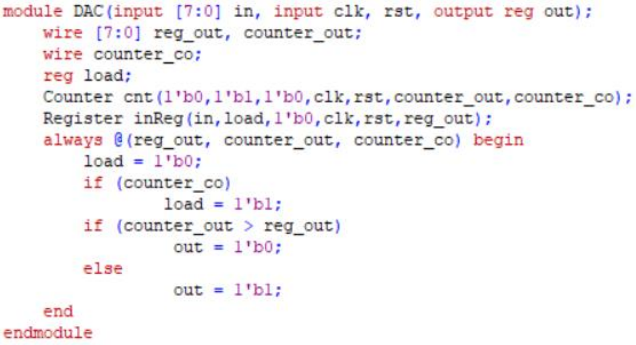
Experiment #3– Function Generator

Sana Sari Navaei 810199435
Maryam Jafarabadi 810199549

1.Digital to Analog conversion using PWM

1.



```
module DAC(input [7:0] in, input clk, rst, output reg out);
    wire [7:0] reg_out, counter_out;
    wire counter_co;
    reg load;
    Counter cnt(1'b0,1'b1,1'b0,clk,rst,counter_out,counter_co);
    Register inReg(in,load,1'b0,clk,rst,reg_out);
    always @(reg_out, counter_out, counter_co) begin
        load = 1'b0;
        if (counter_co)
                load = 1'b1;
        if (counter_out > reg_out)
                out = 1'b0;
        else
                out = 1'b1;
    end
endmodule
```

Fig. 1 Verilog code for DAC module

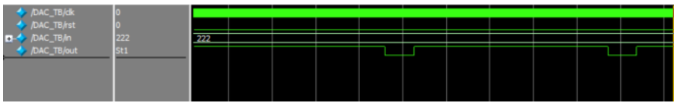


Fig. 2 when input is 222

When input is 222, T_{on} = 222 * clock duration and T_{off} = (256 – 222) * clock duration which are shown by the waveform.

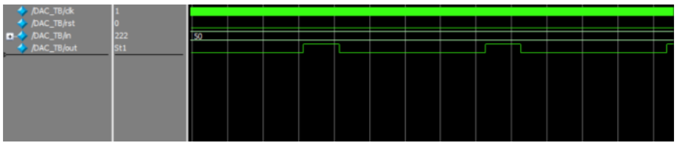


Fig. 3 when input is 50

When input is 50, T_{on} = 50 * clock duration and T_{off} = (256 – 50) * clock duration which are shown by the waveform.

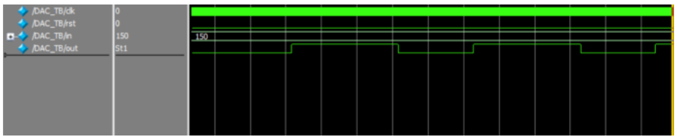


Fig. 4 when input is 150

When input is 150, T_{on} = 150 * clock duration and T_{off} = (256 – 150) * clock duration which are shown by the waveform.

2.Waveform Generator

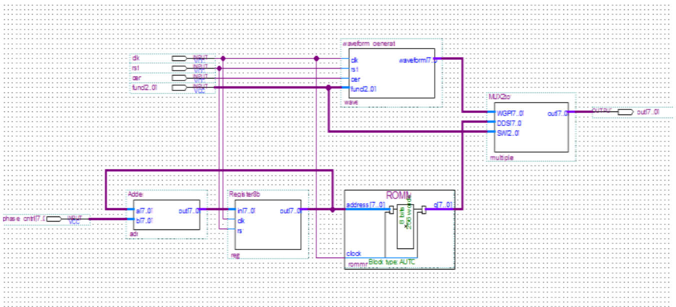
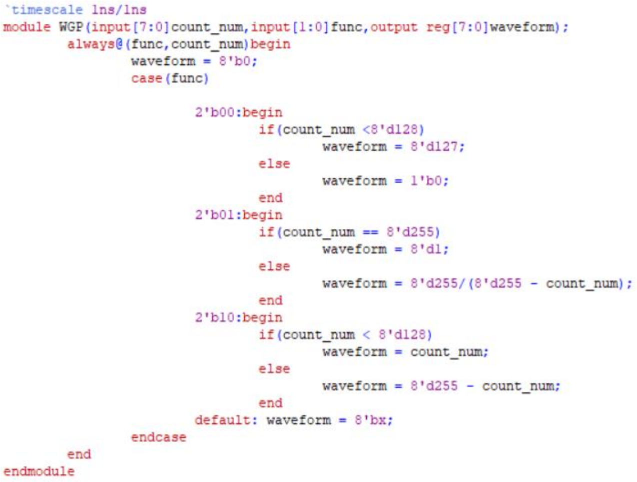


Fig. 5 Waveform generator



```
`timescale 1ns/1ns
module WGP(input[7:0]count_num,input[1:0]func,output reg[7:0]waveform);
        always@(func,count_num)begin
                waveform = 8'b0;
                case(func)

                        2'b00:begin
                                if(count_num <8'd128)
                                        waveform = 8'd127;
                                else
                                        waveform = 1'b0;
                                end
                        2'b01:begin
                                if(count_num == 8'd255)
                                        waveform = 8'd1;
                                else
                                        waveform = 8'd255/(8'd255 - count_num);
                                end
                        2'b10:begin
                                if(count_num < 8'd128)
                                        waveform = count_num;
                                else
                                        waveform = 8'd255 - count_num;
                                end
                        default: waveform = 8'bx;
                endcase
        end
endmodule
```

Fig. 6 waveform generator verilog

1.

