میان‌ترم *DSD*

سوال 4

متین باقری **402105727**

فرکانس کاری بسیار از FPGA ها 50M Hz است، به همین دلیل در این پیاده سازی نیز ما فرکانس کاری مدار را 50M Hz در نظر گرفتیم. فرکانس ramp خواسته شده 1K Hz است، یعنی هر ramp در 1ms ایجاد می‌شود و این بازه زمانی، ۵۰ هزار کلاک داریم. از آنجا که counter را ۸ بیتی در نظر گرفتیم، هر ramp به ۲۵۶ قسمت مساوی تقسیم می‌شود که هر کدام را یک duty cycle می‌نامیم. همچنین در هر ramp ۵۰ هزار کلاک داریم، پس در هر duty cycle 50,000 / 256 ~= 195 کلاک داریم.

tick_counter شمارنده 195 تایی ما است که از 0 تا 195 شمرده و هر بار duty را افزایش می‌دهد. در خط زیر خروجی مشخص می‌شود:

```
pwm_out <= (counter < duty) ? 1 : 0;
```

در ابتدا تمام بیت‌های خروجی (بدیهتا تنها یک بیت خروجی داریم و منظور از بیت‌های خروجی، مقدار آن بیت پس از هر کلاک است) صفر اند، اما با گذشت زمان duty افزایش یافته و تعداد بیت های 1 در هر step بیشتر می‌شود.

```
module pwm_ramp (
    input clk,
    input reset,
    output reg pwm_out
);
    reg [7:0] counter = 0;       // 8-bit PWM counter
    reg [7:0] duty = 0;          // 8-bit Duty cycle (0?255)
    reg [15:0] tick_counter = 0; // Controls ramp speed

    // How fast to ramp duty: 1 ms ramp = 50,000 cycles @ 50MHz
    // 256 steps from 0 to 255 ? increase duty every 195 cycles (roughly)
    localparam STEP_PERIOD = 195;

    always @(posedge clk) begin
    if (reset) begin
        counter <= 0;
        duty <= 0;
        tick_counter <= 0;
        pwm_out <= 0;
    end else begin
        counter <= counter + 1;

        pwm_out <= (counter < duty) ? 1 : 0;

        tick_counter <= tick_counter + 1;

        // Increase duty every fixed interval
        if (tick_counter >= STEP_PERIOD) begin
            tick_counter <= 0;
            if (duty < 255)
                duty <= duty + 1;
            else
                duty <= 0;
        end
    end
end

endmodule
```

تست بنچی که استفاده کردیم به این صورت است:

```
`timescale 1ns/1ps

module pwm_tb;
  reg clk = 0;
  reg reset = 1;
  wire pwm_out;

  // Instantiate PWM module
  pwm_ramp uut (.clk(clk), .reset(reset), .pwm_out(pwm_out));

  // Clock generation: 50 MHz (20 ns period)
  always #1 clk = ~clk;

  integer f, i;

  initial begin
    f = $fopen("pwm_output.txt", "w");

    // Initial reset
    #2;
    reset = 0;

    // Let simulation run and log PWM output
    for (i = 0; i < 250000; i = i + 1) begin // 250,000 iterations / 50,000 = 5
ms
        @(posedge clk);
        $fwrite(f, "%d\n", pwm_out);
    end

    $fclose(f);
    $stop;
  end
endmodule
```

module نوشته شده را instantiate کرده و کلاک را طراحی می‌کنیم. سپس reset را یک کرده، کلاک زده و مجددا صفر میکنیم تا مدار آماده اجرا شود. سپس به مدت ۲۵۰ هزار کلاک، که معادل ۵ میلی ثانیه است، خروجی را در یک فایل متنی (pwm_output.txt) می‌نویسیم تا سپس به کمک کد پایتون، بتوانیم سیگنال آنالوگ آن را مشاهده کنیم.

اکنون به بررسی خروجی می‌پردازیم. در ابتدا ۲۵۷ بیت صفر نوشته شده است، سپس تعداد بیت های یک متوالی به این صورت اند:

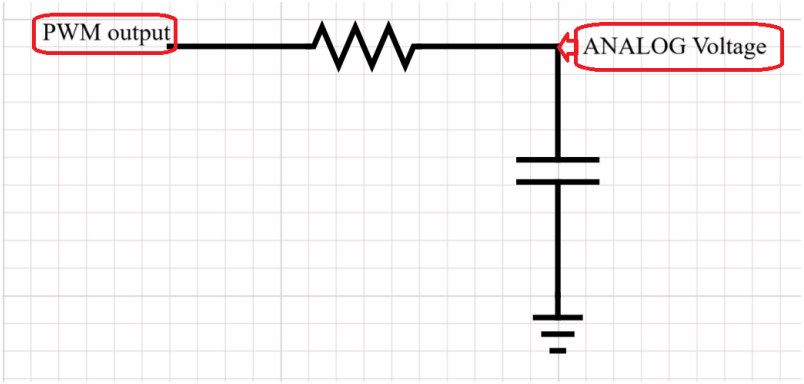
1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, … , … , 250, …

همانطور که مشاهده می‌کنید، در برخی مراحل تعداد بیت های 1 به جای اینکه یکی بیشتر شود، بیشتر افزایش یافته. علت این است که ما در پیاده سازی خود، ۲۵۶ step در نظر گرفتیم که هر کدام ۱۹۵ کلاک زمان می‌برد. از آنجا که این دو عدد برابر نیستند، روند افزایش تعداد بیت های 1 متوالی دقیقا ثابت نیست. اما این موضوع باعث ایجاد تفاوت قابل توجهی در شکل ramp تولید شده نمی‌شود و بدون بررسی دقیق بیت‌های تولید شده قابل تشخیص نیست. اگر بخواهیم خروجی دیجیتال دقیق تری داشته باشیم، کافی است ۲۵۶ مرحله در هر ramp در نظر بگیریم که هر یک ۲۵۶ کلاک طول می‌کشد، در این صورت تعداد بیت های 1 در مرحله ی i ام برابر است با i و پس از آن $256 - i$ بیت صفر داریم. البته برای این کار لازم است فرکانس کاری را بر روی 65.536 M Hz تنظیم کنیم. $256 * 256 * K = 65,536\ K\ Hz$

از آنجا که به تجهیزات آزمایشگاهی و اسیلوسکوپ دسترسی نداریم، از کد زیر برای مشاهده سیگنال آنالوگ استفاده می‌کنیم:

