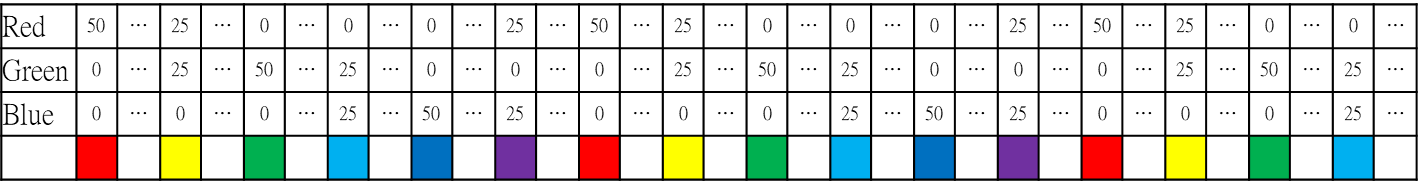
Digital Laboratory Lab08 Report

(Experiment Record Template)

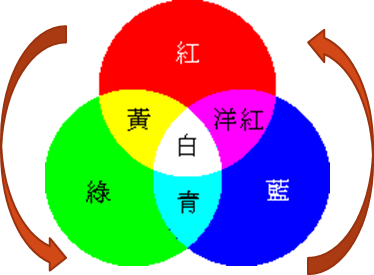
**FPGA Experiment-801: Tri-color LED Light Show**

**Create a 6-color gradient transition which is satisfied the following requirements:**

1. Modeling style: behavior
2. Input: FPGA built-in CMOS oscillator (i.e., pin E3)
3. Output: one built-in tri-color LED
4. Brightness and color should be controlled by PWM signals. A duty cycle of 0 to 50% is recommended to avoid generating uncomfortable levels of brightness. Over 50% duty cycle may cause discomfort.
5. Light show pattern (Duty cycle change list)

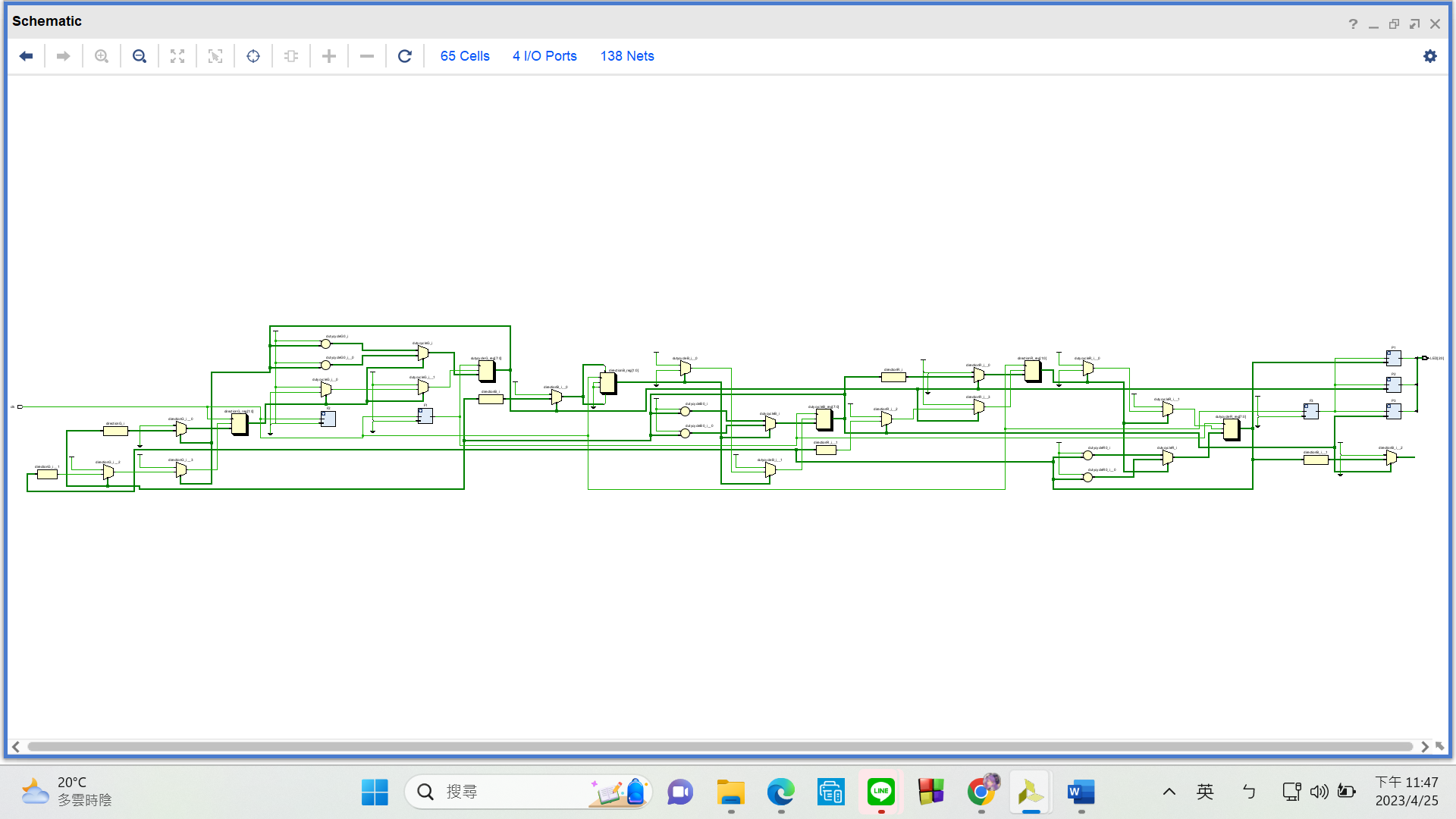


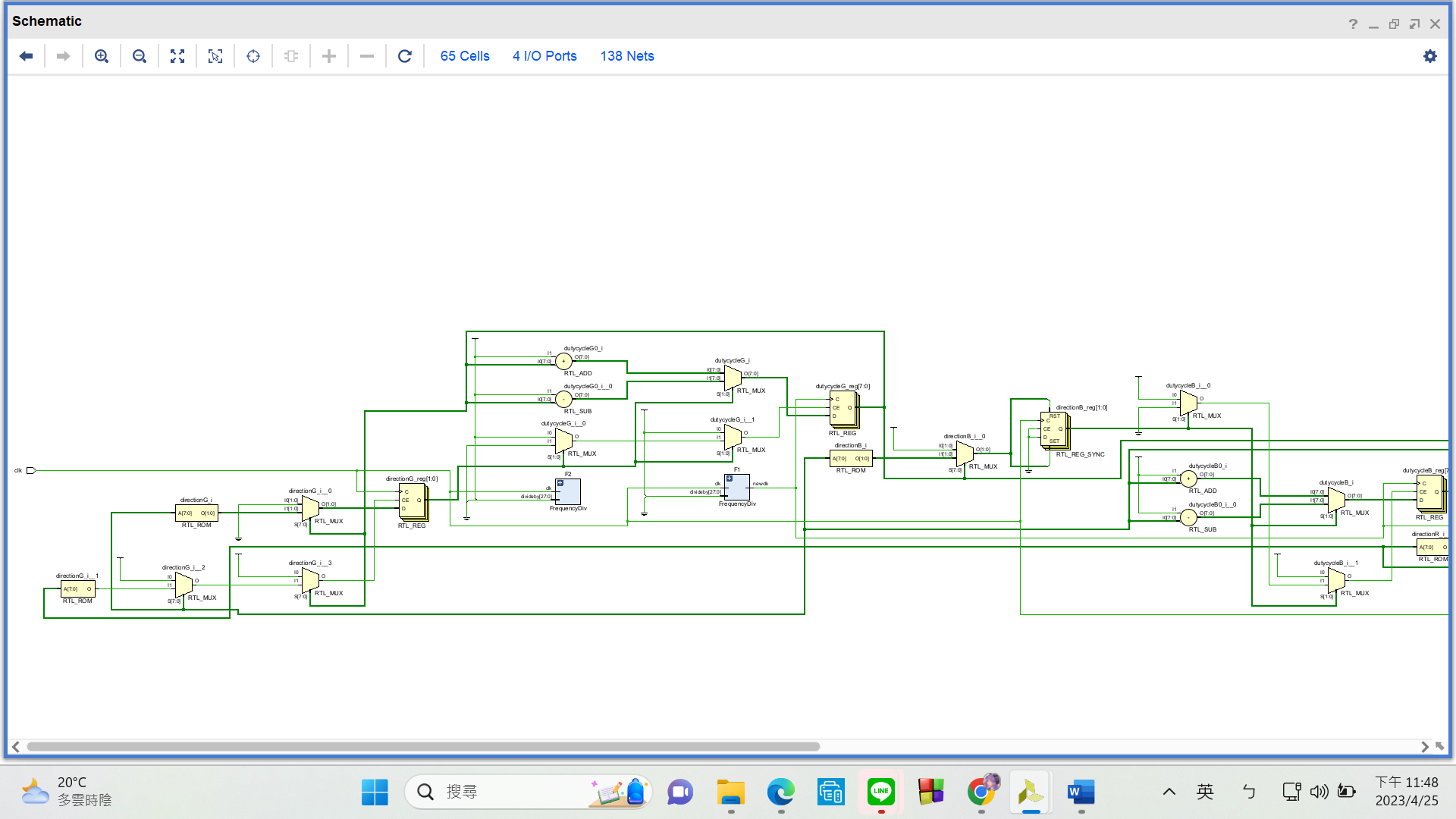
Note: The relationship between six colors.

