Digital Laboratory Lab06 Report

(Experiment Record Template)

**FPGA Experiment-601: ID Marquee**

**Create a marquee display which is satisfied the following requirements:**

1. Modeling style: behavior
2. Input: FPGA built-in CMOS oscillator (i.e., pin E3)
3. Output: **8-digit** 7 segment display
4. The sequence elements correspond to your student ID and follow the format of

[student ID] + [1-digit blank]. 111511076(space)

1. The marquee can move in two directions: MSB to LSB or LSB to MSB."

sample-601

* 1. [0310165] + [blank] <https://youtu.be/iBXXnb0YeqI>
  2. [2022-1850] + [blank] + [AM] + [blank] + [PM] + [blank] <https://youtu.be/joIULBs6zGw>

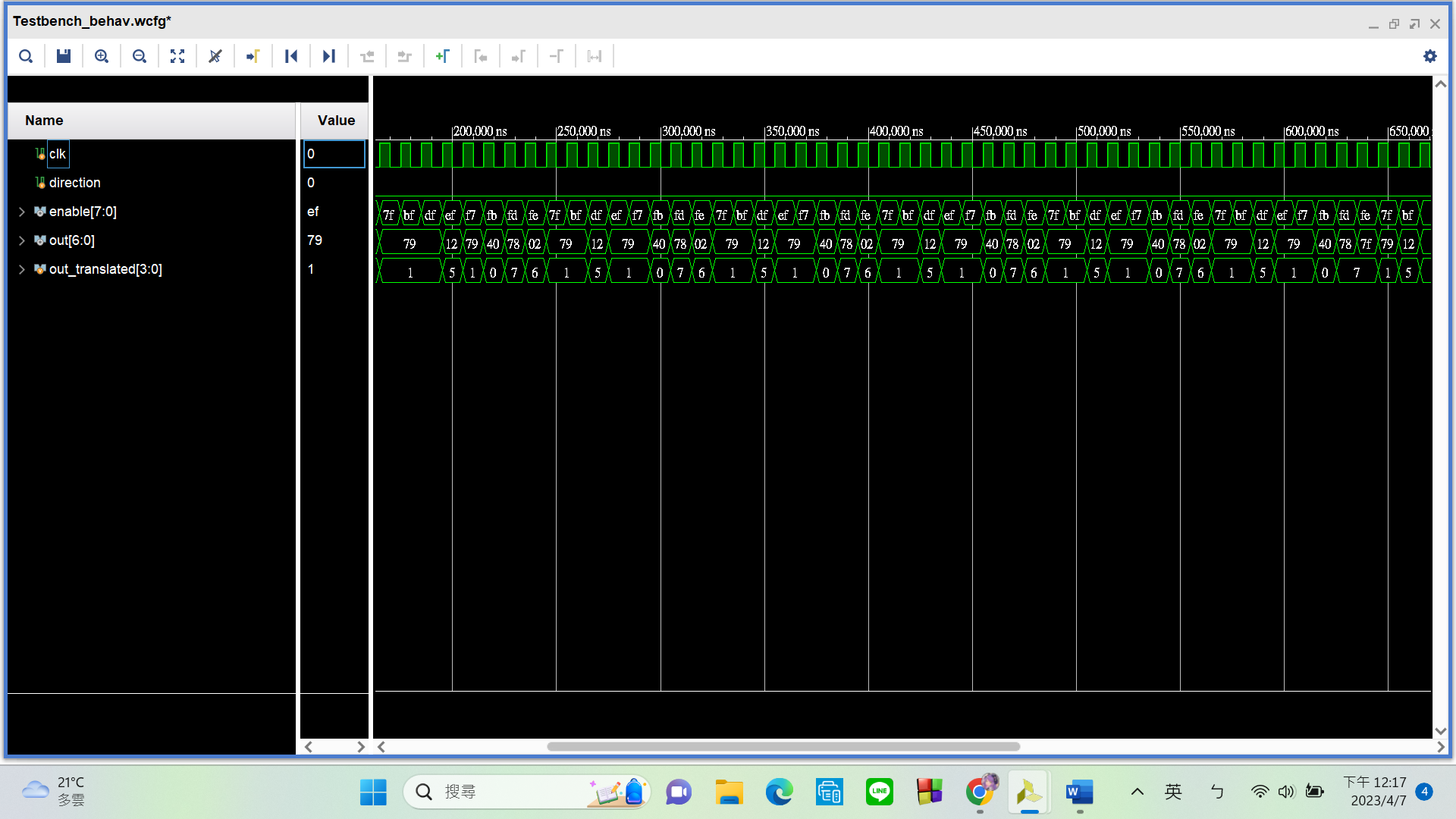
Experiment Data

RTL Schematic



Simulation results

(Design your own testbench)