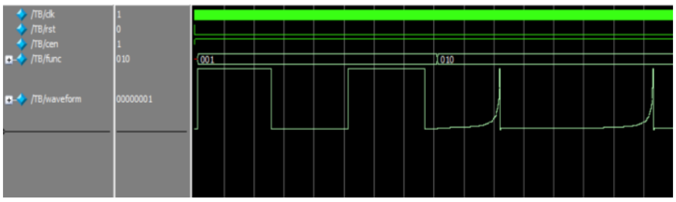


Fig. 7 waveform generator output

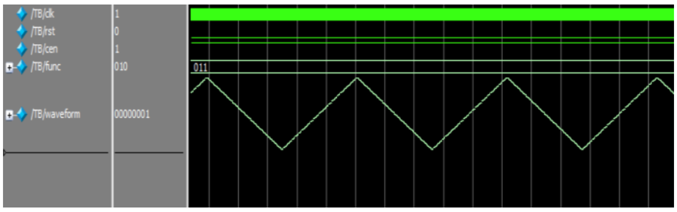
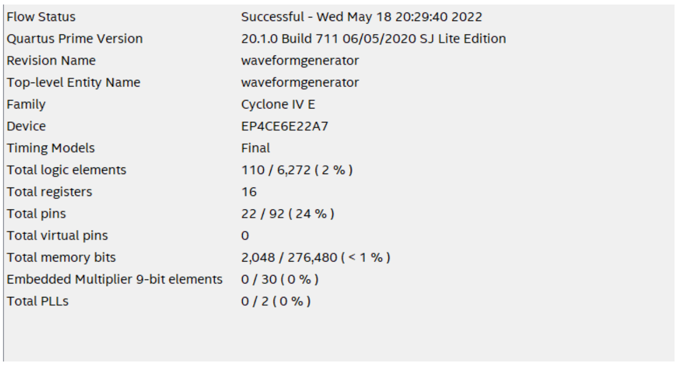


Fig. 8 waveform generator output

2.



Flow Status	Successful - Wed May 18 20:29:40 2022
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	waveformgenerator
Top-level Entity Name	waveformgenerator
Family	Cyclone IV E
Device	EP4CE6E22A7
Timing Models	Final
Total logic elements	110 / 6,272 (2 %)
Total registers	16
Total pins	22 / 92 (24 %)
Total virtual pins	0
Total memory bits	2,048 / 276,480 (< 1 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)

Fig. 9 synthesis summary

3.

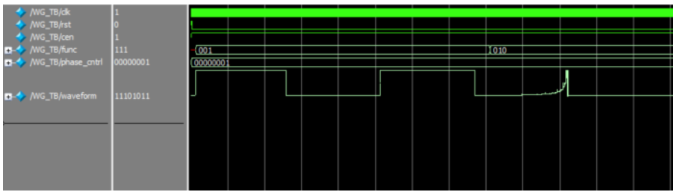


Fig. 10 Square and Reciprocal waveform

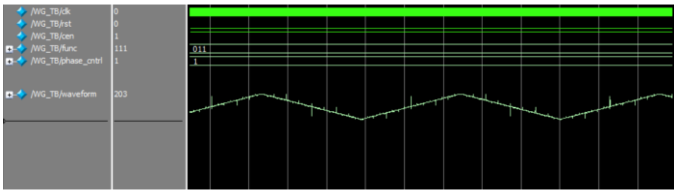


Fig. 11 Triangle waveform

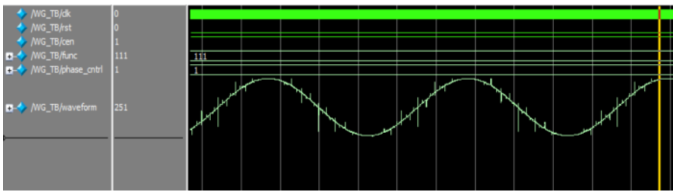


Fig. 12 Sine waveform

3.Frequency Selector

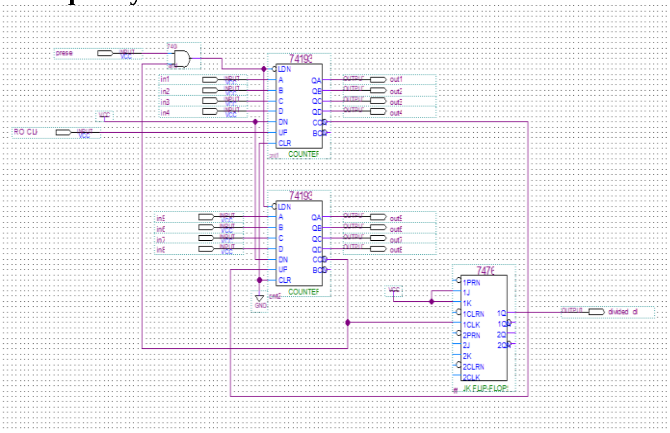


Fig. 13 Frequency Selector

4.