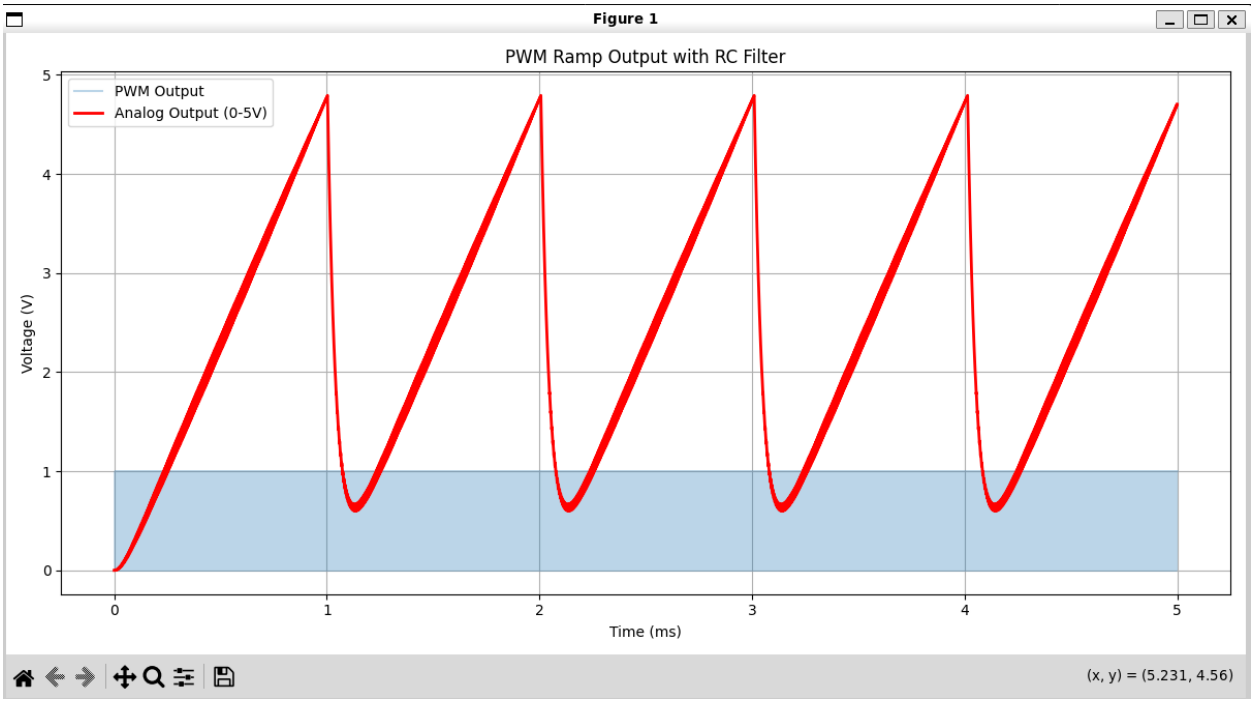
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter

# --- Load Verilog PWM Output ---
with open("pwm_output.txt") as f:
    pwm_signal = np.array([
        int(line.strip()) for line in f
        if line.strip() in ('0', '1')
    ])

# --- Time settings ---
sample_rate = 50_000_000  # 50 MHz
dt = 1 / sample_rate
t = np.arange(len(pwm_signal)) * dt

# --- Optimized RC Filter ---
R = 400 # 400 Ohm
C = 0.0000001 # 0.1 micro Farad = 100 nano Farad
rc_time_constant = R * C # 0.00004 s = 40 micro second

alpha = dt / (rc_time_constant + dt)
analog_output = lfilter([alpha], [1, alpha - 1], pwm_signal)

# Scale to 0-5V range (assuming V0=0V, V5=5V)
voltage_output = analog_output * 5.0

# --- Plot ---
plt.figure(figsize=(12, 6))

# Plot first few ms to see the ramp clearly
plot_samples = int(5e-3 / dt)  # Show first 5ms

plt.plot(t[:plot_samples] * 1000, pwm_signal[:plot_samples],
         label="PWM Output", alpha=0.3)
plt.plot(t[:plot_samples] * 1000, voltage_output[:plot_samples],
         label="Analog Output (0-5V)", linewidth=2, color='red')
plt.xlabel("Time (ms)")
plt.ylabel("Voltage (V)")
plt.title("PWM Ramp Output with RC Filter")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

مداری که ایجاد می‌شود به این صورت است:

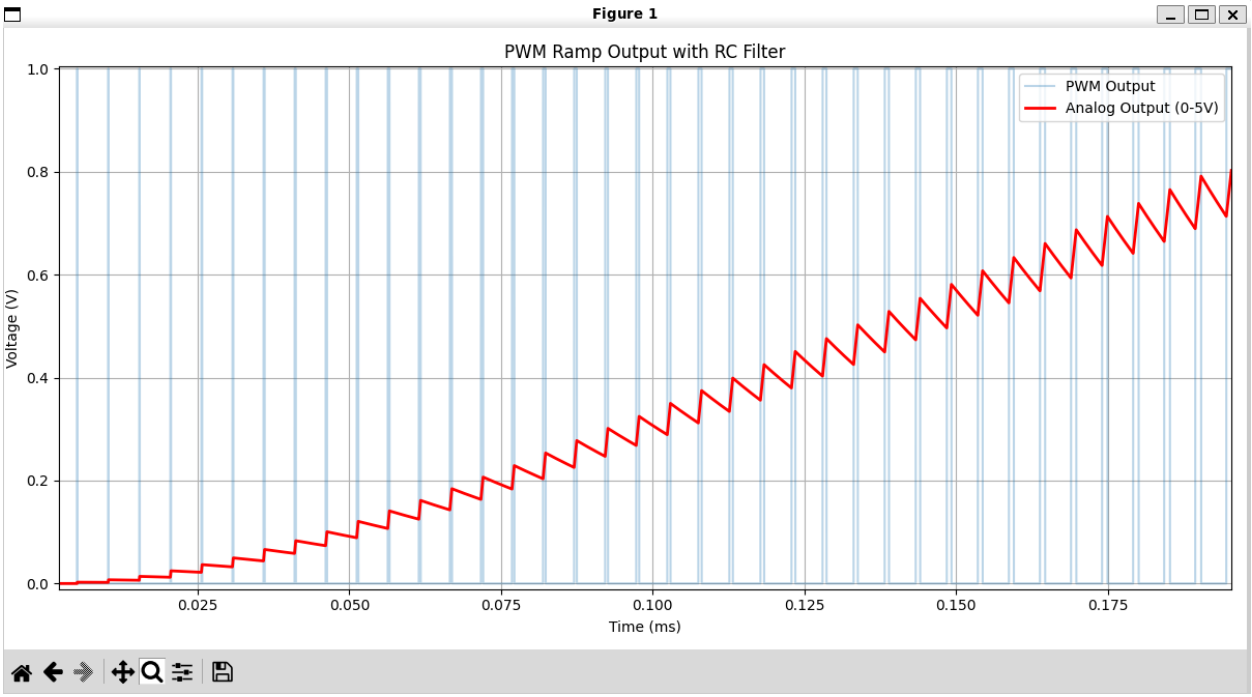


خروجی FPGA را به محل مشخص شده متصل می‌کنیم و در سمت دیگر به جای صفر و یک، سیگنال آنالوگ داریم چرا که مدار RC از تغییرات ناگهانی ولتاژ جلوگیری می‌کند. فرض می‌کنیم از یک LED برای مشاهده سیگنال آنالوگ استفاده می‌کنیم.