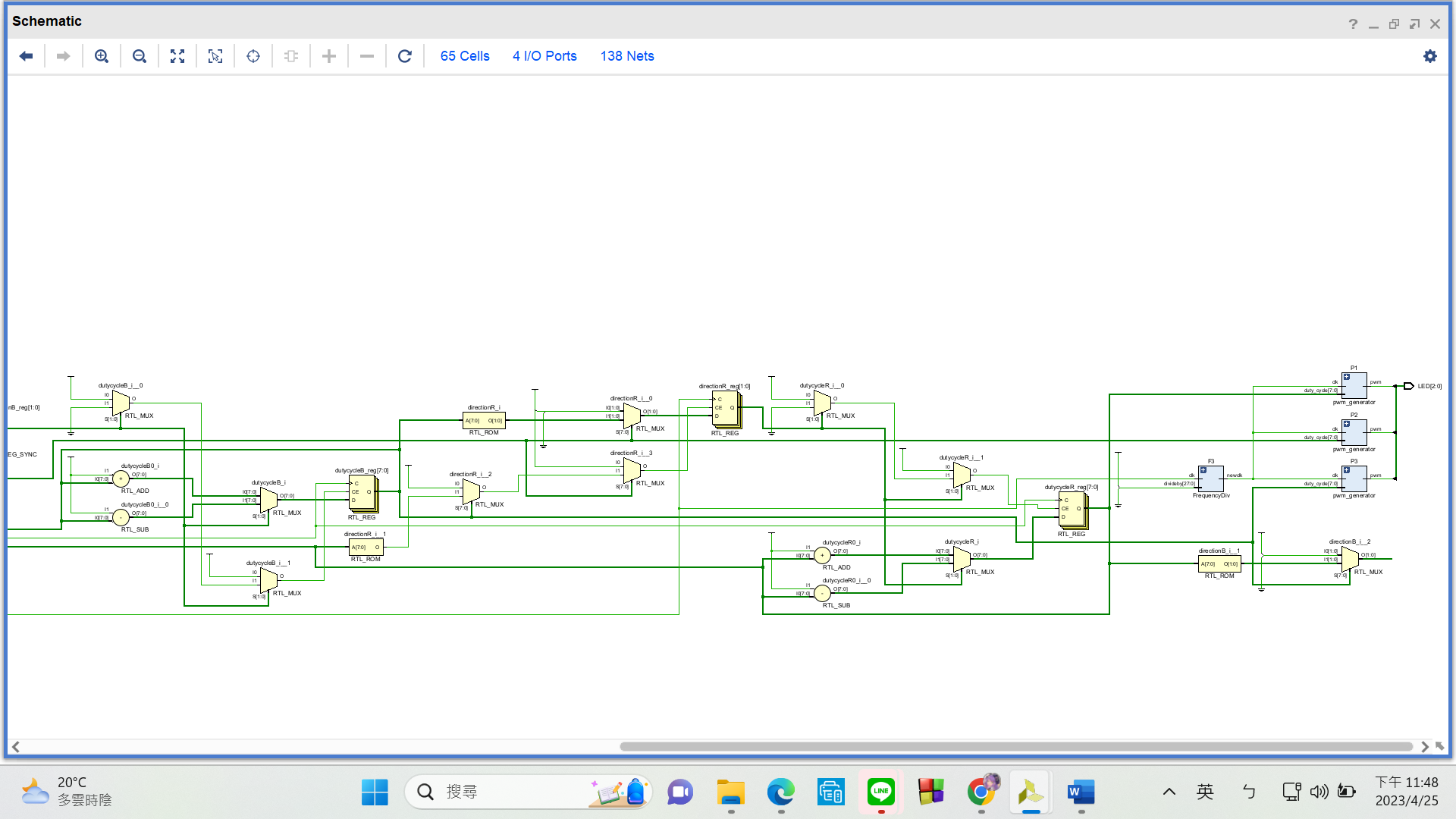


sample-801 <https://youtu.be/L5Jh26ULEEs>

Experiment Data

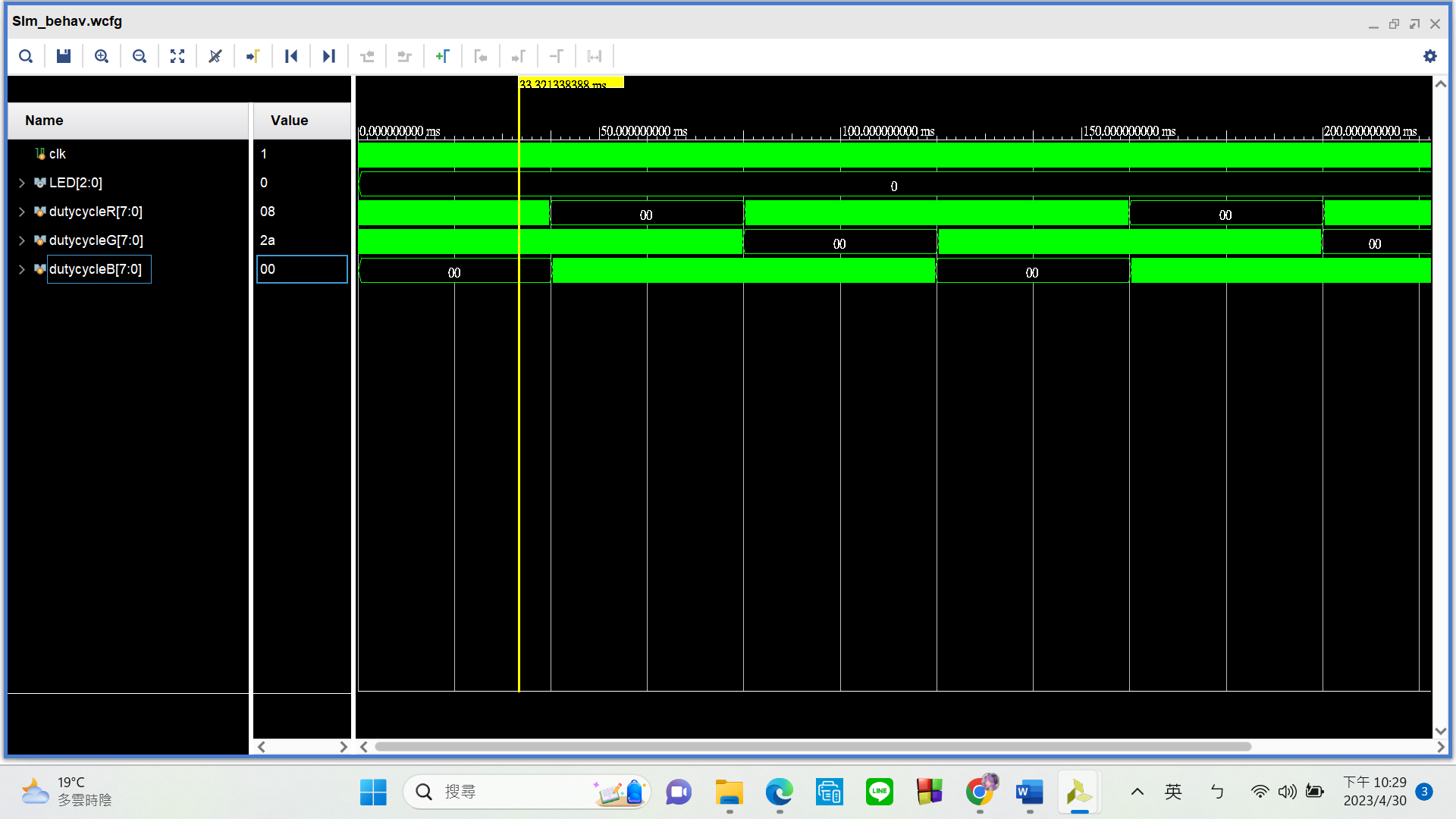
RTL Schematic





Simulation results

(Design your own testbench)



Notice that the duty cycles of the three **different