PROJECT REPORT

Project D3: Numerically Controlled Oscillator (NCO)

Mentor : Dr. Chithra

Mentees : Vaibhav Meena & Shital Niras

OBJECTIVE

The objective of this project was to implement a Numerically Controlled Oscillator (NCO) using Verilog. The NCO is a digital circuit used to generate periodic waveforms such as sine, cosine, etc., whose frequency can be controlled using a digital word (phase increment). This project aimed to demonstrate digital frequency control by generating three sine waves with different frequencies using a shared sine lookup table (LUT).

TOOLS AND TECHNOLOGIES USED

- **Verilog HDL** – For designing the RTL (Register Transfer Level) implementation of the NCO module.
- **Icarus Verilog** – Used for compiling and simulating the Verilog code.
- **GTKWave** – A waveform viewer used to analyze and verify simulation results.
- **Visual Studio Code (VS Code)** – The code editor used for writing and managing the Verilog source files.
- **Python** – Used to generate the sine lookup table file (sine_lut.hex) with 256-point values.

WHAT IS AN NCO?

An **NCO** is a digital signal generator which uses a **phase accumulator** and a **waveform lookup table (LUT)** to produce waveforms. On every clock cycle, a fixed value called the **phase increment** is added to a register called the phase accumulator. The **most significant bits (MSBs)** of the accumulator are used to index into the LUT to generate the waveform.

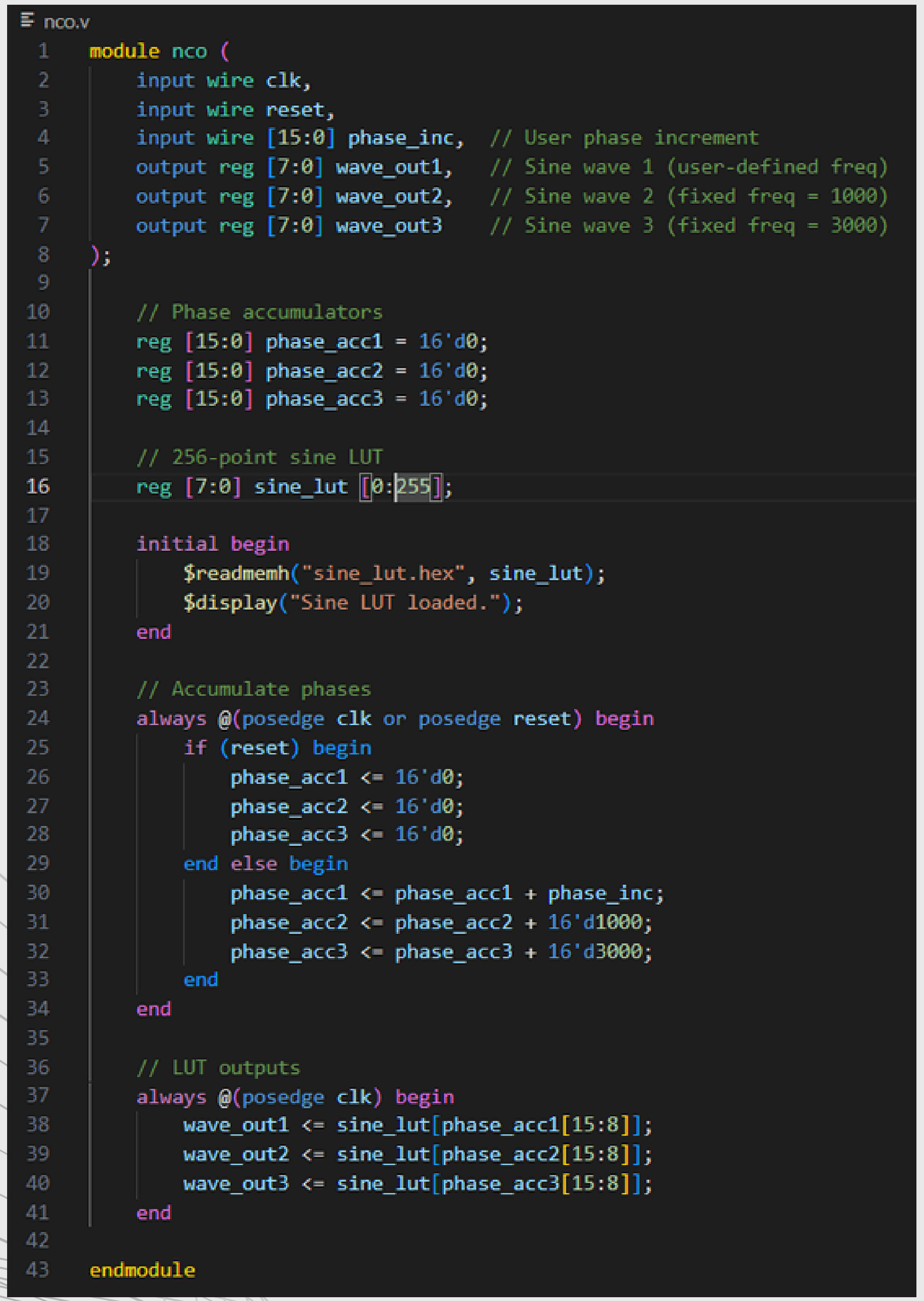
Frequency Equation:

$$f_{out} = \frac{\text{Phase Increment} \times f_{clk}}{2^N}$$

- f_clk is the system clock frequency
- N is the number of bits in the phase accumulator (16 in our case)

IMPLEMENTATION

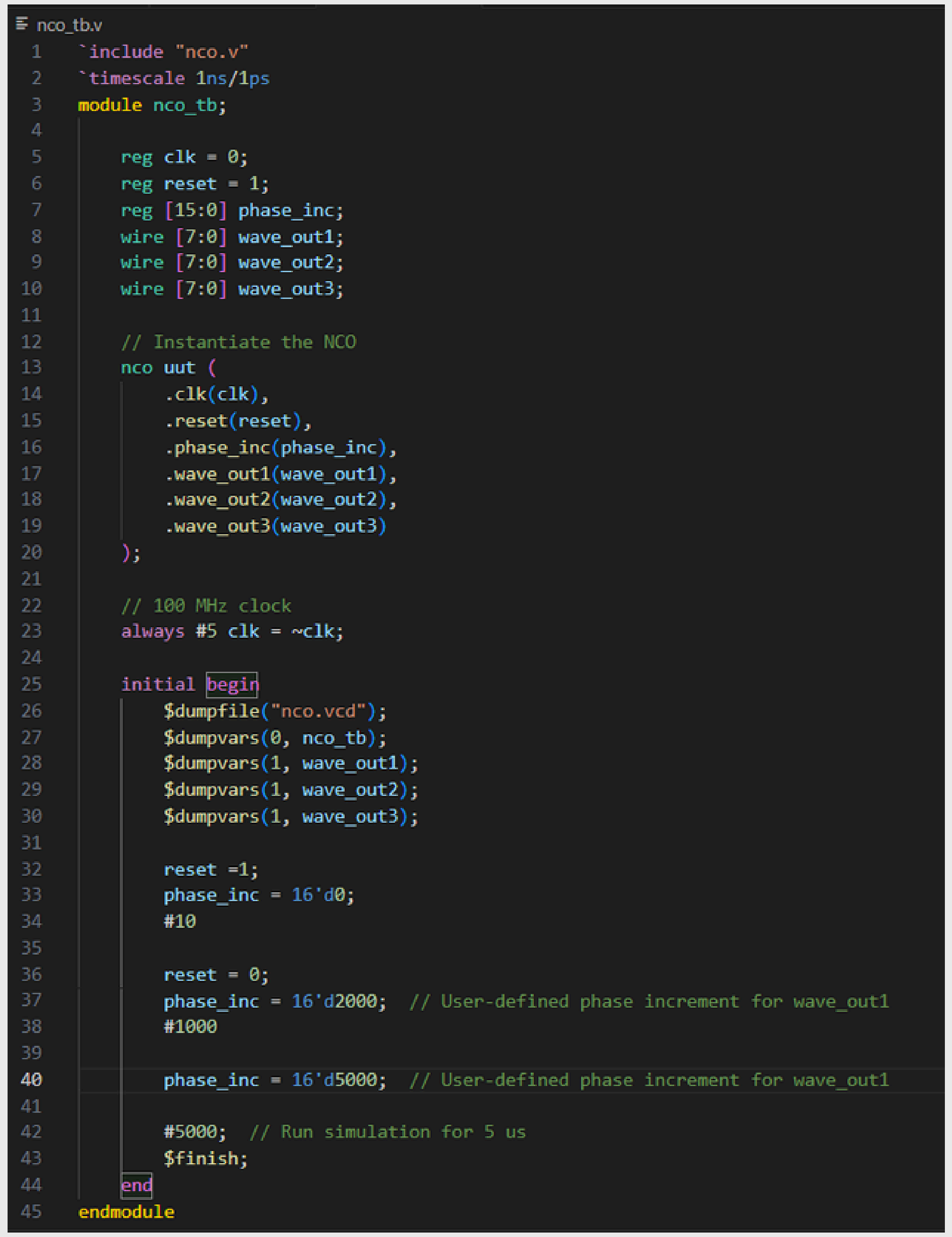
NCO Design (**nco.v**)