با اجرای کد پایتون، چنین سیگنالی مشاهده می‌کنیم:

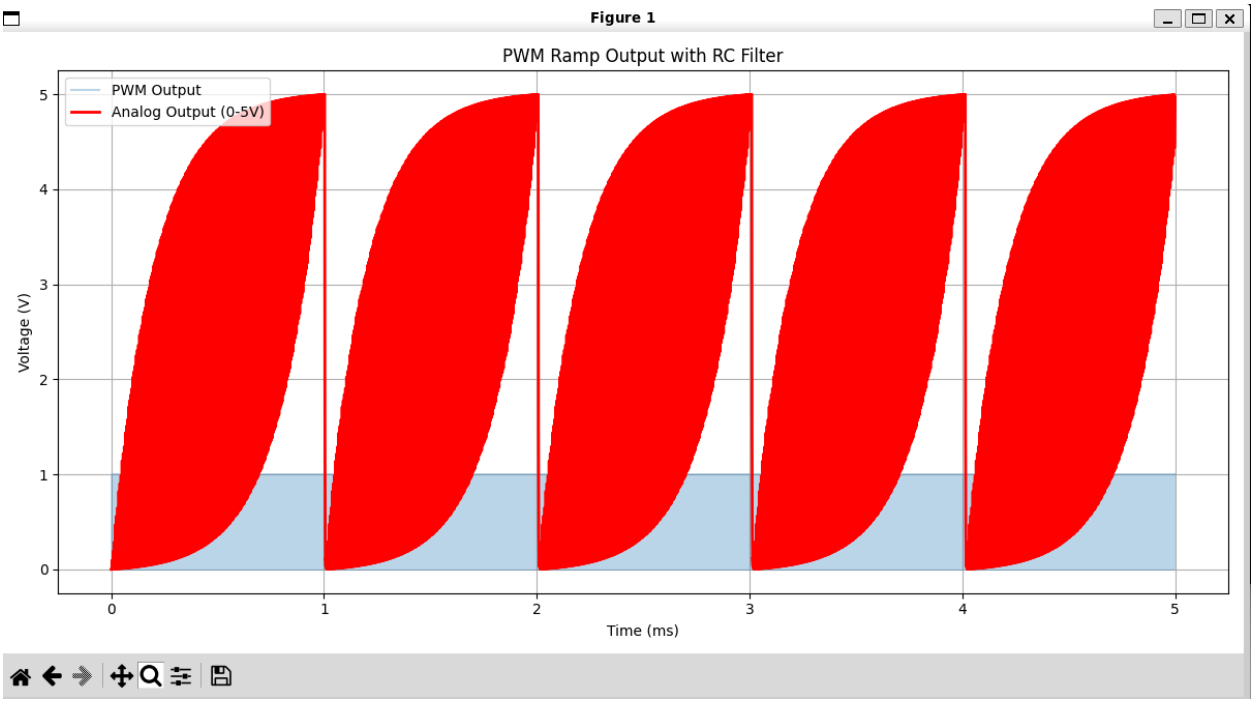


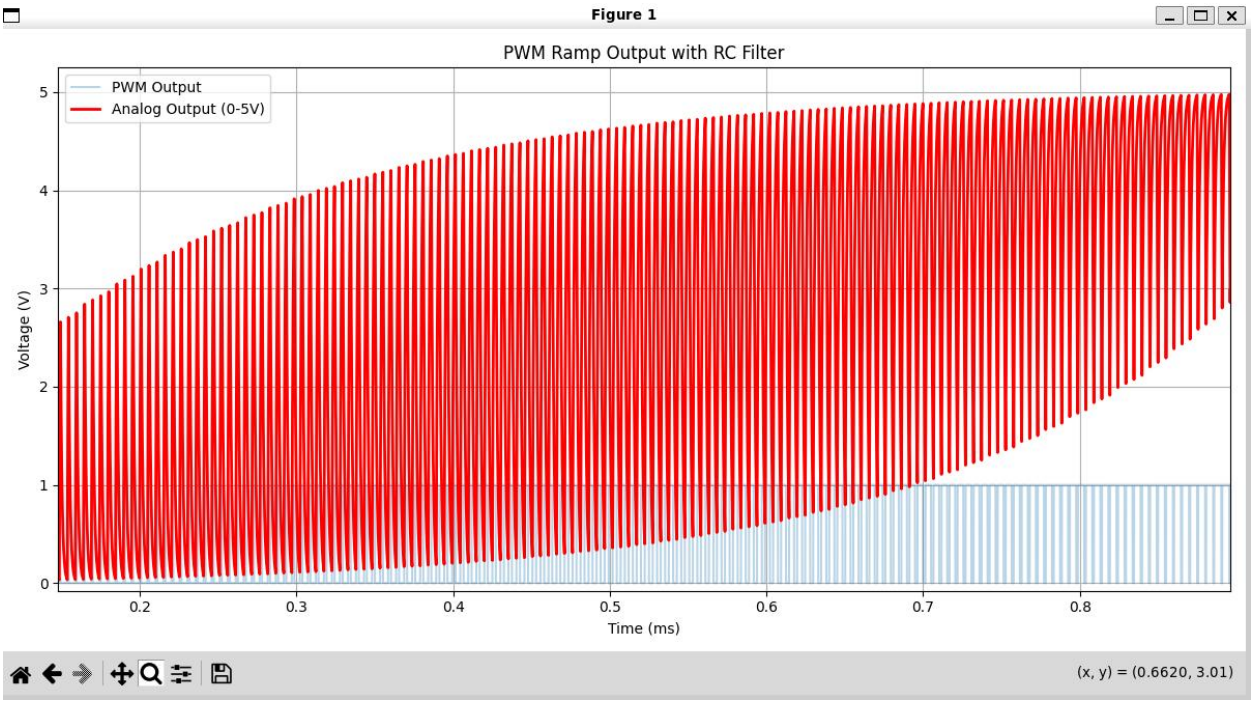
ناحیه آبی رنگ مقدار دیجیتال خروجی pwm است که از آنجا که فرکانس بالایی دارد، به نظر می‌رسد که تمام ناحیه بین صفر تا یک را پر کرده است، اما با زوم کردن بر روی ناحیه دلخواه، می توان مشاهده کرد کرد که مقدار بیت خروجی در هر لحظه یکی از مقادیر صفر یا یک است:



خط قرمز سیگنال آنالوگ خروجی است. مشخصات مدار RC و به طور مشخص ثابت زمانی مدار در شکل این سیگنال بسیار تاثیر گذار است. در دو تصویر بالا دو مورد مشاهده می‌شود. یکی اینکه سیگنال آنالوگ دقیقا به ۵ ولت نمیرسد و پس از آن هم دقیقا به صفر ولت باز نمی‌گردد. مورد دیگر اینکه با زوم کردن بر روی خط قرمز، متوجه نوسان آن میشویم. این مشخصات با تغییر مقدار ثابت زمانی مدار RC قابل کنترل اند. هرچه ثابت زمانی را کمتر کنیم، خازن در زمان کوتاه‌تری شارژ یا دشارژ شده و به ۰ و ۵ ولت نزدیک‌تر می‌شود. پس از اتمام ramp خازن سریع‌تر میتواند دشارژ شود و در زمان کوتاه تری به مقدار نزدیک‌تری به صفر برسد. اما در همین حال، مدار نسبت به ورودی که از FPGA می‌گیرد حساس تر شده و نوسانات خط قرمز رنگ بیشتر می‌شود و smoothness سیگنال آنالوگ کاهش می‌یابد. از طرف دیگر با افزایش ثابت زمانی مدار، حساسیت مدار کاهش یافته و خط قرمز رنگ صاف تر شده اما پس از اتمام ramp زمان بیشتری برای reset شدن نیاز دارد و شکل ramp از حالت ایده‌آل خود دور می‌شود.