pwm signals are 0 at separate intervals**. In practice, this should **give us the correct color changing** effect.

Practical results on the FPGA board

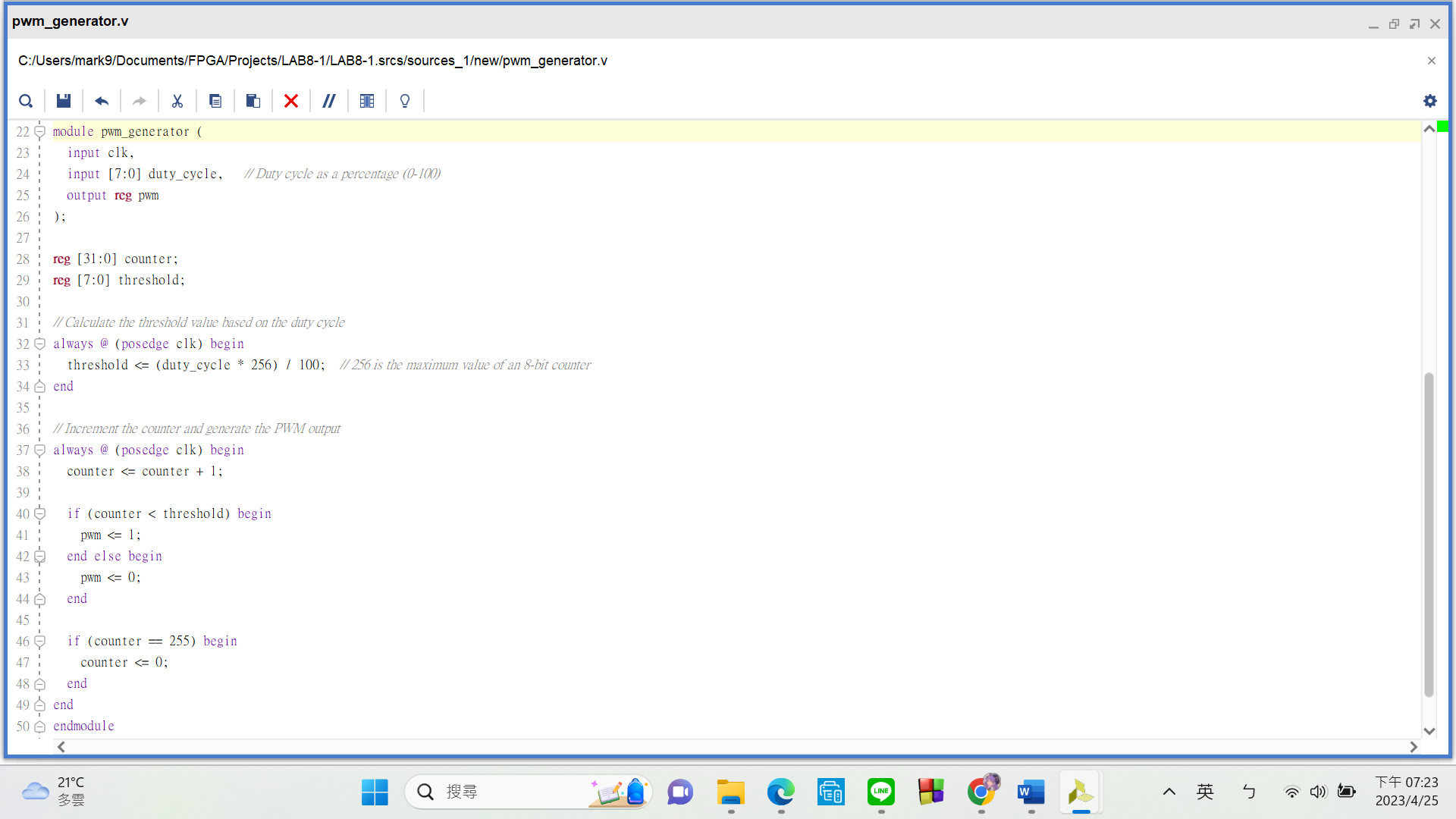
static record (photo, screen capture) or dynamic record (video link)