nco.v

```
module nco (
    input wire clk,
    input wire reset,
    input wire [15:0] phase_inc,  // User phase increment
    output reg [7:0] wave_out1,   // Sine wave 1 (user-defined freq)
    output reg [7:0] wave_out2,   // Sine wave 2 (fixed freq = 1000)
    output reg [7:0] wave_out3    // Sine wave 3 (fixed freq = 3000)
);

    // Phase accumulators
    reg [15:0] phase_acc1 = 16'd0;
    reg [15:0] phase_acc2 = 16'd0;
    reg [15:0] phase_acc3 = 16'd0;

    // 256-point sine LUT
    reg [7:0] sine_lut [0:255];

    initial begin
        $readmemh("sine_lut.hex", sine_lut);
        $display("Sine LUT loaded.");
    end

    // Accumulate phases
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            phase_acc1 <= 16'd0;
            phase_acc2 <= 16'd0;
            phase_acc3 <= 16'd0;
        end else begin
            phase_acc1 <= phase_acc1 + phase_inc;
            phase_acc2 <= phase_acc2 + 16'd1000;
            phase_acc3 <= phase_acc3 + 16'd3000;
        end
    end

    // LUT outputs
    always @(posedge clk) begin
        wave_out1 <= sine_lut[phase_acc1[15:8]];
        wave_out2 <= sine_lut[phase_acc2[15:8]];
        wave_out3 <= sine_lut[phase_acc3[15:8]];
    end

endmodule
```

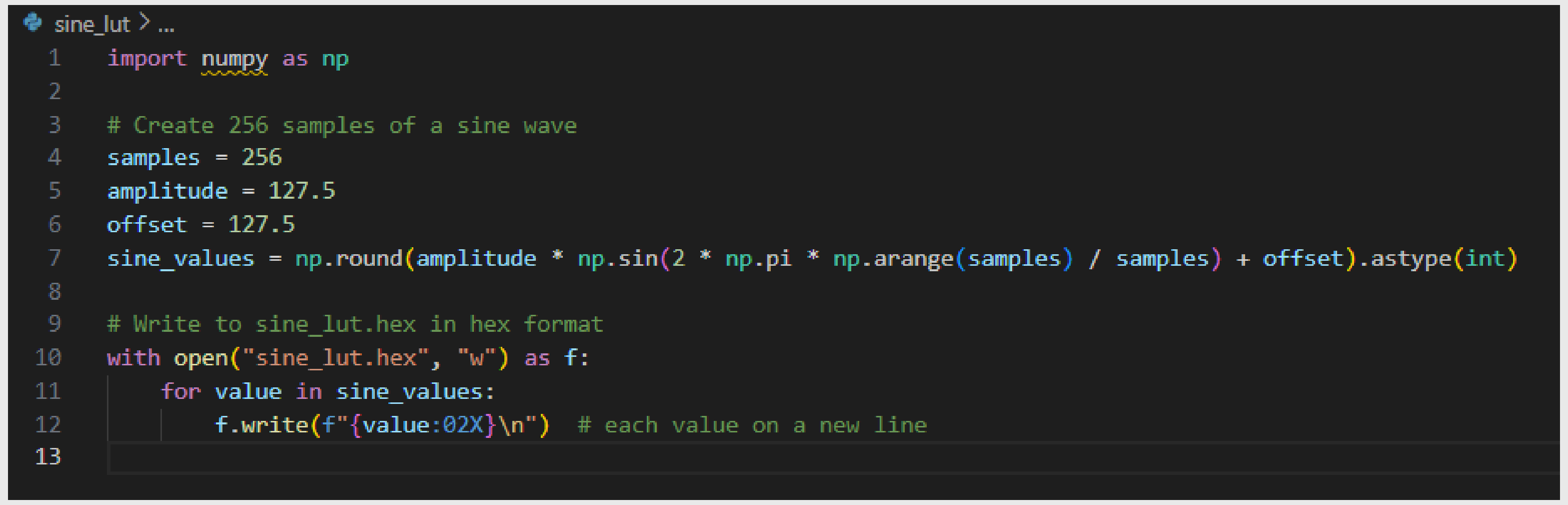
Testbench (**nco_tb.v**)