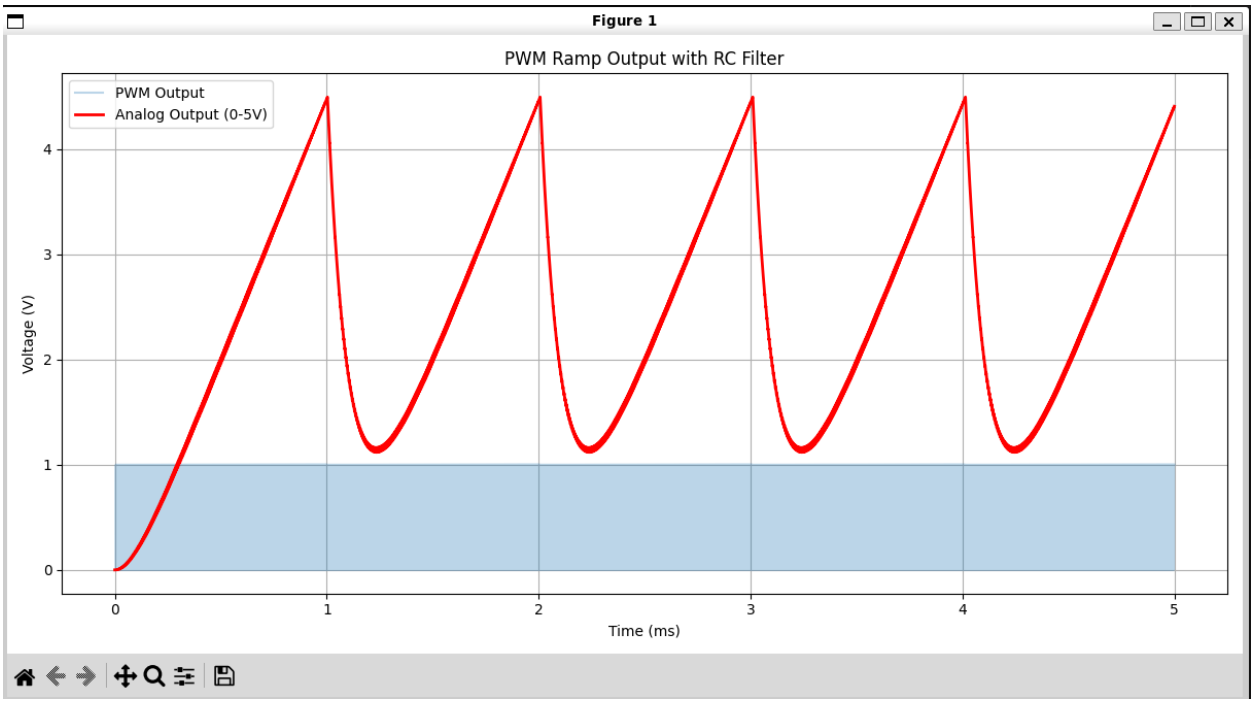
در این تصاویر ثابت زمانی ۱ میکرو ثانیه است:





همانطور که مشاهده می‌کنید سیگنال آنالوگ به مقدار دقیق ۵ رسیده و بلافاصله در زمان بسیار کوتاهی به صفر می‌رسد. اما نوسان آن نیز بسیار زیاد است.

در تصویر زیر ثابت زمانی ۱۰۰ میکرو ثانیه است:



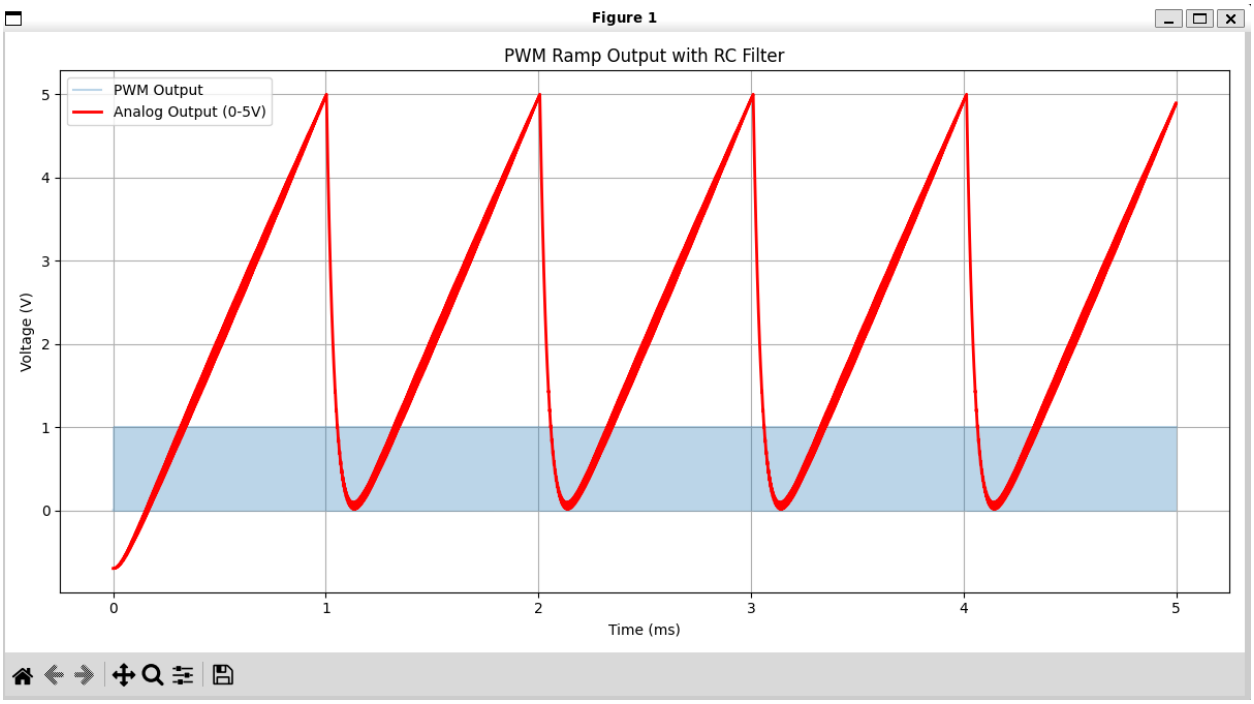
بر خلاف حالت قبلی، در اینجا سیگنال فاصله زیادی تا رسیدن به مقادیر ۵ و ۰ ولت دارد و همچنین دشارژ شدن خازن و کاهش ولتاژ پس از هر ramp زمان بسیار زیادی می‌برد و حالت طبیعی ramp را از دست می‌دهد. البته که خط قرمز بسیار باریک تر شده که نشان دهنده نوسان کم سیگنال دارد.

نهایتا ما در اینجا ثابت زمانی ۴۰ میکرو ثانیه را برای مدار خود انتخاب کردیم (مقاومت ۴۰۰ اهم و خازن با ظرفیت ۱۰۰ نانو فاراد). تنظیم ثابت زمانی مدار RC یک trade-off بین حفظ شکل کلی ramp و یا smoothness سیگنال آنالوگ است که می‌توان با توجه به انتظارات ما از سیگنال آنالوگ خروجی، آن را در محیط آزمایشگاه/کد پایتون تنظیم کرد و تفاوتی در کد Verilog ایجاد نمی‌کند.