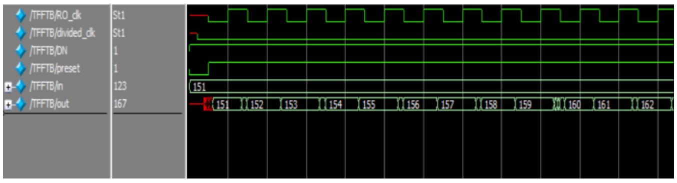


Fig. 14 parallel load = 151

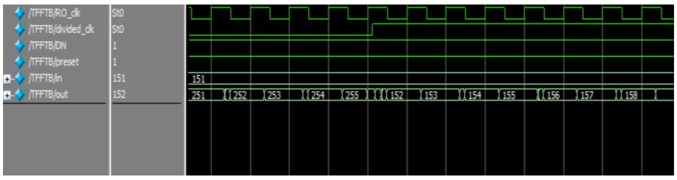


Fig. 15 parallel load = 151

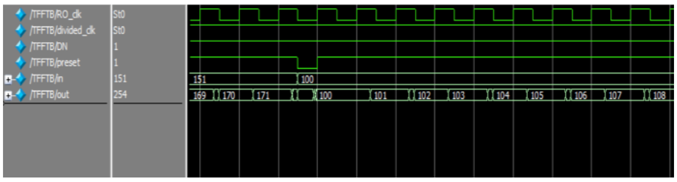


Fig. 16 parallel load = 100

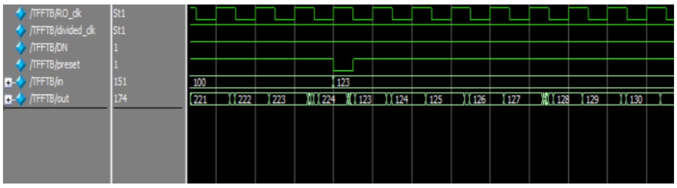


Fig. 17 parallel load = 123

5.

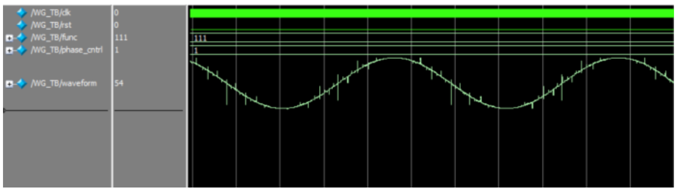


Fig. 18 phase_cntrl = 1

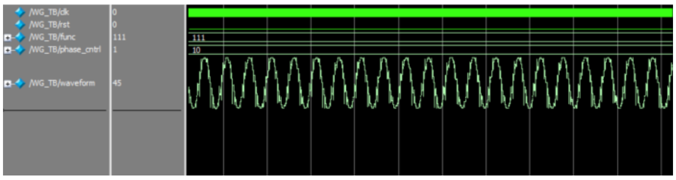


Fig. 19 phase_cntrl = 10

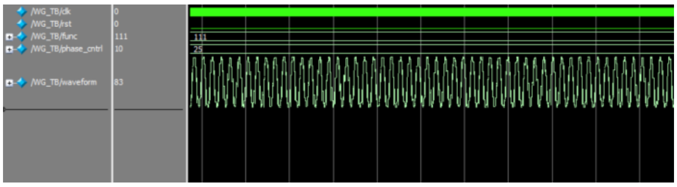


Fig. 20 phase_cntrl = 25

By increasing the number of phase cntrl some data from our Rom will be skipped and we loose more data (for instance if phase cntrl = 100 the result of our adder will be increase by 100 each time and as we see the output of our adder is being used for the address we want to access in the Rom so 100 data's in our Rom will be skipped)
As result our frequency becomes higher and the periodicity becomes lower (T = 1/f).

4.Amplitude Selector

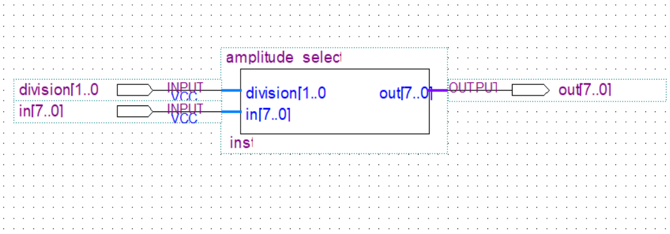
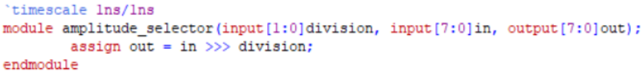


Fig. 21 amplitude selector



```
`timescale 1ns/1ns
module amplitude_selector(input[1:0]division, input[7:0]in, output[7:0]out);
        assign out = in >>> division;
endmodule
```

Fig. 22 amplitude selector verilog

2.