nco_tb.v

```
`include "nco.v"
`timescale 1ns/1ps
module nco_tb;

    reg clk = 0;
    reg reset = 1;
    reg [15:0] phase_inc;
    wire [7:0] wave_out1;
    wire [7:0] wave_out2;
    wire [7:0] wave_out3;

    // Instantiate the NCO
    nco uut (
        .clk(clk),
        .reset(reset),
        .phase_inc(phase_inc),
        .wave_out1(wave_out1),
        .wave_out2(wave_out2),
        .wave_out3(wave_out3)
    );

    // 100 MHz clock
    always #5 clk = ~clk;

    initial begin
        $dumpfile("nco.vcd");
        $dumpvars(0, nco_tb);
        $dumpvars(1, wave_out1);
        $dumpvars(1, wave_out2);
        $dumpvars(1, wave_out3);

        reset =1;
        phase_inc = 16'd0;
        #10

        reset = 0;
        phase_inc = 16'd2000;  // User-defined phase increment for wave_out1
        #1000

        phase_inc = 16'd5000;  // User-defined phase increment for wave_out1

        #5000;  // Run simulation for 5 us
        $finish;
    end
endmodule
```

Sine LUT File (**sine_lut.hex**) (generated using python)



sine_lut > ...

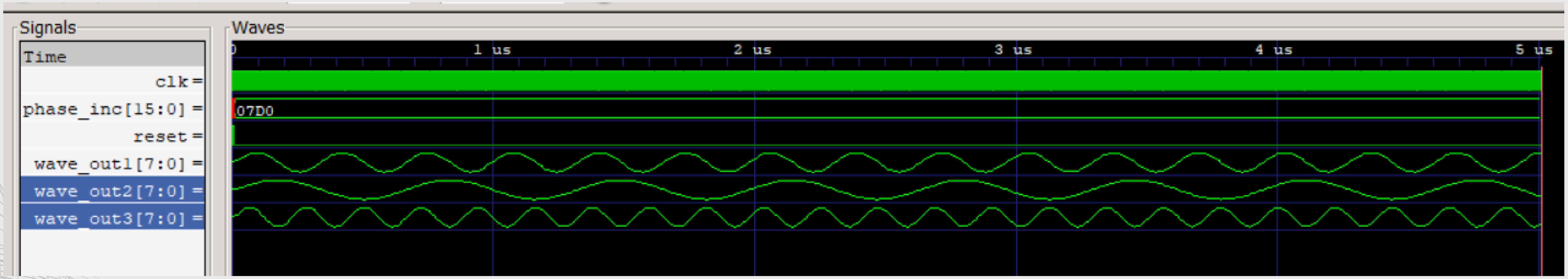
```
import numpy as np

# Create 256 samples of a sine wave
samples = 256
amplitude = 127.5
offset = 127.5
sine_values = np.round(amplitude * np.sin(2 * np.pi * np.arange(samples) / samples) + offset).astype(int)

# Write to sine_lut.hex in hex format
with open("sine_lut.hex", "w") as f:
    for value in sine_values:
        f.write(f"{value:02X}\n")  # each value on a new line

```

OBSERVATIONS

Simulation Waveform Output (**gtkwave**) showing 3 sine waveforms :

- **wave_out1**: Phase increment = 2000 → **Medium frequency**
- **wave_out2**: Phase increment= 1000 → **Low frequency**
- **wave_out3**: Phase increment = 3000 → **High frequency**

The sine waveform shape is clearly visible for all 3 signals.
All signals are generated using a shared 256-point LUT.
Output frequency increases with phase increment.



CONCLUSION

The project successfully demonstrates a working **Numerically Controlled Oscillator (NCO)** with frequency control. The relationship between phase increment and output frequency is observed clearly in simulations.

This project helped in:

- Understanding the internals of waveform synthesis
- Learning practical use of phase accumulators and lookup tables
- Using simulation tools (GTKWave, Icarus Verilog)

THANK YOU