<https://youtu.be/4SJtnICU4k8>

OPTIONAL: design process

**Let’s look at my code in three sections.**

1. **PWM module**

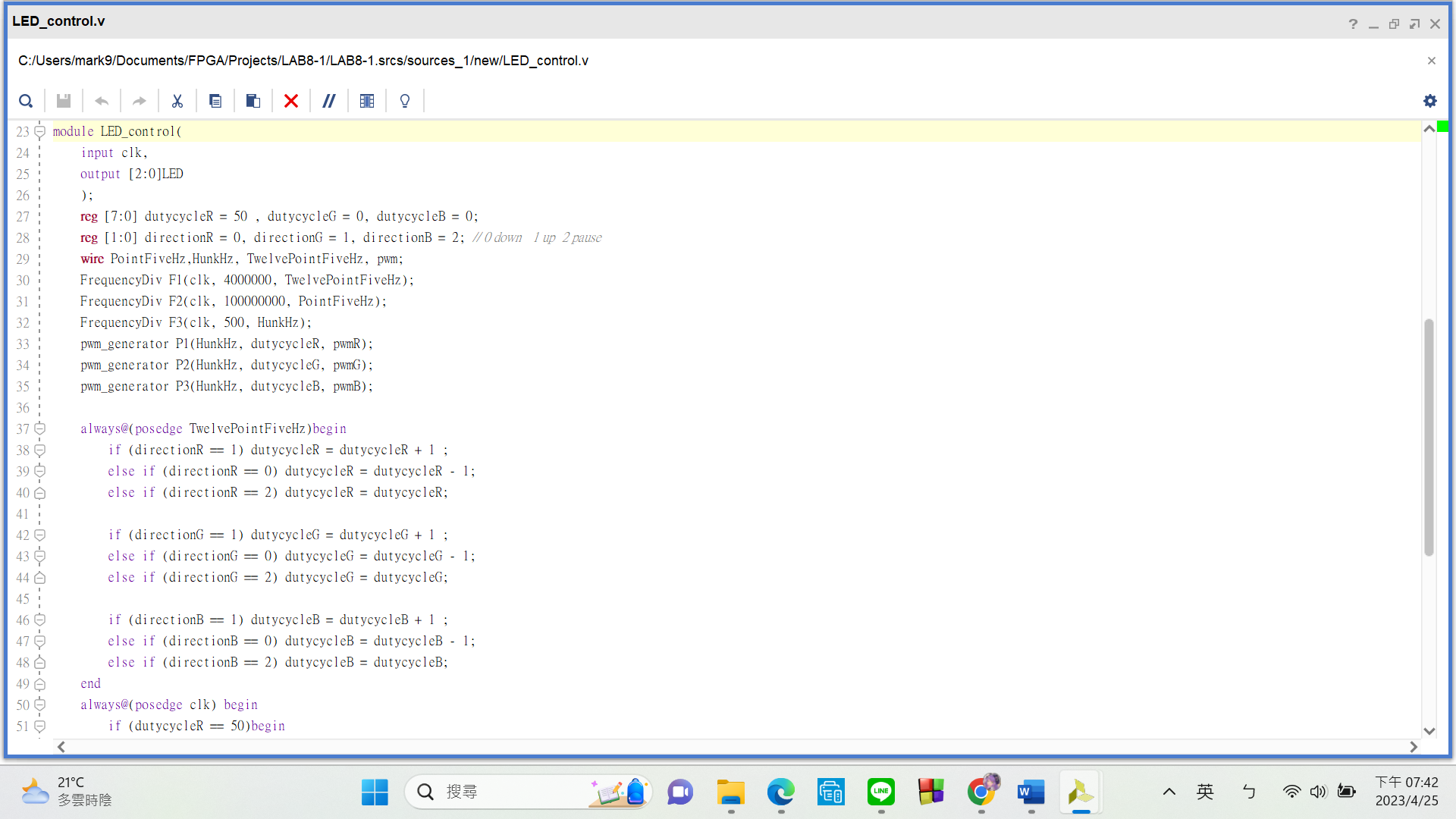


**PWM** stands for **pulse width modulation**. Simply put, **this module changes the duty cycle of the clock to whatever percentage we require.** My “pwm\_generator” has 2 inputs (clk: the clk whose duty cycle you want to change, duty\_cycle: the duty cycle you wish to change to)and1 output (pwm: the new clk signal with modified duty cycle).

In the first always block, the 8-bit-reg **“threshold” is set to 256 \* “duty\_cycle” / 100. 256 is the maximum value of an 8-bit-reg. The “duty\_cycle” / 100 is to convert our input into percentage.** For example, **I’ll give “duty\_cycle” a value of 25 if I want a duty cycle of 25% for my new modified pwm clk**. **The reason this is done in an always block** is that “duty\_cycle” may not stay constant. In fact, in this problem, we have to **let our duty cycle count down from 50 to 0 and then back to 50** to create the gradient change in LED color. We will get into that part in detail later on.

In the second always block, I let reg “counter” start counting from 0 and rest back to 0 once it reaches 255. When its value is lower than “threshold”, it will send out 1. When its value is above the threshold, it will send out 0. This allows us to change the duty cycle of the clk.

1. **LED\_control (Top Module) Part 1**

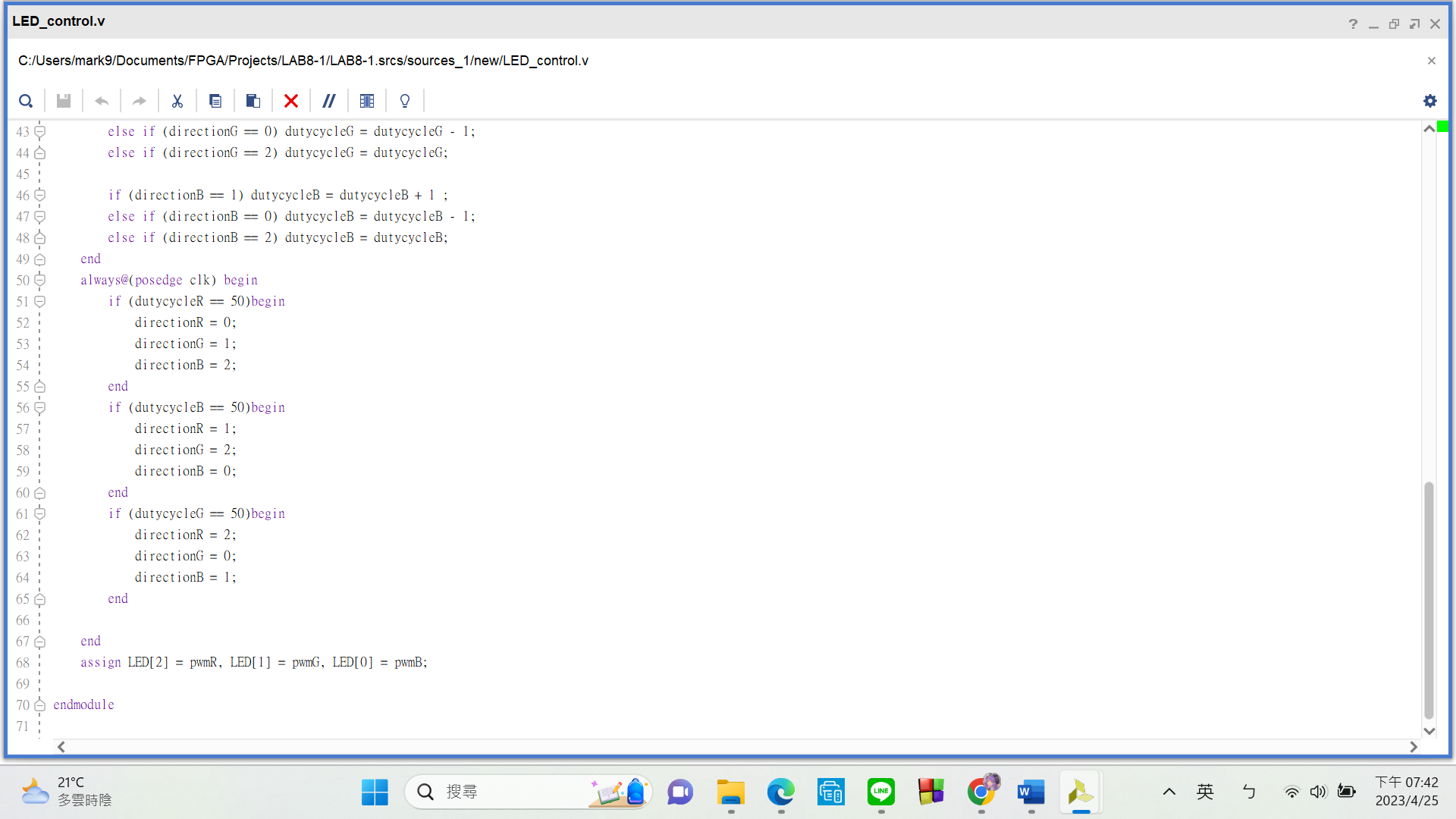


My method here **requires each color to have its own pwm signal,** that’s why I have 3 8-bit-regs(dutycycleR, dutycycleG, dutycycleB) and three wires(pwmR, pwmG, pwmB). There are **three control options** for changing the duty cycles of the pwm signals. In these three regs (“directionR”, “directionG”, “directionB”) **0 is for counting down, 1 is for counting up, and 2 is for pause**. The first always block is used to define these options.

1. **LED\_control (Top Module) Part 2**

**How do I know when to change between counting up, down, or pause?** Please refer to the **following picture** for further clarification. As you can see, whenever the duty cycle counters (dutycycleR, dutycycleG, dutycycleB) have to change direction, **there’s always one of them who has a value of 50.** In the always block below, I **used this characteristic to control when and how the counting should change.** Lastly, I assigned “pwmR”, “pwmG”, and “pwmB” each to a bit in a 3-bit-reg named “LED”, which is linked to the FPGA’s tricolor LED ports.