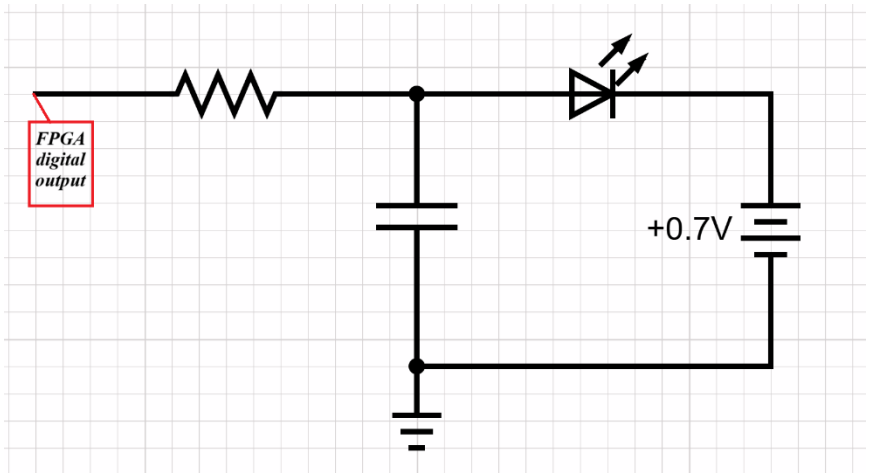
در ادامه برای اینکه سیگنال آنالوگ ما بین ۰ تا ۵ ولت باشد، این خط کد را جایگزین کد قبلی کرده:

```
# Scale to 0-5V range (assuming V0=0V, V5=5V)
voltage_output = analog_output * 5.95 - 0.7
```

و طبق خواسته سوال در خروجی داریم:



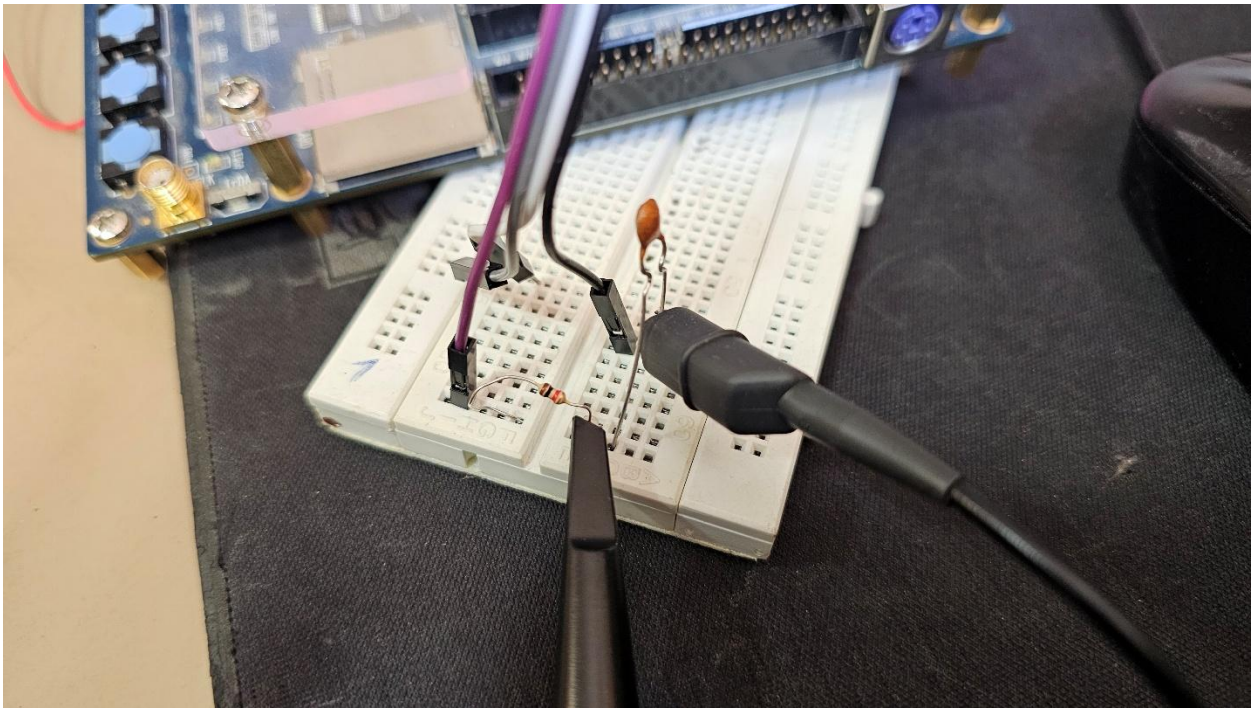
معادل این کار در محیط آزمایشگاه بدین صورت است که ورودی ولتاژ هایی که به FPGA می‌دهیم ۰ ولت و ۵.۹۵ ولت باشد. همچنین سمت دیگر LED را به ولتاژ ثابت ۰.۷ ولت متصل کنیم. در این صورت تمام بازه ۰ تا ۵ ولت در خروجی آنالوگ ما (در اینجا LED) مشاهده می‌شود. توجه داریم که ولتاژ منفی ایجاد شده در تصویر فقط در شروع کار مدار رخ داده و دیگر تکرار نمی‌شود، از طرفی مشکلی هم در روند کار ایجاد نمی‌کند، چراکه در این بازه زمانی LED خاموش می‌ماند و با کمی تاخیر روشن می‌شود.