Notice that I am **correctly iterating through my ID** num: **111511076** over and over again. Additionally, the **enable output is also changing** as intended, allowing me to pass different numbers to different displays.

Practical results on the FPGA board

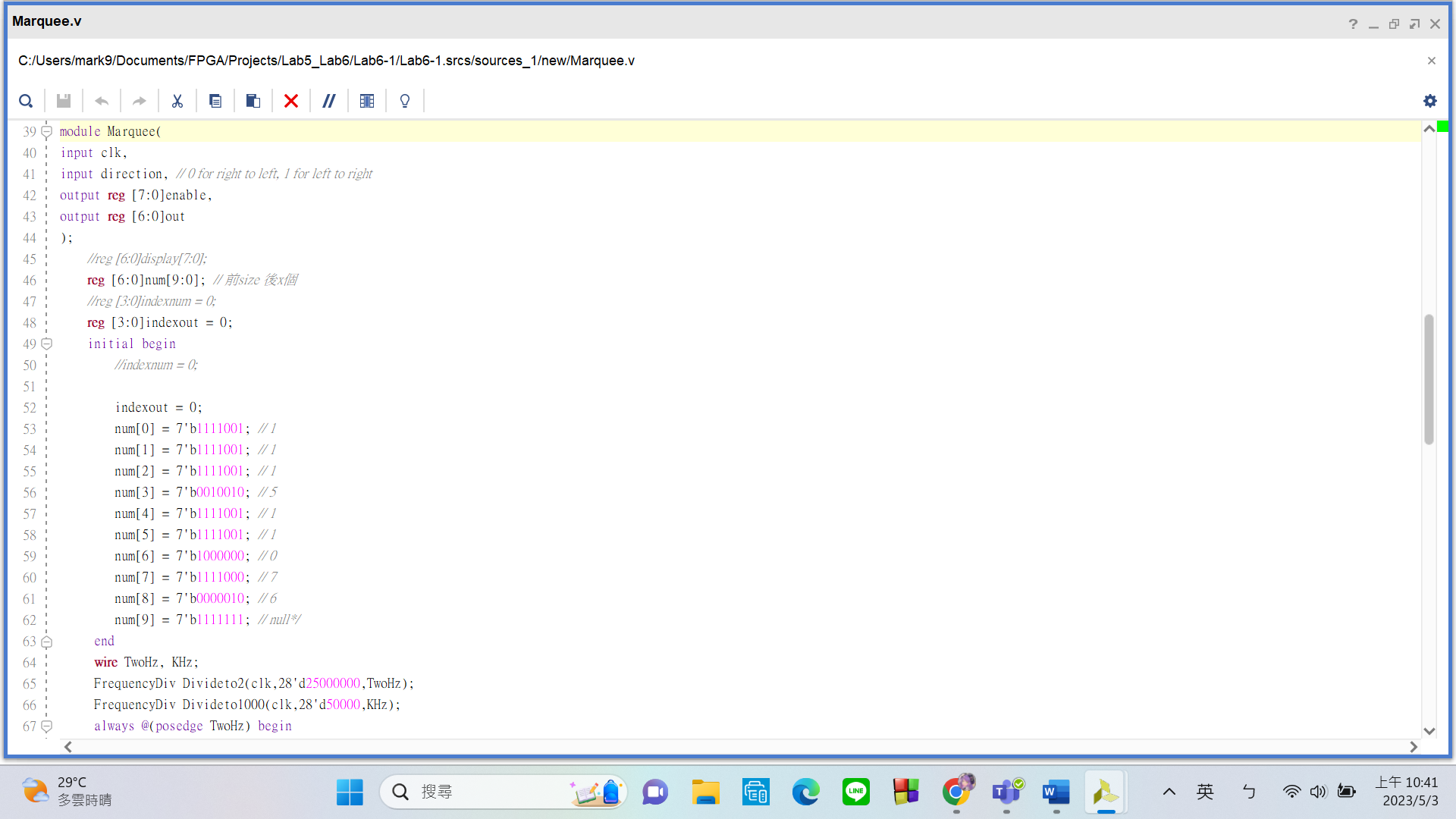
static record (photo, screen capture) or dynamic record (video link)