**FPGA Experiment-802: Traffic Light**

**Create a traffic light with count-down timer which is satisfied the following requirements:**

1. Modeling style: behavior
2. Input: FPGA built-in CMOS oscillator (i.e., pin E3)
3. Output: one built-in tri-color LED, 2-digit 7-seg display
4. Traffic light changing rules are as follows:

i. The LED color starts from red, and the timer counts down from 10 to 1.

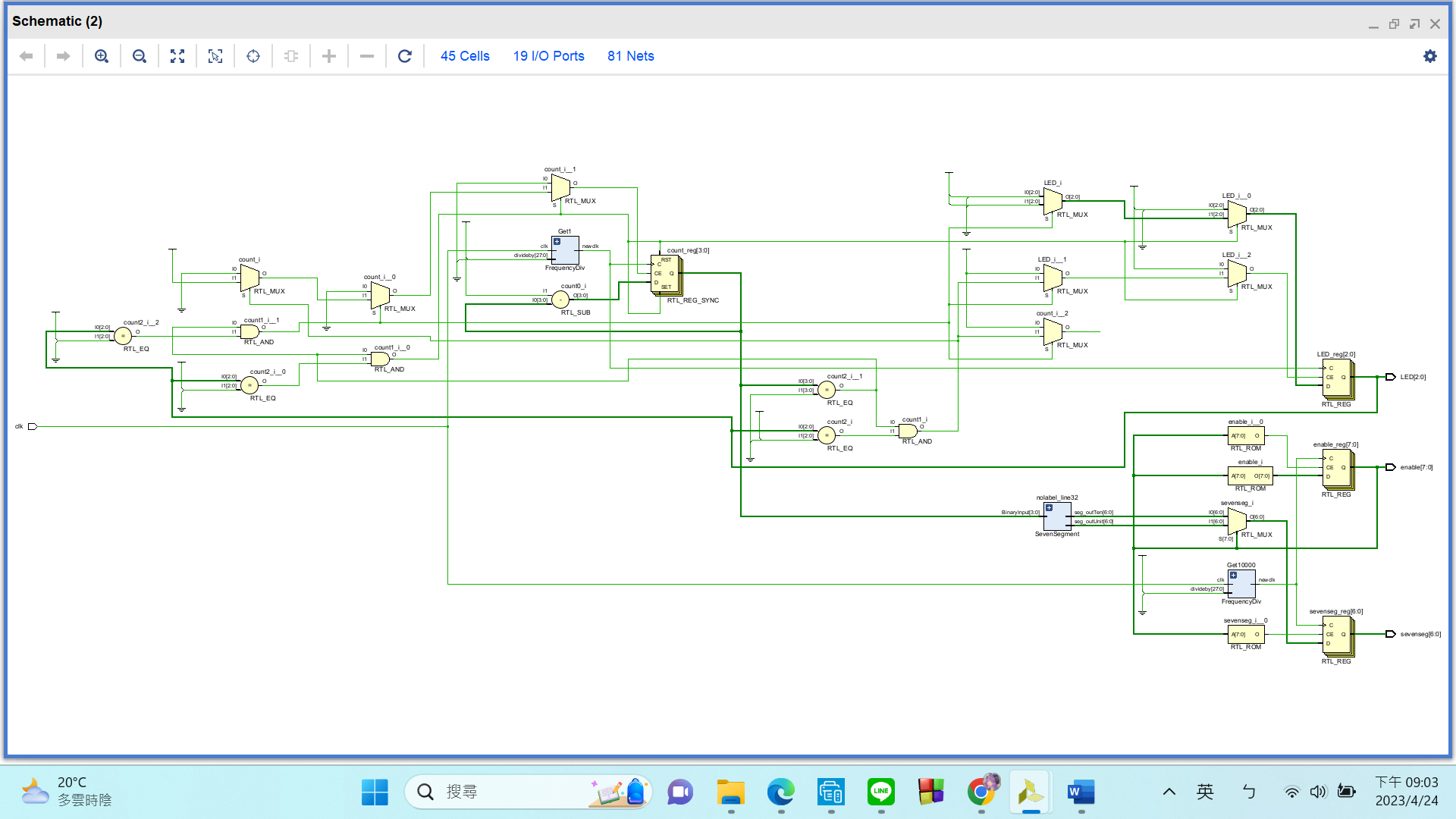
ii. When the timer reaches 0, the LED color changes to yellow, and the timer counts down from 5 to 1.

iii. When the timer reaches 0 again, the LED color changes to green, and the timer counts down from 10 to 1.

iv. When the timer reaches 0 for the third time, repeat step i.

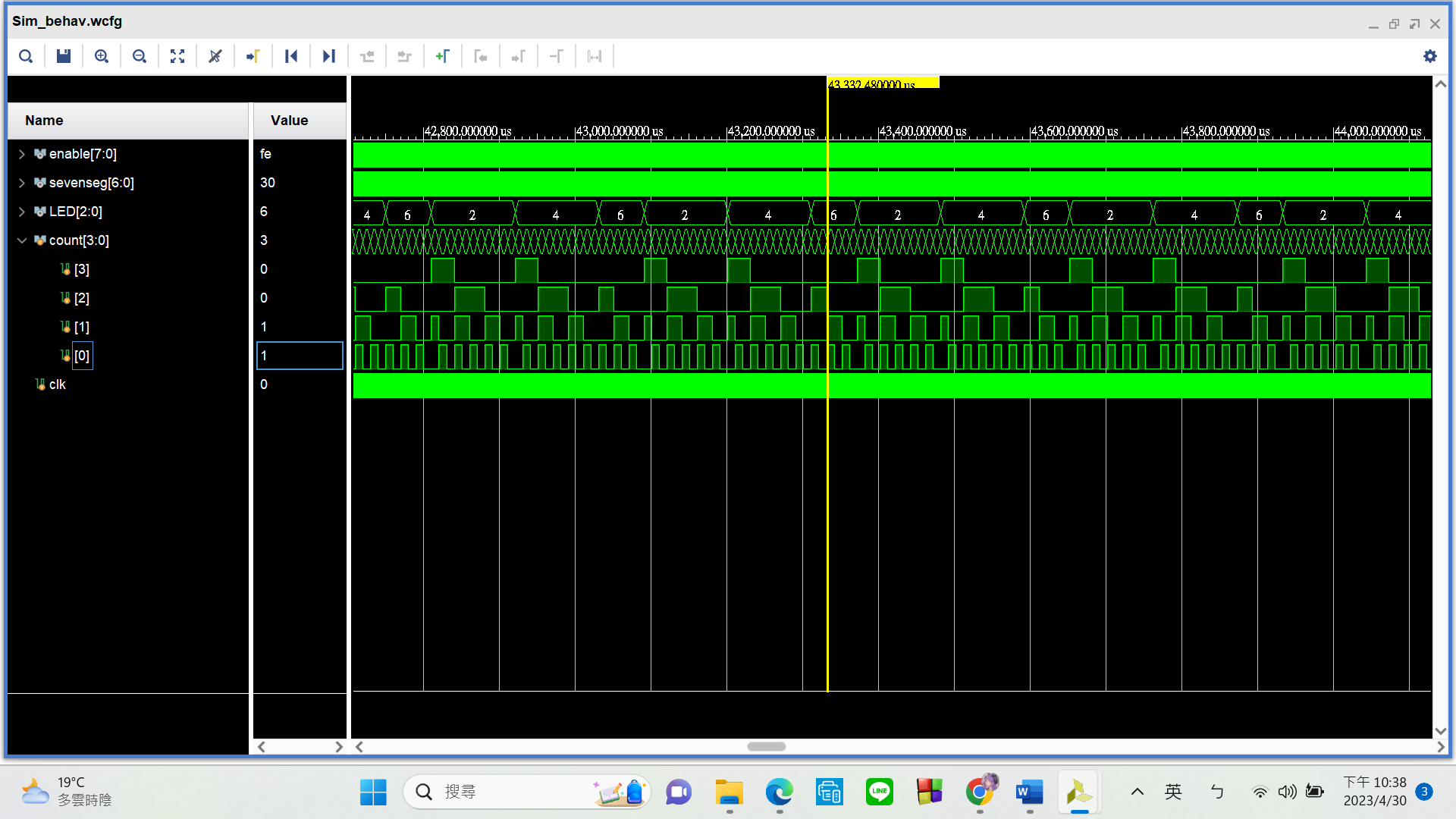
sample-802 <https://youtu.be/Uajy9qdueZc>

Experiment Data

RTL Schematic

Simulation results

(Design your own testbench)



Take a look at the “LED” reg:

When “LED” = **4**, that means its currently **100**, which should make the FPGA’s tri-color LED glow **red**.

