulation was successful in that it combines the values from all 3 sine graphs into one waveform to represent the wave if all 3 buttons were to be pressed at the same time.

12. What would you differently if you had more time?

If I had more time to work on this project, I would try to implement an ADSR envelope module. I would also have tried to simulate chords that contain more than 3 note presses.

13. Conclusion

In conclusion, most of my design goals were met for this project. Through this project I learned a lot about the process of designing and testing Verilog modules. Because of the nature of this project, I was also able to understand how to create and instantiate memory blocks directly into a Verilog module. In the future, I would like to attempt to add the features discussed in section 12 of this report.

14. Hours spent on this project (50 Hours).