Computer Architecture MP3 Report

Chang Jun Park

March 9, 2025

1 Introduction

This report describes the design and operation of a pipelined sine wave generator using a quarter-sine lookup table (LUT) approach on an FPGA. By storing only 128 samples corresponding to the first quarter of the sine wave and performing mirroring and inversion, the design reconstructs the full 512-sample (9-bit phase) sine wave with minimal memory usage. The quarter-sine data is read from a small memory module and transformed in real time to produce a 10-bit output.

2 Design Overview

The design includes two main modules:

1. **Top-level (`top.sv`)**: Manages the 9-bit phase counter, calculates addresses for the quarter-sine LUT, inverts the samples as needed for the 3rd/4th quarter, and drives the 10-bit output.
2. **Memory (`memory.sv`)**: Stores 128 samples of a quarter sine wave in a synchronous memory array. Data is loaded at synthesis or simulation time using `$readmemh`.

3 Circuit Operation

3.1 Phase Counter and Quarters

A 9-bit counter (`counter`) increments every clock cycle, giving a 512-step cycle for the sine wave. The two most significant bits `counter[8:7]` identify which quarter of the sine wave is currently in use:

$$
\begin{aligned}
00 &\rightarrow \text{Quarter 1 (0–127)} \\
01 &\rightarrow \text{Quarter 2 (128–255)} \\
10 &\rightarrow \text{Quarter 3 (256–383)} \\
11 &\rightarrow \text{Quarter 4 (384–511)}
\end{aligned}
$$

3.2 Memory Address Generation

To minimize storage, only the first quarter of the sine wave (128 samples) is stored in `quarter_sine.txt`. The `mem_address` logic reuses those samples for quarters 2, 3, and 4 by selectively mirroring:

```
Quarter 1 (00): mem_address = counter[6:0]
Quarter 2 (01): mem_address = 127 - counter[6:0]
Quarter 3 (10): mem_address = counter[6:0]
Quarter 4 (11): mem_address = 127 - counter[6:0]
```

This gives the correct “shapes” for the first and second quarters by reversing addresses for the second. The identical logic applies for the third and fourth quarters.

3.3 Data Inversion

For the third and fourth quarters, the sine wave goes negative. Instead of storing negative values, the design inverts the data in real time:

$$\texttt{output\_data} = \begin{cases} \texttt{mem\_data}, & \text{for quarters 0 and 1 (00, 01),} \\ (1023 - \texttt{mem\_data}), & \text{for quarters 2 and 3 (10, 11).} \end{cases}$$

Because each sample is 10 bits, 1023 (decimal `10'd1023`) is the maximum value, so `1023 - mem_data` provides a symmetric inversion.

3.4 Output Pins

The final 10-bit value is mapped to output pins in a specific bit order:

$$\{\_48b,\ \_45a,\ \_49a,\ \_3b,\ \_5a,\ \_0a,\ \_2a,\ \_4a,\ \_6a,\ \_9b\} = \text{output\_data}$$

4 Simulation Results

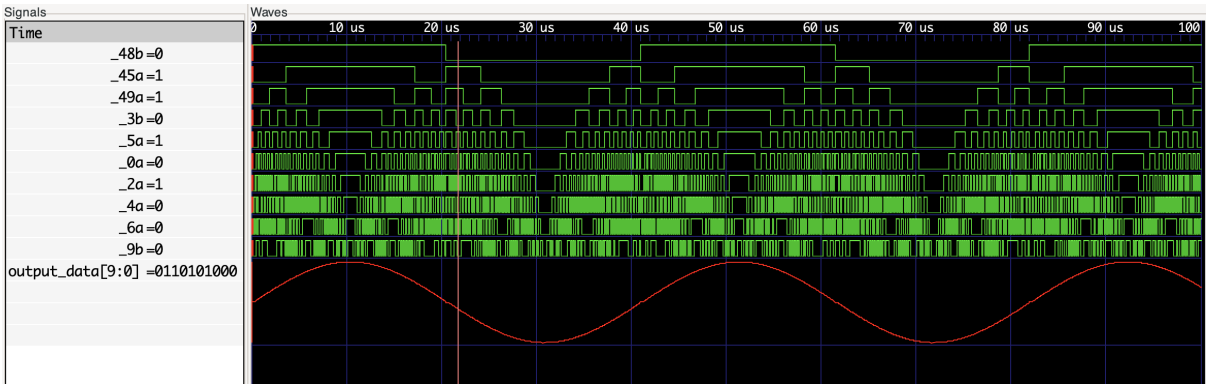

Figure 1: Sine Wave Simulation in GTKWave

The green digital signals (top) show the 10 output pins, while the red analog trace (bottom) is a digital-to-analog plot of the same 10-bit output. We can see how the amplitude follows the sine curve through each quarter-cycle, reflecting our read-and-inversion logic.