When “LED” = **6**, that means its currently **110**, which should make the FPGA’s tri-color LED glow **yellow**.

When “LED” = **2**, that means its currently **010**, which should make the FPGA’s tri-color LED glow **green**.

Additionally, notice how the “**4**” and “**2**” sections takes up around **twice the time** “**6**” does. This corresponds to how when the counter should count down from 10 to 0 when the light is red or green; the counter should count down from 5 to 0 when the light is yellow. **The counter is also working as intended**, you can see how it’s counting down correctly from 10 to 0 or 5 to 0.

Practical results on the FPGA board

static record (photo, screen capture) or dynamic record (video link)

<https://youtu.be/C6ON80YtrZU>

OPTIONAL: design process

**Let’s look at this code in two parts:**

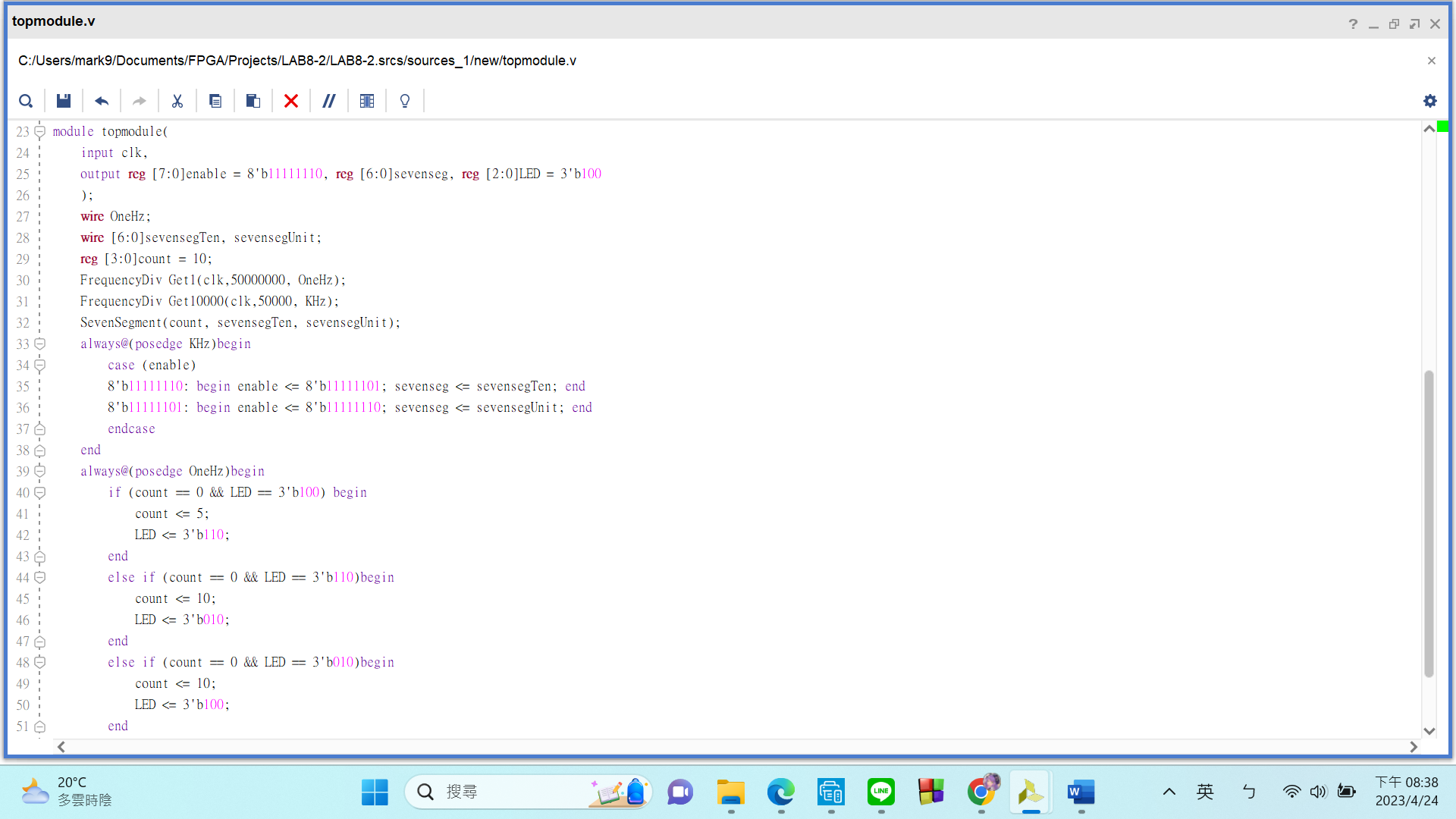
1. **Basic:**

I have two submodules in this top module, “FrequencyDiv” and “SevenSegment”. Here’s their purpose:

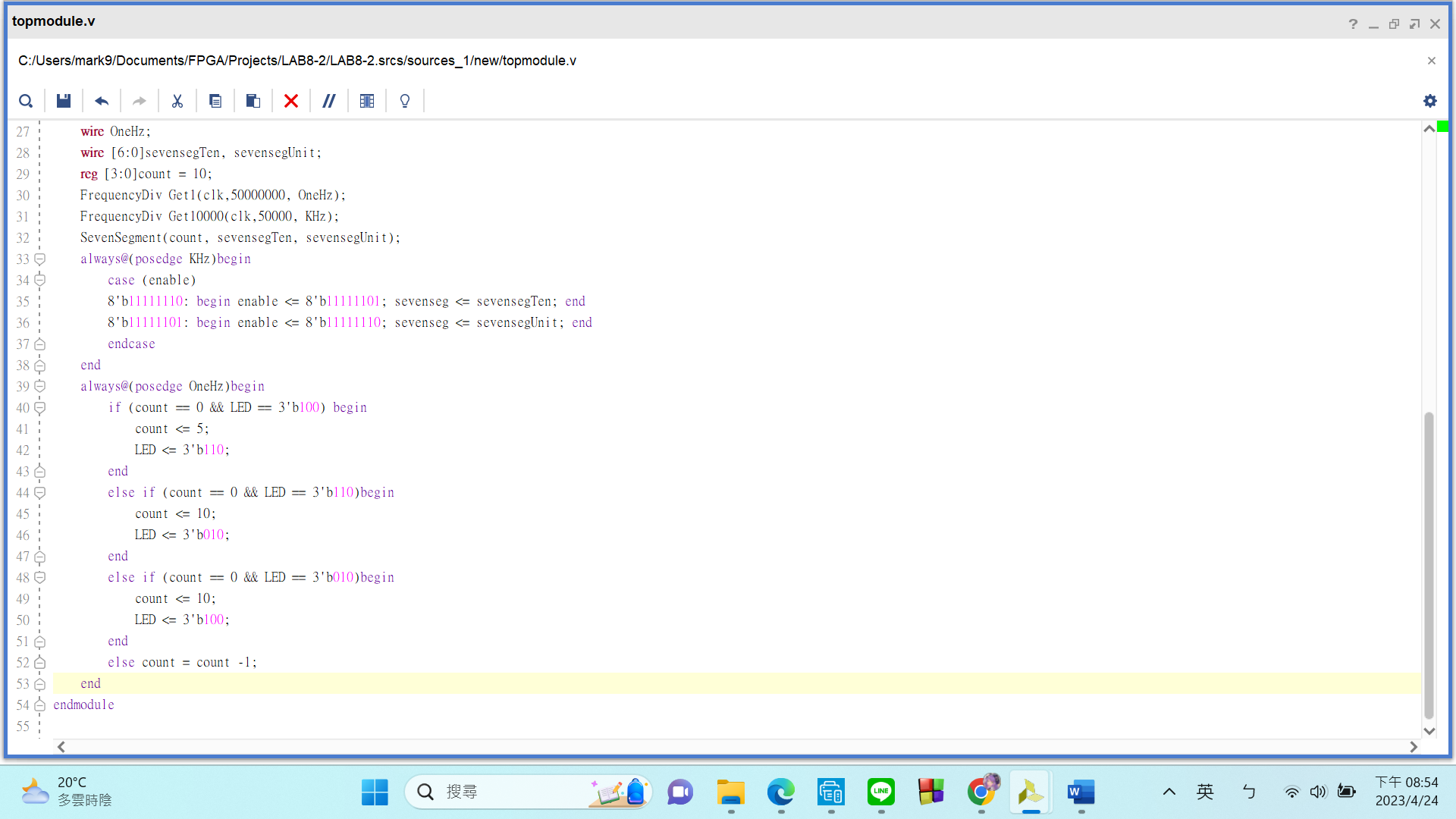
FrequencyDiv: simple frequency divider

SevenSegment: **converts** binary numbers into what we see on **seven segment displays**

The first always block here **quickly toggles between enabling the tens digit seven seg and unit digit seven seg**, allowing me to give them different outputs.



1. **Main section:**



This is the most important part of my code. I have **four conditions** in this always block that triggers every second.

1st condition: If counter reaches 0 and the LED is currently **red**, set counter to 5 and LED color to **yellow**.

2nd condition: If counter reaches 0 and the LED is currently **yellow**, set counter to 10 and LED color to **green**.

3rd condition: If counter reaches 0 and the LED is currently **green**, set counter to 10 and LED color to **red**.

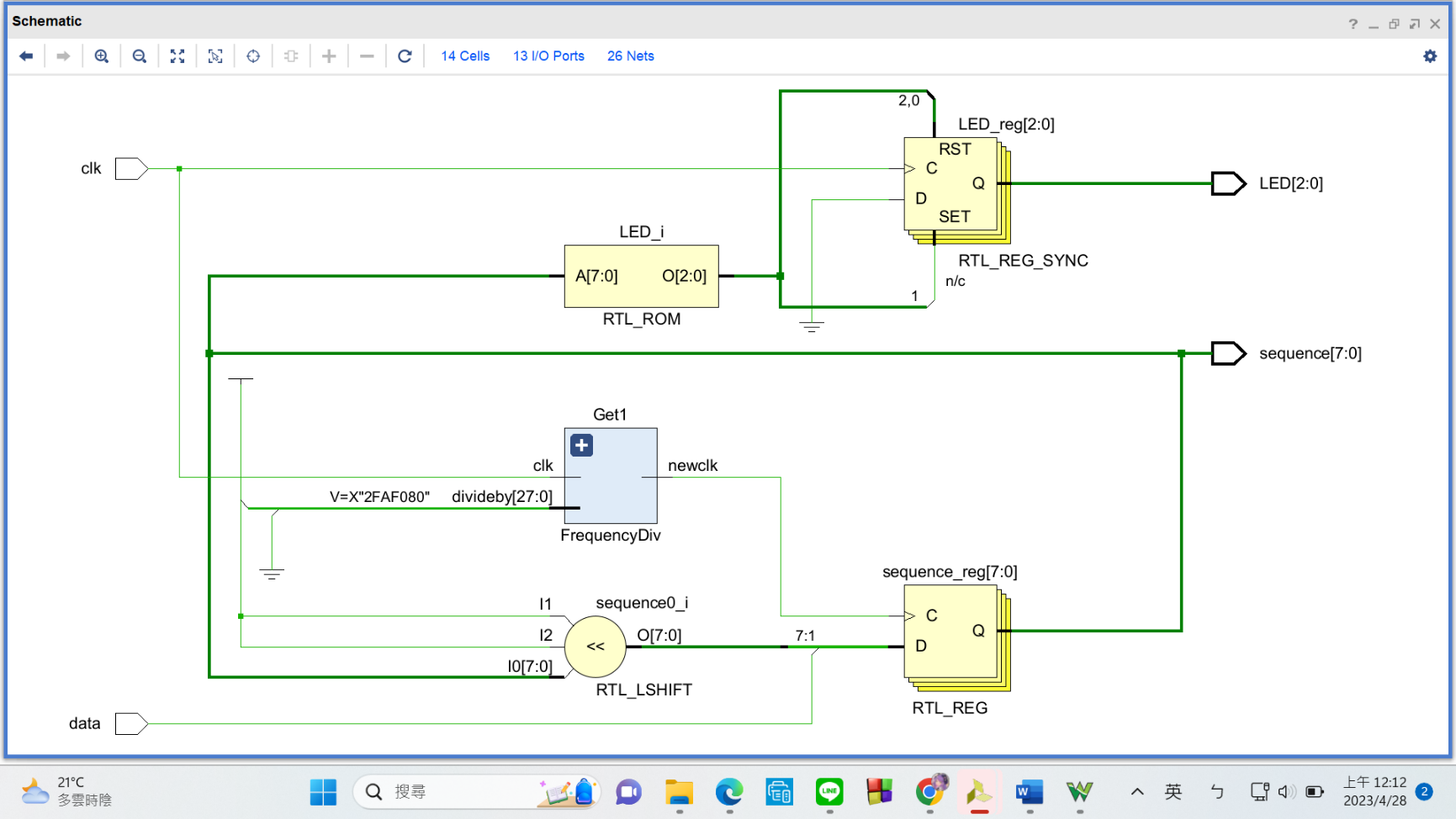
4th condition: If **none of the above** is true, **simply count down by 1.**