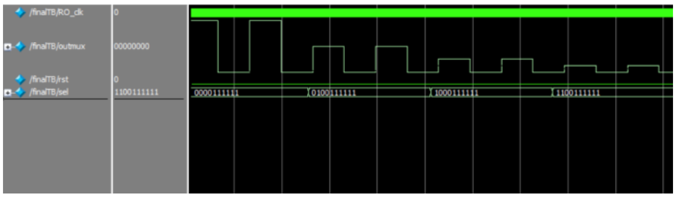


Fig. 23 Square with shift

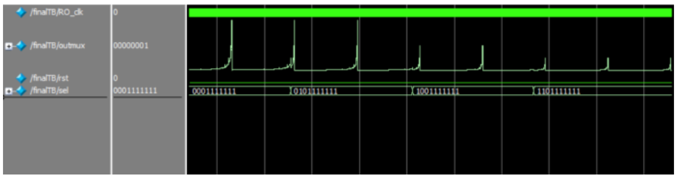


Fig. 24 Reciprocal with shift

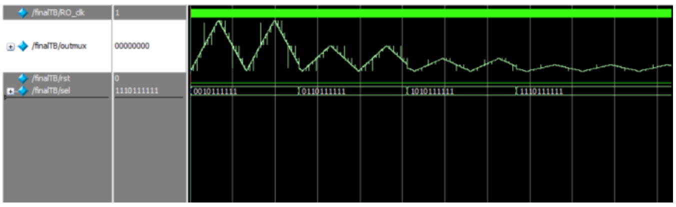


Fig. 25 Triangle with shift

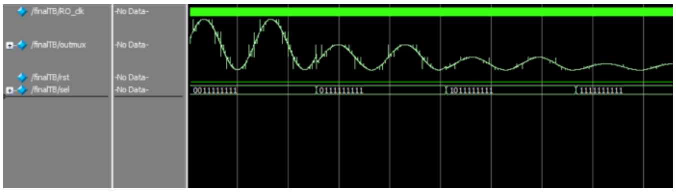


Fig. 26 Sine with shift

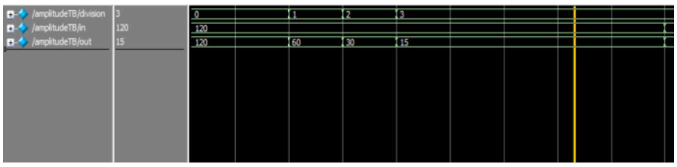


Fig. 27 when input is 120

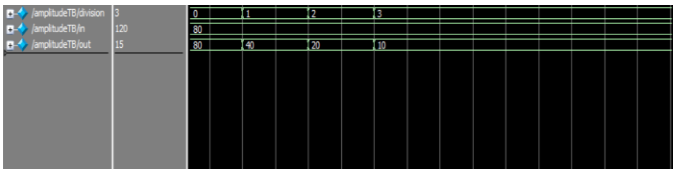


Fig. 28 when input is 80