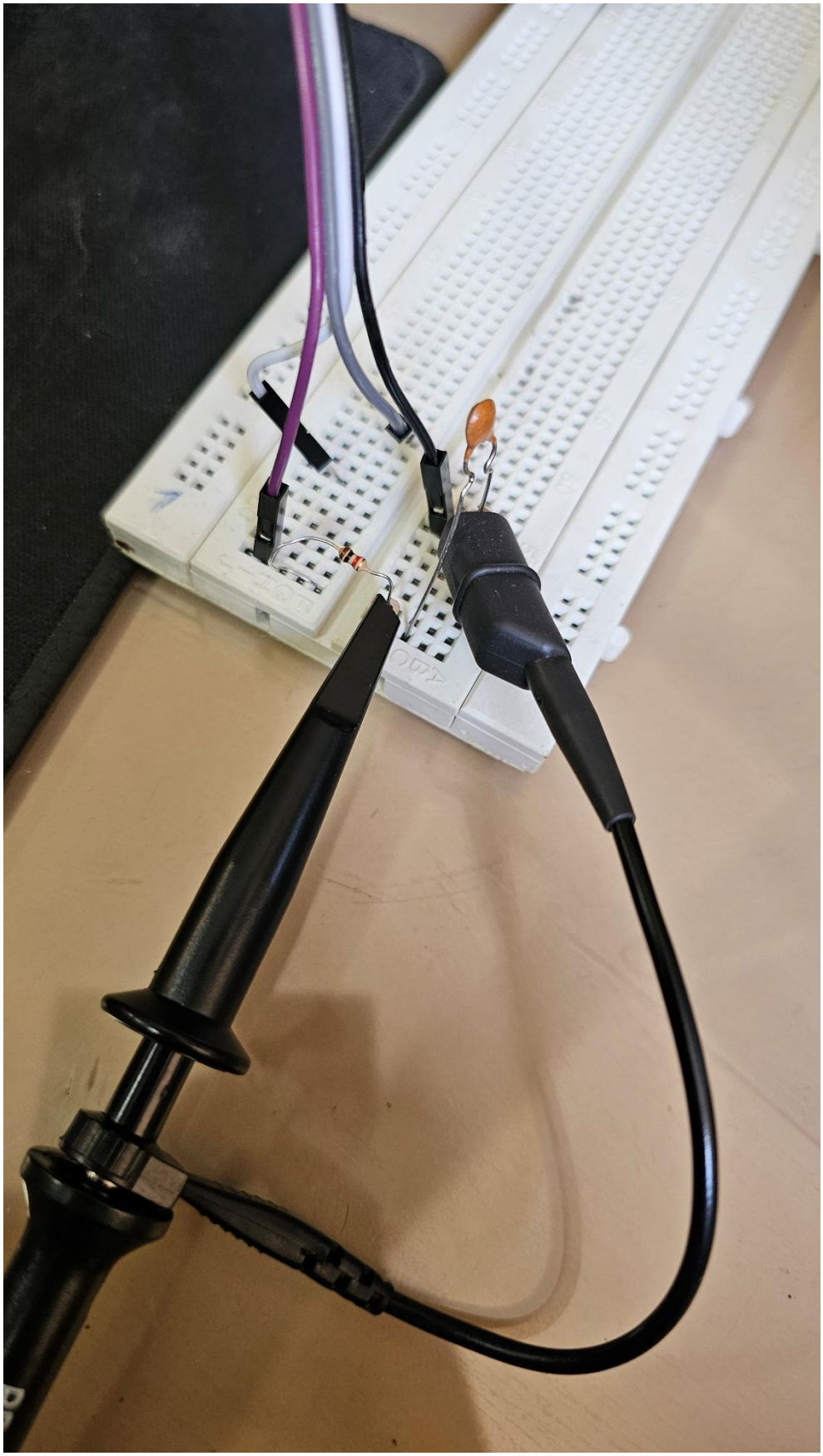


آزمایشگاه

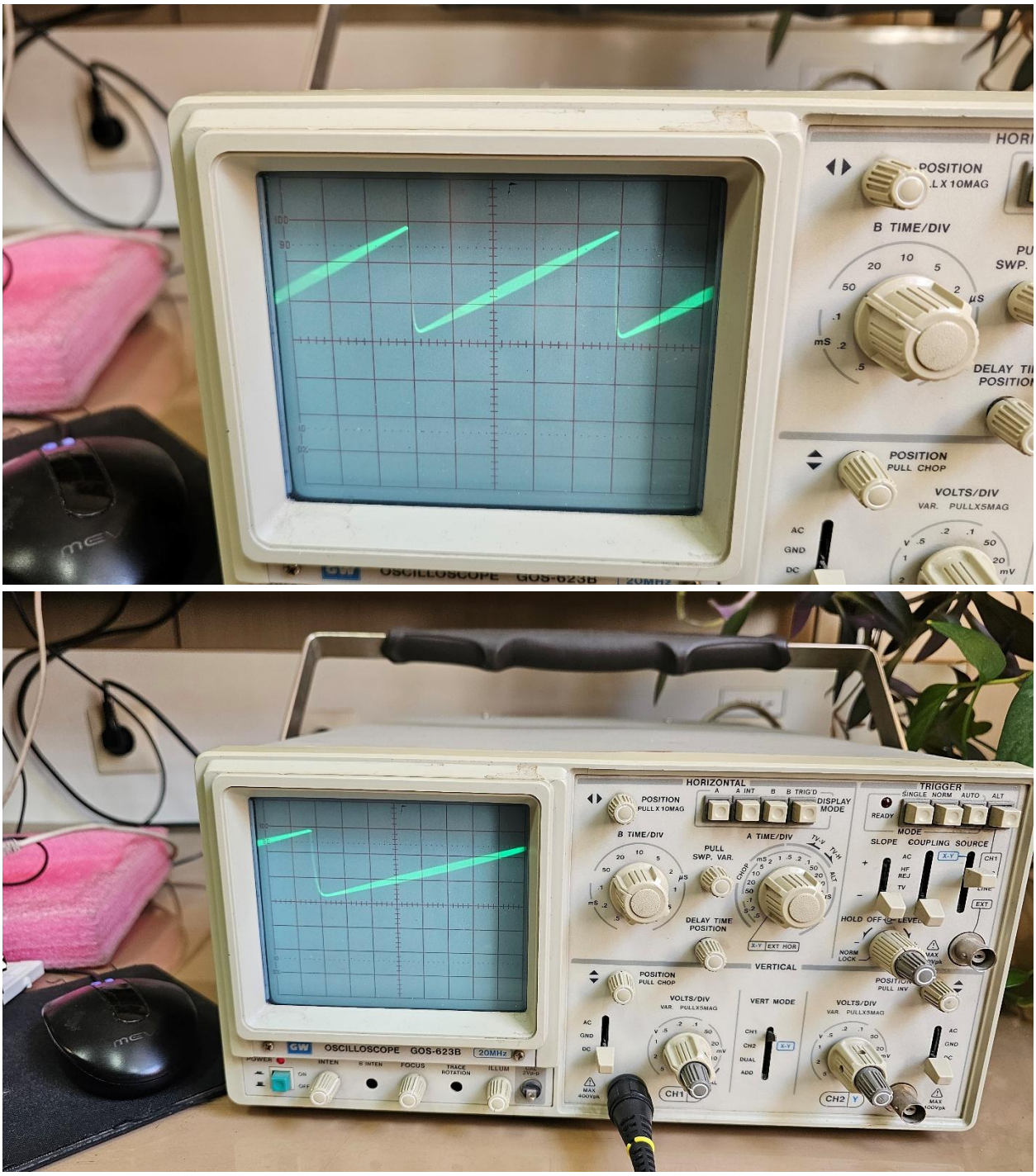
در ادامه برای اجرای کد به آزمایشگاه رفتیم. کد Verilog را با برنامه Quartus کامپایل کردیم، به کمک دفترچه راهنمای بورد FPGA موجود در آزمایشگاه، ۳ پین برای ۳ پارامتر برنامه (clk, reset, pwm_out) تعریف کردیم. reset را یک سوئیچ بر روی بورد تعریف کردیم و مقدار آن را صفر نگه داشتیم. clock را به یک پین دیگر مرتبط کردیم که به طور خودکار کلاک ۵۰ مگا هرتز پیش فرض بورد بود. نهایتا pwm_out را به یک پین خروجی بر روی بورد مرتبط کرده و با اتصال سیم به آن، به خروجی module دسترسی پیدا کردیم. با اجرای کد کامپایل شده در برنامه Quartus ، FPGA شروع به کار و تولید سیگنال کرد.