**FPGA Experiment-803: Sequence Detector**

**Create a sequence detector which is satisfied the following requirements:**

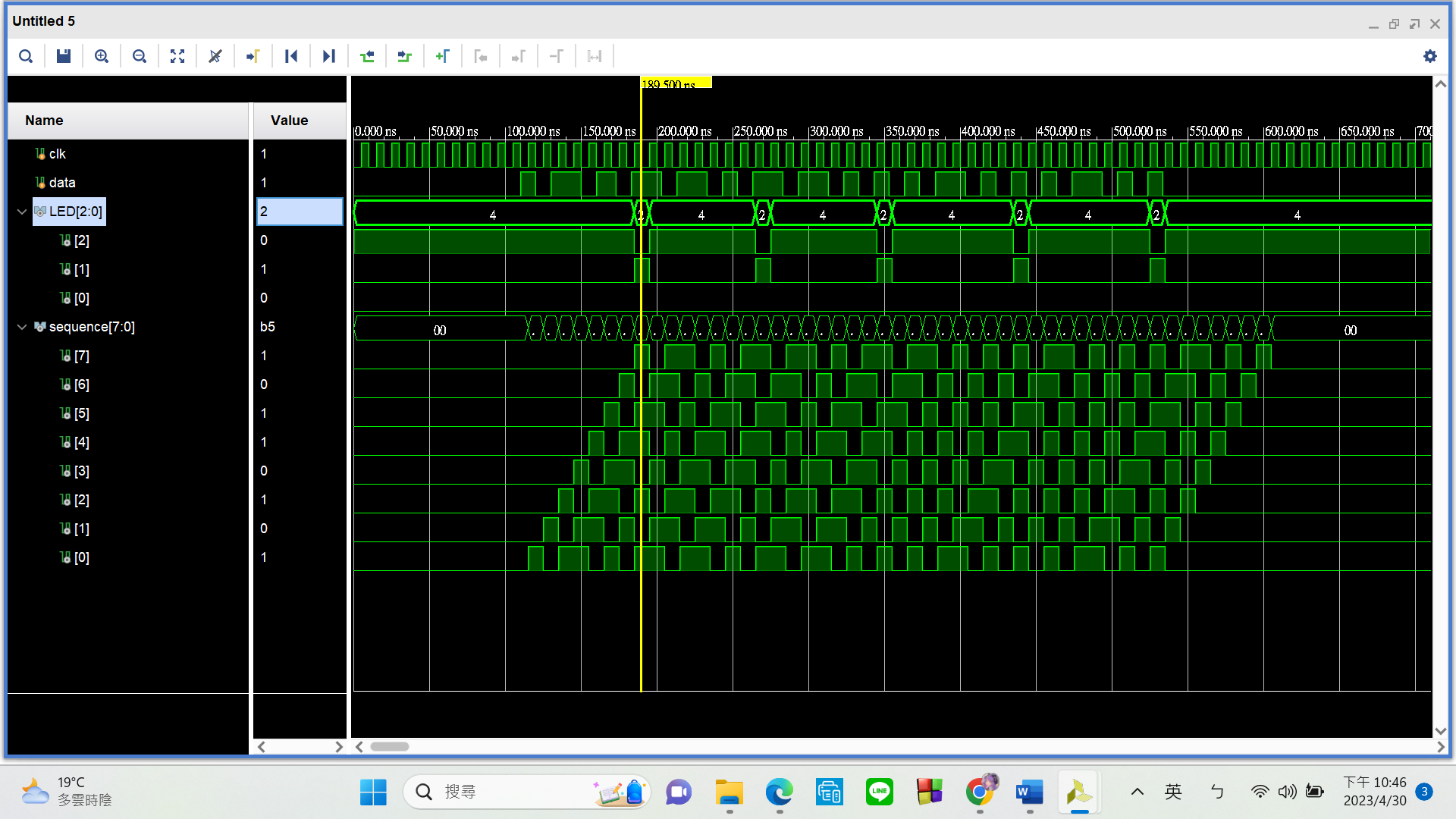
1. Modeling style: behavior
2. Input: Pmod input. Built-in CMOS oscillator (pin E3)
3. Output: 8 LEDs
4. Design a detection circuit that can detect the sequence **10110101**.
5. There is an LED indicating that the sequence has been detected.
6. Use eight LEDs to display the currently captured data

Experiment Data

RTL Schematic

Simulation results

(Design your own testbench)



As you can see, the “LED” output becomes **010** when **“sequence” is 10110101** (the sequence we want to detect). In practice, it **should make the tri-color LED on the FPGA** **green**. For the rest of time, when **“sequence” isn’t 10110101**, “LED” stays at **100**, telling the tri-color LED to glow **red**.

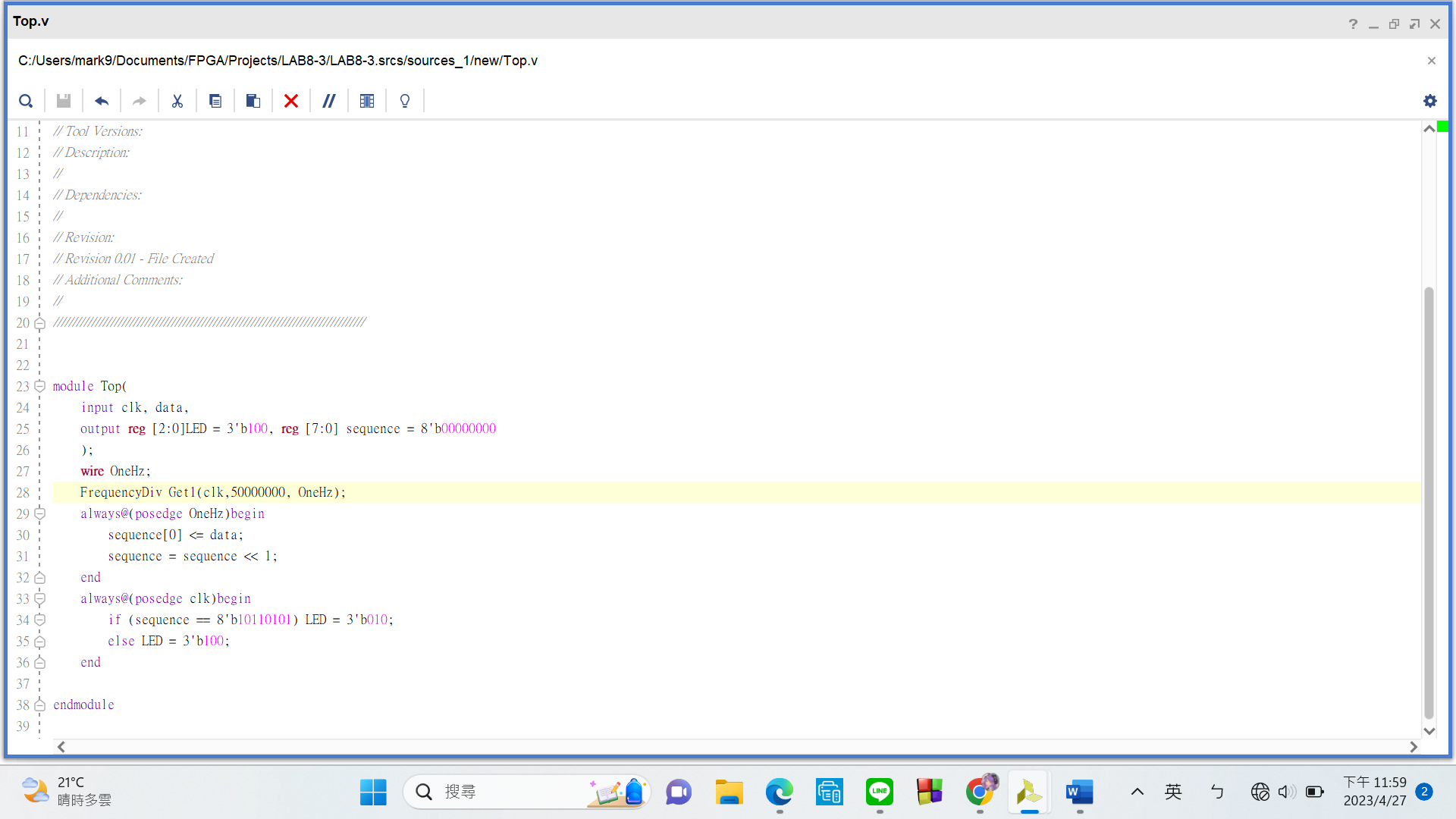
Practical results on the FPGA board

static record (photo, screen capture) or dynamic record (video link)

<https://youtu.be/FyPHdcDFnFg>

OPTIONAL: design process

**My code here is fairly simple, there’s only 1 part:**



I declared an **8-bit-reg called “sequence”**, and it is connected to **8 of the LEDs** at the bottom of the FPGA to indicate what the current sequence is. The input is passed in through “data”. I **provided input test data using our Analog Discovery 2’s “Pattern” mode**. **New data will be passed in at a frequency of 1Hz**. (one bit per second) In my always block, I used the **left shift operator “<<” to shift “sequence” by 1 to the left and then assign the new input to the rightmost bit.** The second always block tells the tri-color LED to **glow green when** the **correct** sequence is found or **glow red otherwise**.