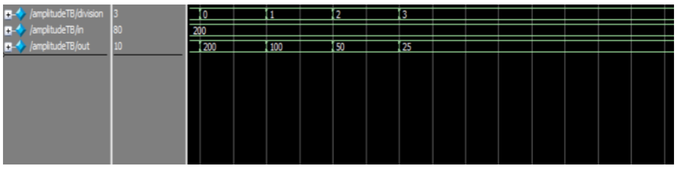


Fig. 29 when input is 200

5.The total design

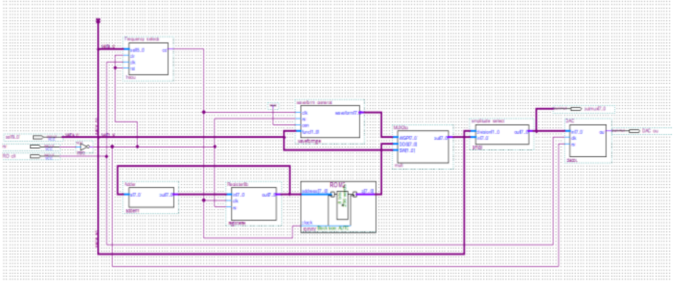


Fig. 30 Total design

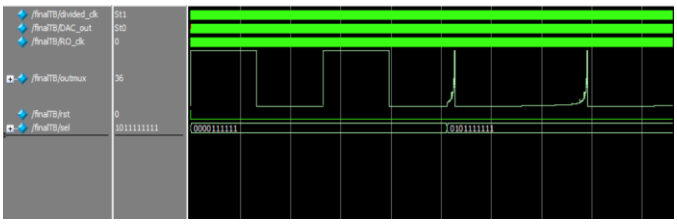


Fig. 31 waveform of square and reciprocal

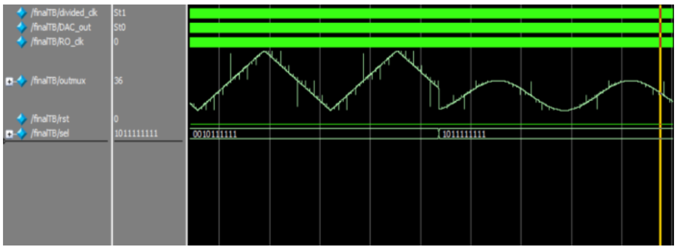


Fig. 32 waveform of triangle and sine

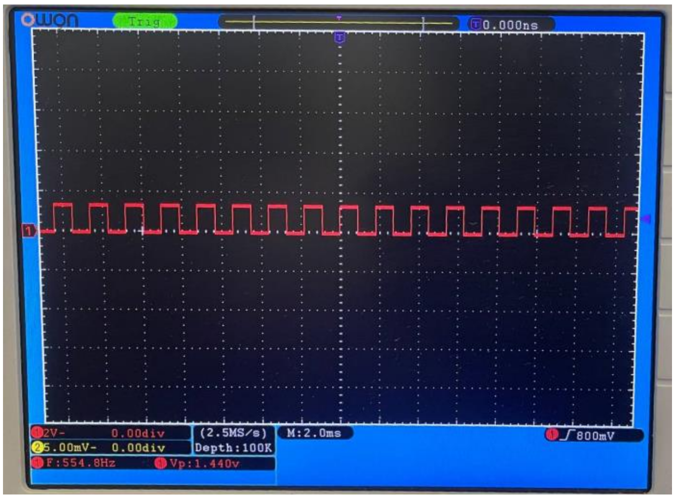


Fig. 33 Square waveform without shift