بر روی یک برد بورد، یک خازن ۱۰ نانو فاراد و یک مقاومت ۱ کیلو اهم قرار دادیم و این دو را به شکل سری (متوالی) به هم متصل کردیم. به سر دیگر خازن، ground بورد FPGA و سر ground اسیلوسکوپ را متصل کردیم. به سر دیگر مقاومت، خروجی که از FPGA گرفتیم را متصل کردیم. سر دیگر اسیلوسکوپ را نیز به محل اتصال خازن و مقاومت متصل کردیم. (سیم مشکی رنگ ground و سیم بنفش خروجی سیگنال دیجیتال است)

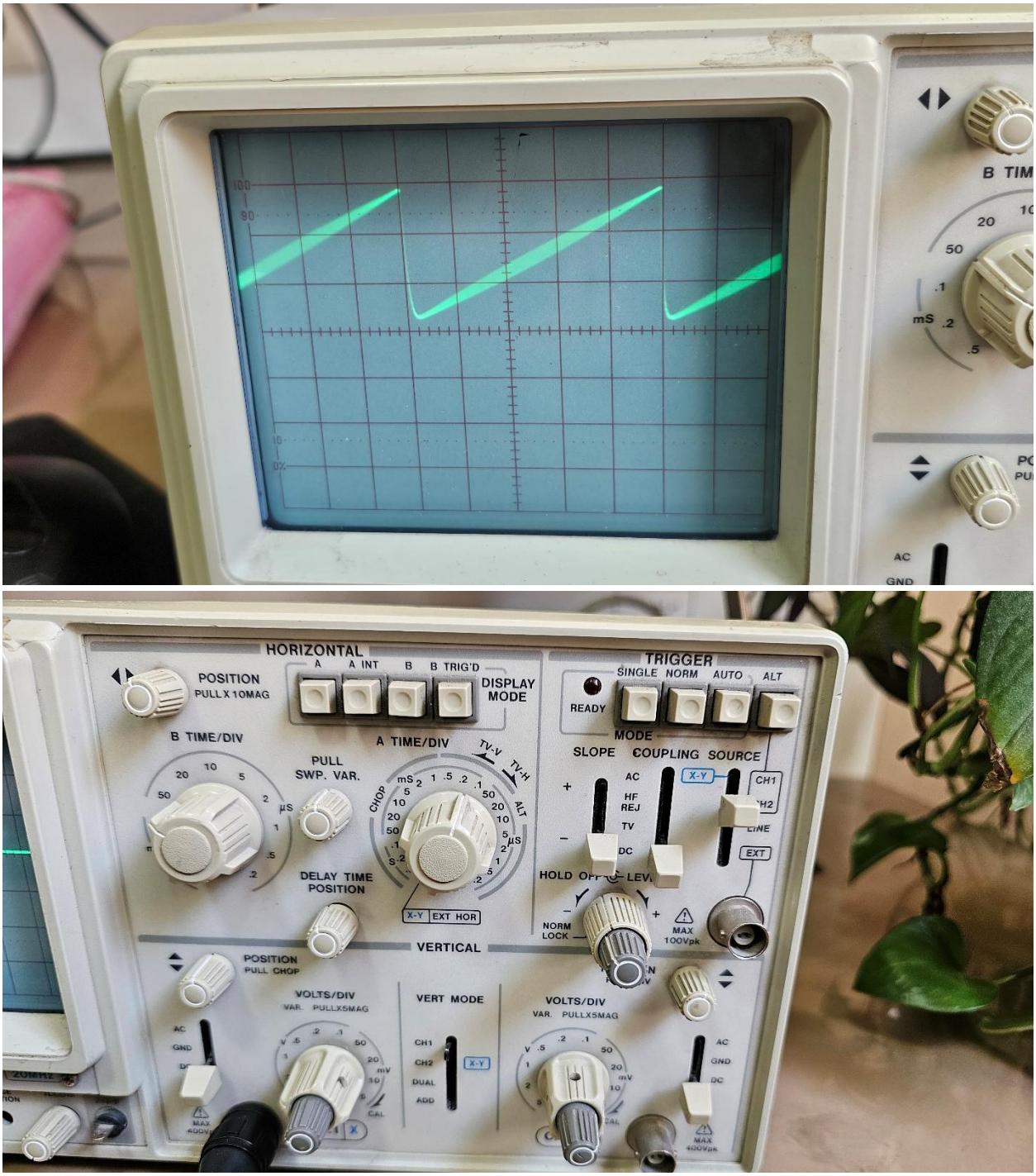




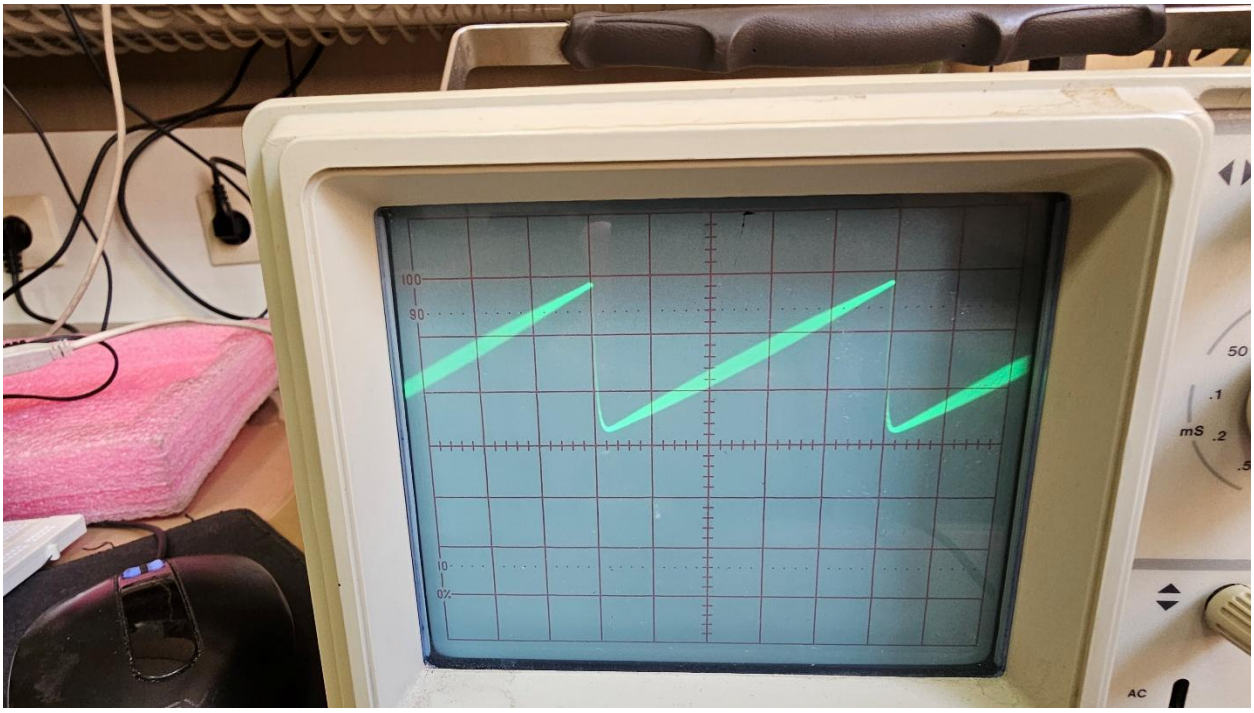
در ادامه تصاویر سیگنال آنالوگ مشاهده شده بر روی اسیلوسکوپ را مشاهده می‌کنید:

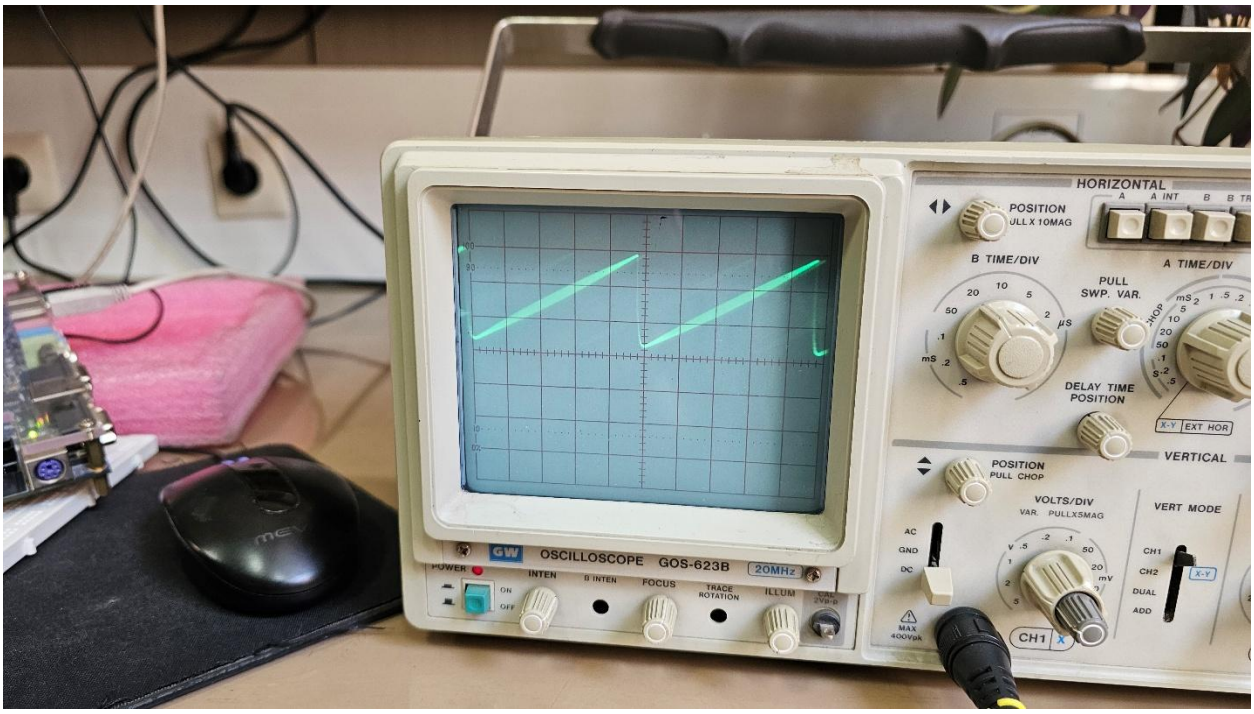


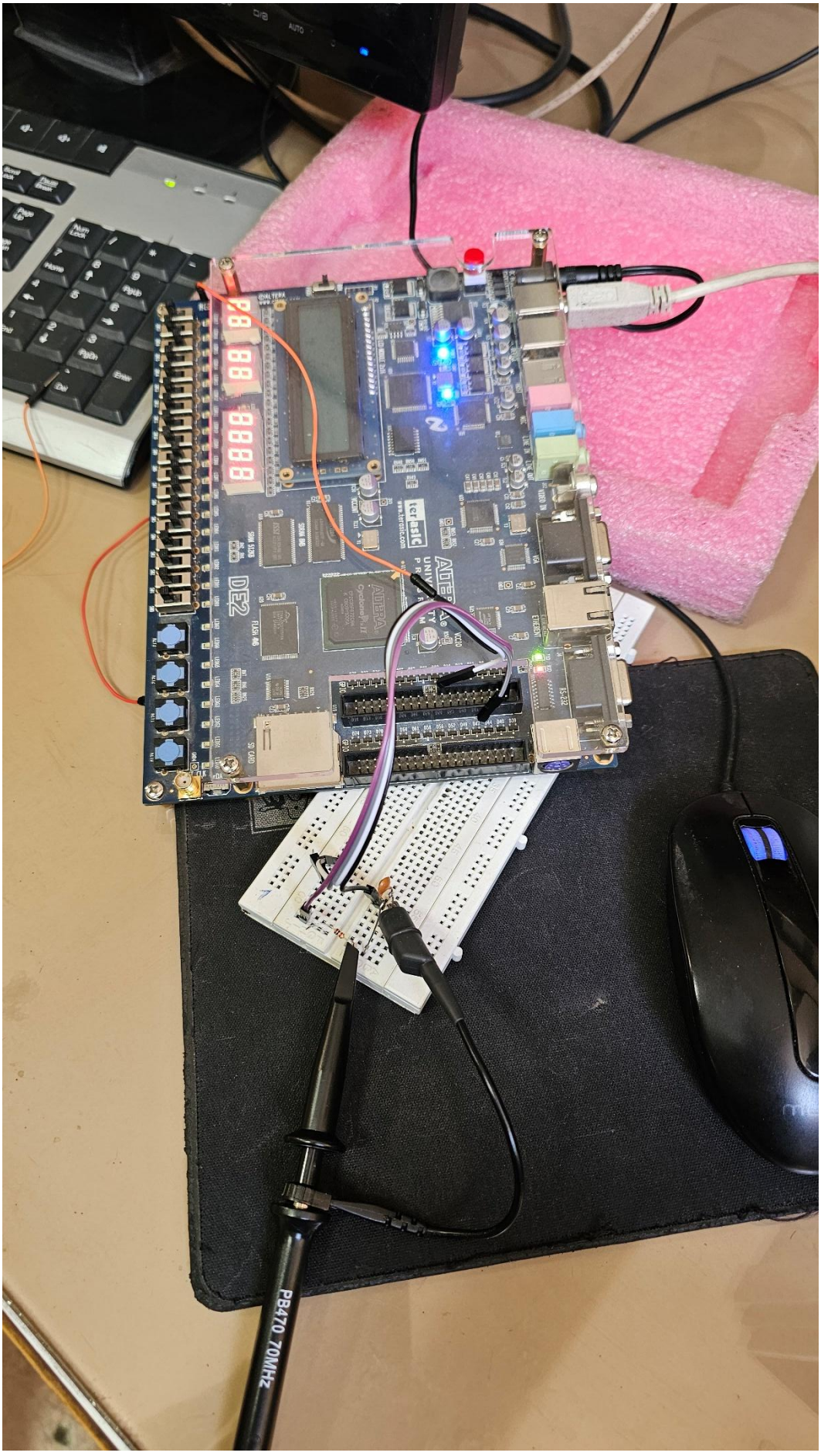
در تصویر بالا می‌بینیم که شیب بالا رفتن ولتاژ یکنواخت و مناسب است، همچنین drop از ولتاژ بالا به ولتاژ پایین در زمان کوتاه و قابل قبولی انجام می‌شود.



تنظیمات اسیلوسکوپ به هنگام مشاهده سیگنال آنالوگ در تصویر بالا مشخص است.




HORIZONTAL
POSITION
B TIME/DIV
A TIME/DIV
PULL SWP. VAR.
DELAY TIME POSITION
X-Y
EXT HOR
VERTICAL
POSITION
PULL CHOP
VOLTS/DIV
VAR. PULLX5MAG
VERT MODE
AC
GND
DC
CH1
CH2
DUAL
ADD
X-Y
CH1
X
GW
OSCILLOSCOPE GOS-623B
20MHz
POWER
ON
OFF
INTEN
B INTEN
FOCUS
TRACE ROTATION
ILLUM


DE2
terasic
PB470_70MHz