5 Oscilloscope Test

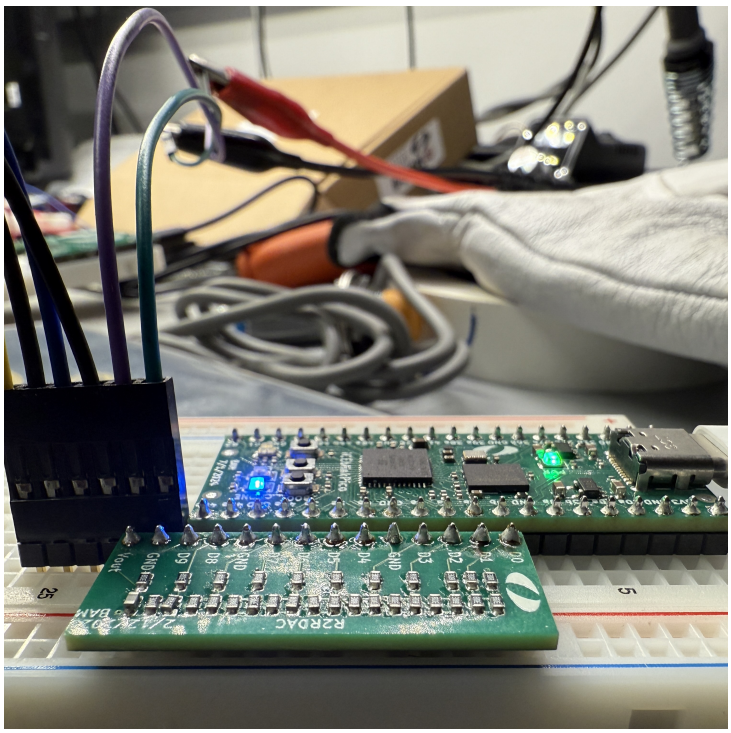

Figure 2: Testing Setup

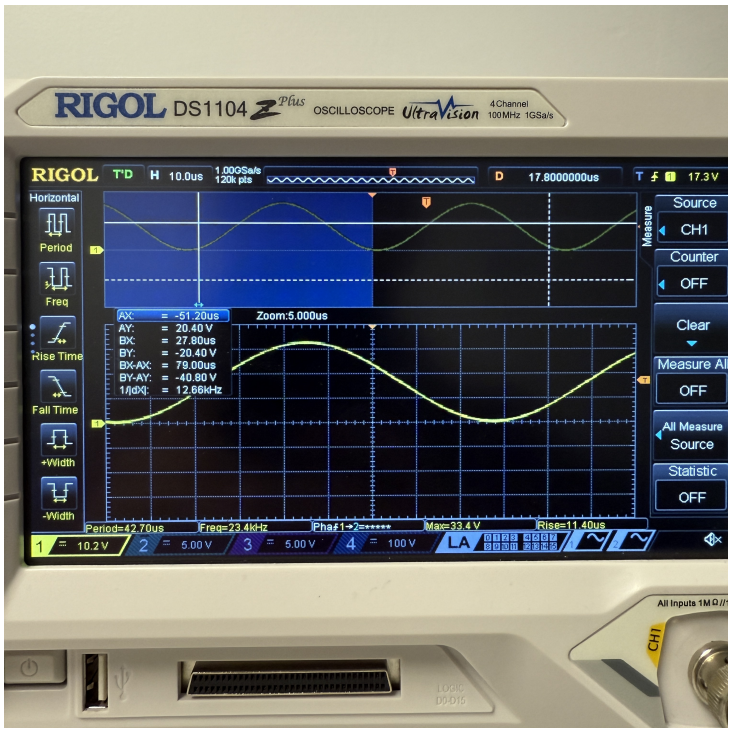

Figure 3: Oscilloscope Output of Sine Wave

6 Source Code

6.1 Top-level Module (top.sv)

top.sv

```
`include "memory.sv"

// MP3 top level module - Sine Wave Generator

module top(
    input logic     clk,
    output logic    _9b,    // D0
    output logic    _6a,    // D1
    output logic    _4a,    // D2
    output logic    _2a,    // D3
    output logic    _0a,    // D4
    output logic    _5a,    // D5
    output logic    _3b,    // D6
    output logic    _49a,   // D7
    output logic    _45a,   // D8
    output logic    _48b    // D9
);
    // 9 bits for 512 positions of the sine wave
    logic [8:0] counter = 0;

    logic [6:0] mem_address;
    logic [9:0] mem_data;
    logic [9:0] output_data;

    always_ff @(posedge clk) begin
        counter <= counter + 1;
    end

    always_ff @(posedge clk) begin
        case (counter[8:7])
            2'b00: mem_address <= counter[6:0];          // Quarter 1
            2'b01: mem_address <= 7'd127 - counter[6:0]; // Quarter 2
            2'b10: mem_address <= counter[6:0];          // Quarter 3
            2'b11: mem_address <= 7'd127 - counter[6:0]; // Quarter 4
        endcase
    end

    memory #(
        .INIT_FILE("quarter_sine.txt")
    ) sine_mem (
        .clk          (clk),
        .read_address (mem_address),
        .read_data    (mem_data)
    );

    always_ff @(posedge clk) begin
        case (counter[8:7])
            2'b00: output_data <= mem_data;
            2'b01: output_data <= mem_data;
            2'b10: output_data <= 10'd1023 - mem_data;
            2'b11: output_data <= 10'd1023 - mem_data;
        endcase
    end

    assign {_48b, _45a, _49a, _3b, _5a, _0a, _2a, _4a, _6a, _9b} = output_data;
endmodule
```

6.2 Memory Module (`memory.sv`)

memory.sv

```
// Memory
module memory #(
    parameter INIT_FILE = ""
)(
    input logic       clk,
    input logic [6:0] read_address,
    output logic [9:0] read_data
);

    // store 128 10-bit samples (first quarter of sine)
    logic [9:0] sample_memory [0:127];

    initial if (INIT_FILE) begin
        $readmemh(INIT_FILE, sample_memory);
    end

    always_ff @(posedge clk) begin
        read_data <= sample_memory[read_address];
    end

endmodule
```