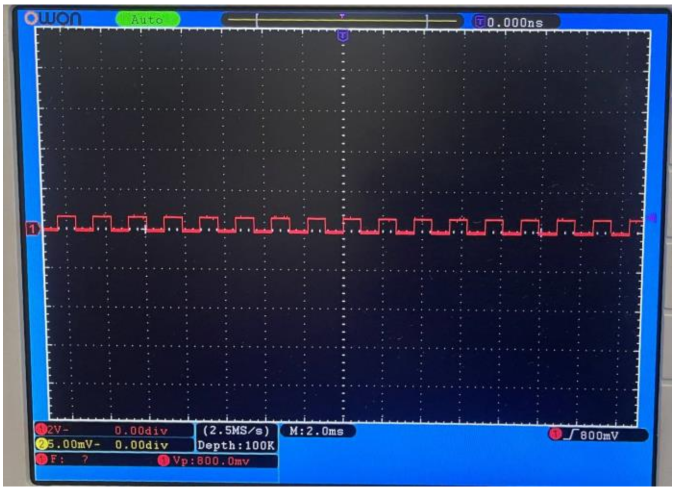


Fig. 34 Square waveform with one shift

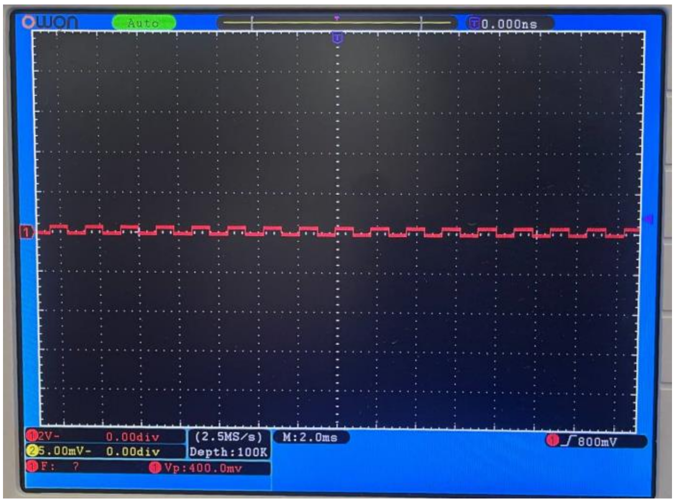


Fig. 35 Square waveform with two shifts

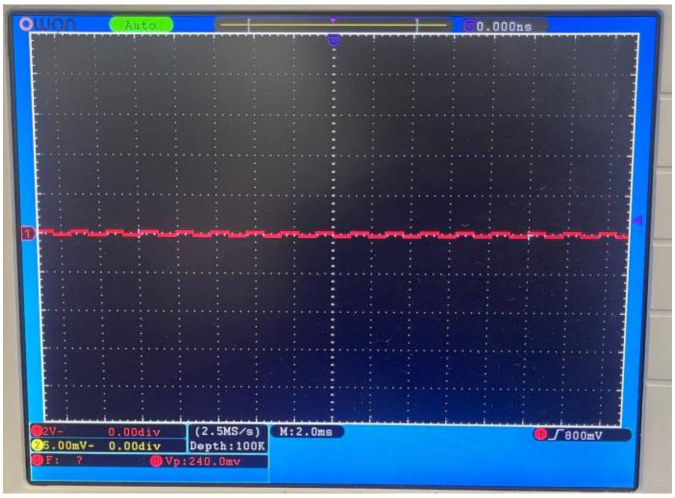


Fig. 36 Square waveform with three shifts

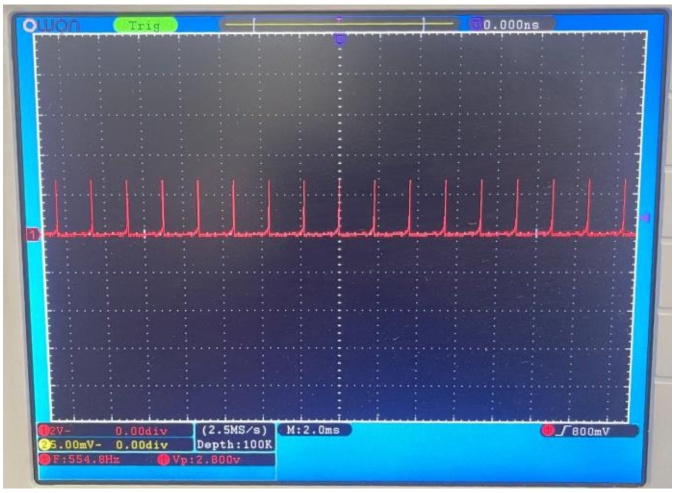


Fig. 37 Reciprocal waveform without shift

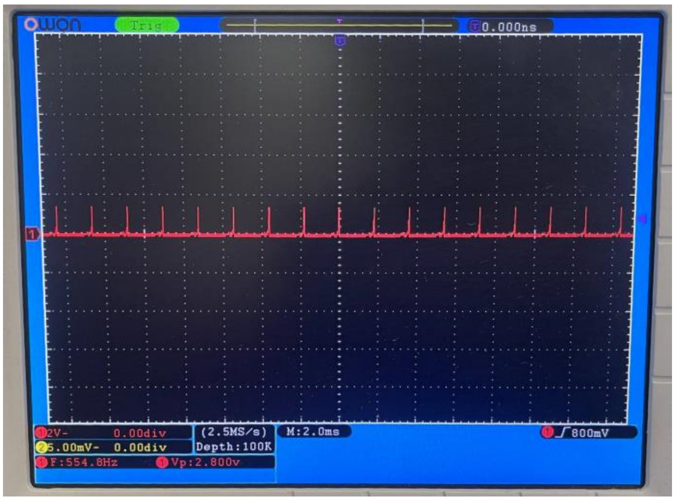


Fig. 38 Reciprocal waveform with one shift