<https://youtu.be/xHVlA4mQWFg>

OPTIONAL: design process

**Let’s look at my code in 3 parts:**

1. **Reg Array Declaration**

My top module – Marquee has

Input:

* clk
* **direction**: allows me to control the **direction in which the numbers are moving**

Output:

* enable: turns on the seven segment
* out: controls the A to G of the seven segment display

Initially, I declared a **10 by 7 reg array** called “num”. Since there should be **10 numbers on the marquee** and each number’s **seven seg form should be stored in 7 bits**, hence the size.

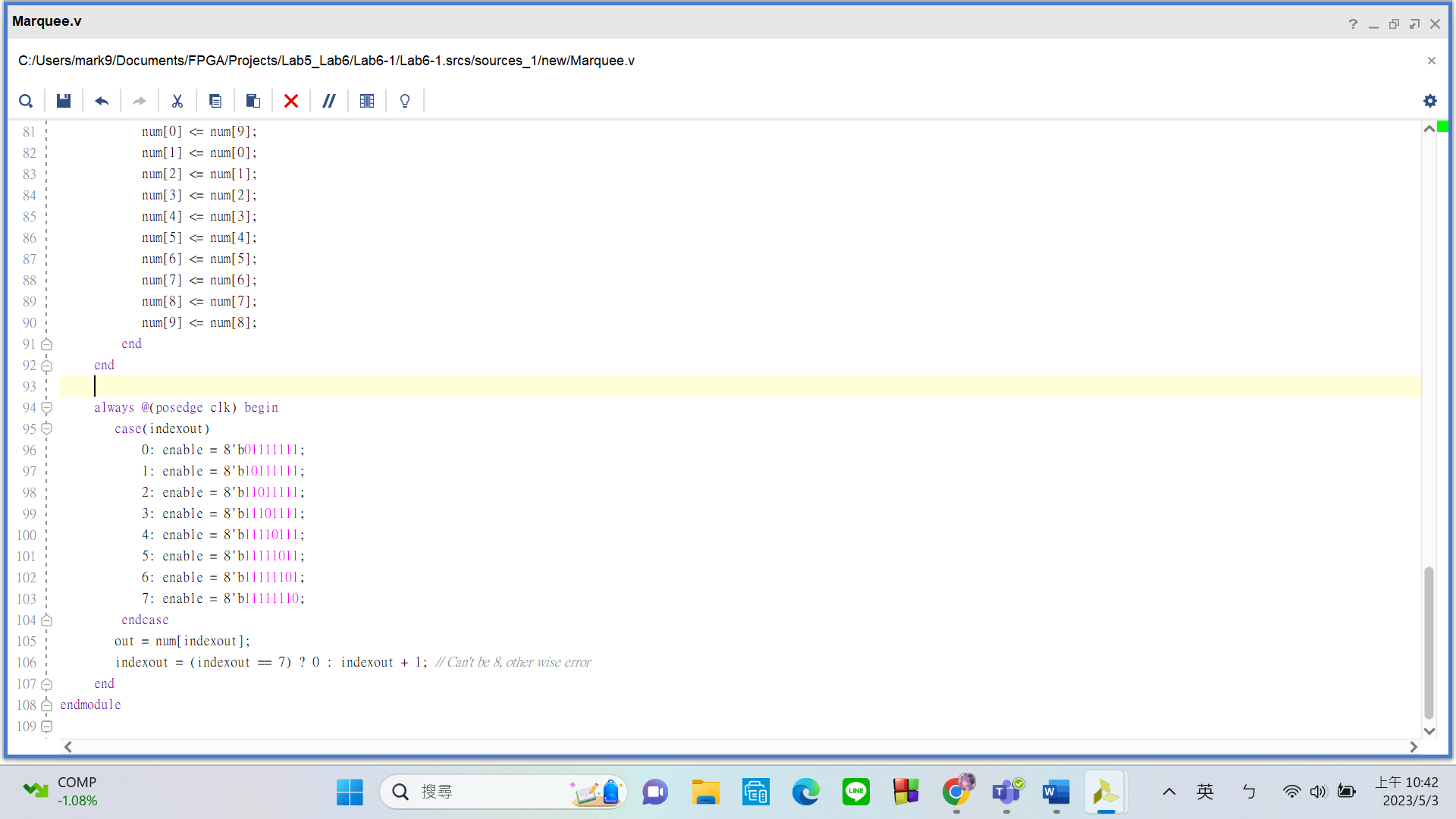
1. **Left Shift/Right Shift**

Here I used the same **FrequencyDiv** module previously used in Lab5. I slightly modified it to allow me to pass in values, instead of having it locked at 1 Hz. **The value passed in is given by: 100MHz ÷ Desired Frequency÷2**

In the always block, I am **shifting the bits** to the left/right (depends on the direction we want). num[index] denotes the nth 7-bit, so **this code here basically shifts 7 bits to the left/right at a frequency of 2Hz.** However, the **leftmost/rightmost 7 bits** are not forgotten. They will be **moved back to position 0**.