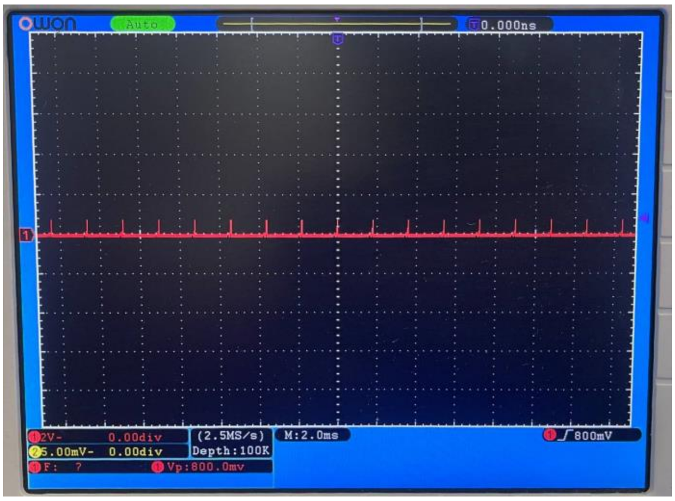


Fig. 39 Reciprocal waveform with two shifts

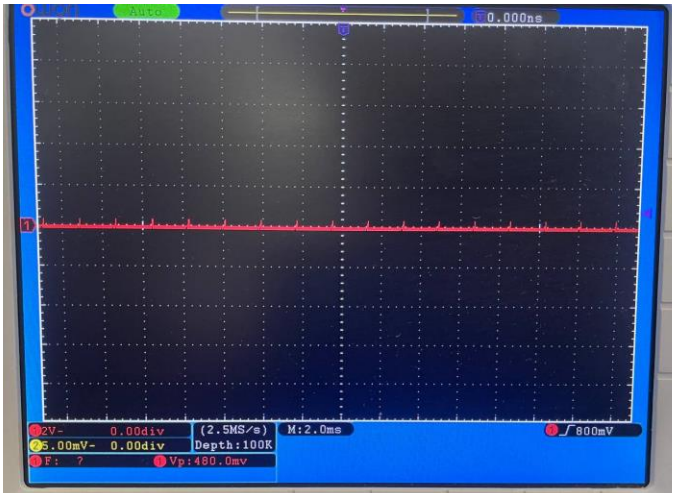


Fig. 40 Reciprocal waveform with three shifts

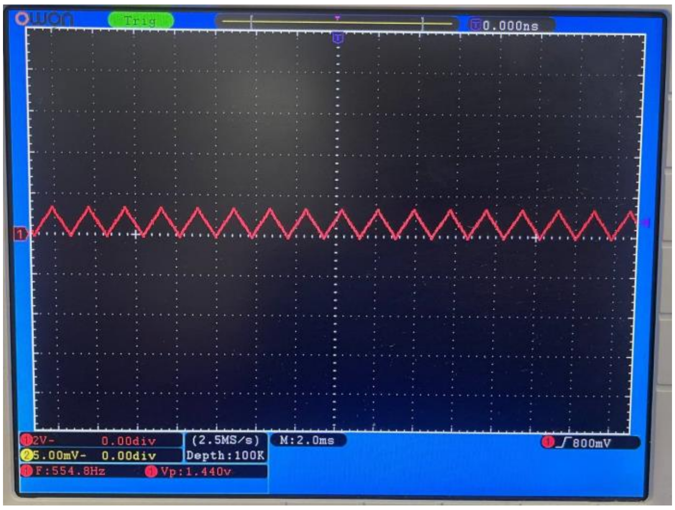


Fig. 41 Triangle waveform without shift

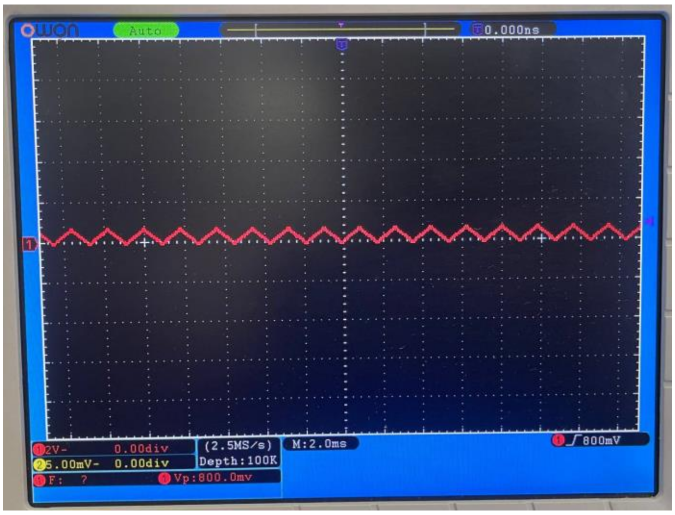


Fig. 42 Triangle waveform with one shift

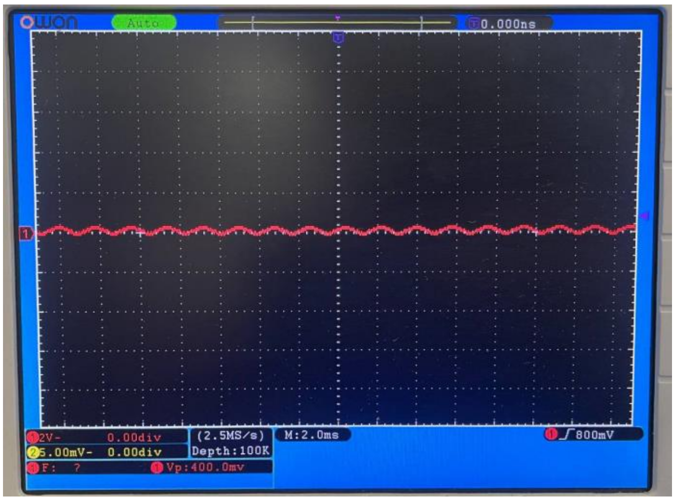


Fig. 43 Triangle waveform with two shifts

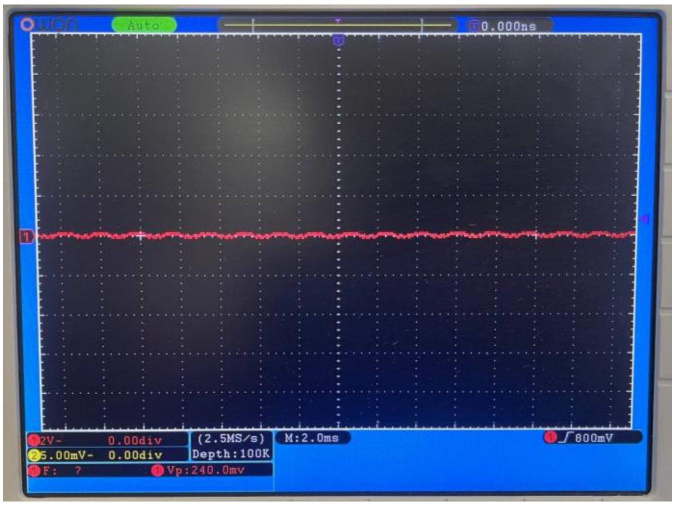


Fig. 44 Triangle waveform with three shifts

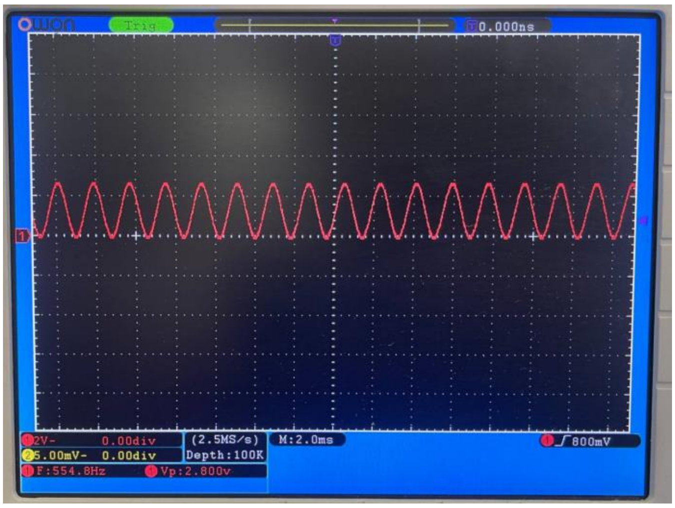


Fig. 45 Sine waveform without shift

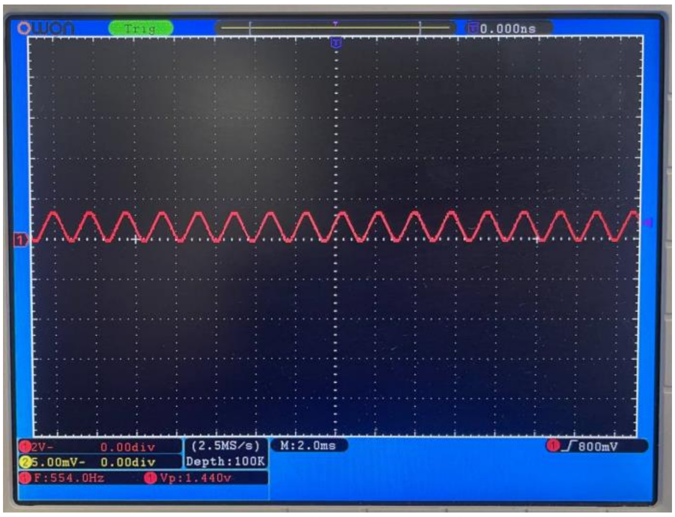


Fig. 46 Sine waveform with one shift

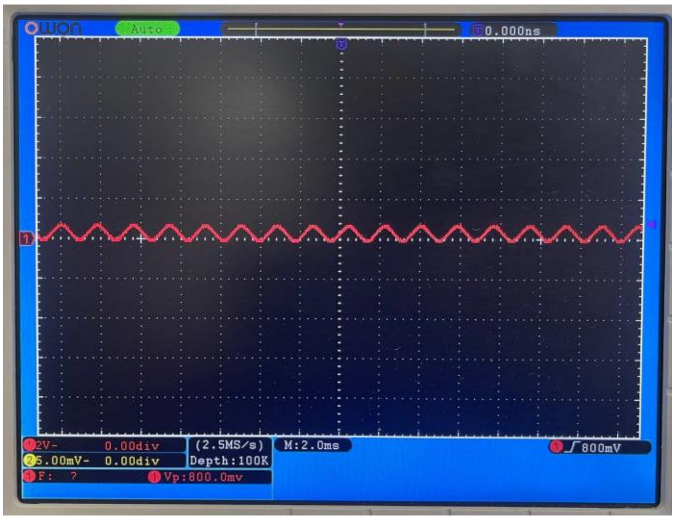


Fig. 47 Sine waveform with two shifts

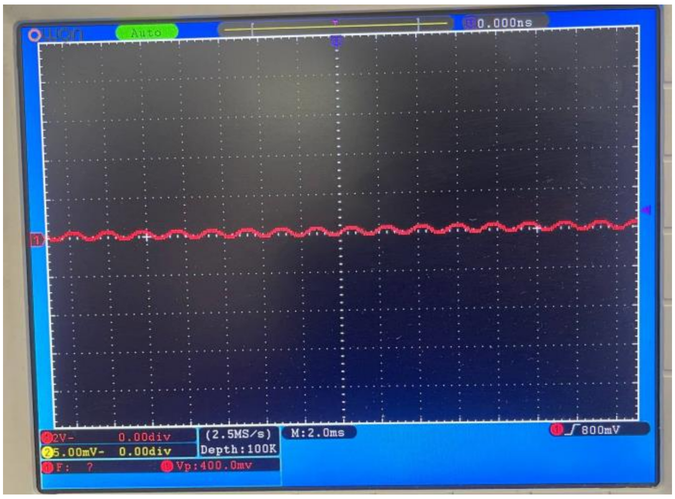


Fig. 48 Sine waveform with three shifts