To shift the bits, we **must use non-blocking assignment “<=” instead of the blocking assignment “=”**. “<=” allows all the actions to be performed at the same time, preventing any data from being overwritten.

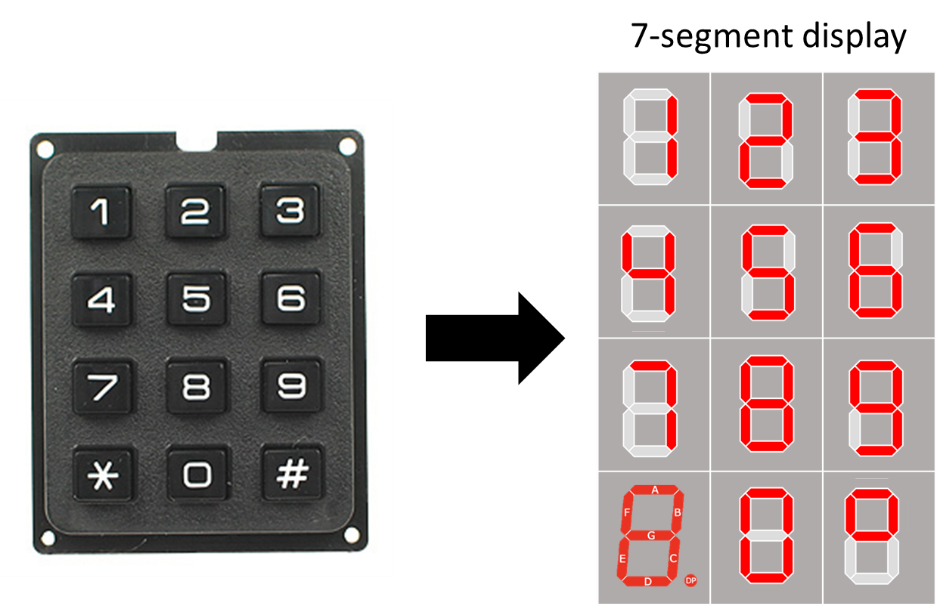
1. **Enable Swapping**

All 8 seven segment displays have the same data input. That’s means I can’t pass different inputs to them at the same time. Please take a look at the always block. The method we must use is to **toggle between these 8 displays at a very fast speed** (I used the undivided 100MHz clk). At such a speed, the LEDs act as if they are always on. In this way, we can give them different numbers.

**FPGA Experiment-602: Keypad + 7-seg Display**

**Create a display driver which is satisfied the following requirements:**

1. Modeling style: behavior
2. Input: keypad (7-pin or 8-pin) [pinout definition](https://nycu1-my.sharepoint.com/:b:/g/personal/ccata_m365_nycu_edu_tw/EQnUcihujq9MphW7GsiA6r4B_bw7QQcqL-QYBgVrQ0RqXA?e=EvADHS)
3. Output: **exactly one** 7 segment display = 1-digit number
4. Display the pressed number or symbol on the 7-segment display.

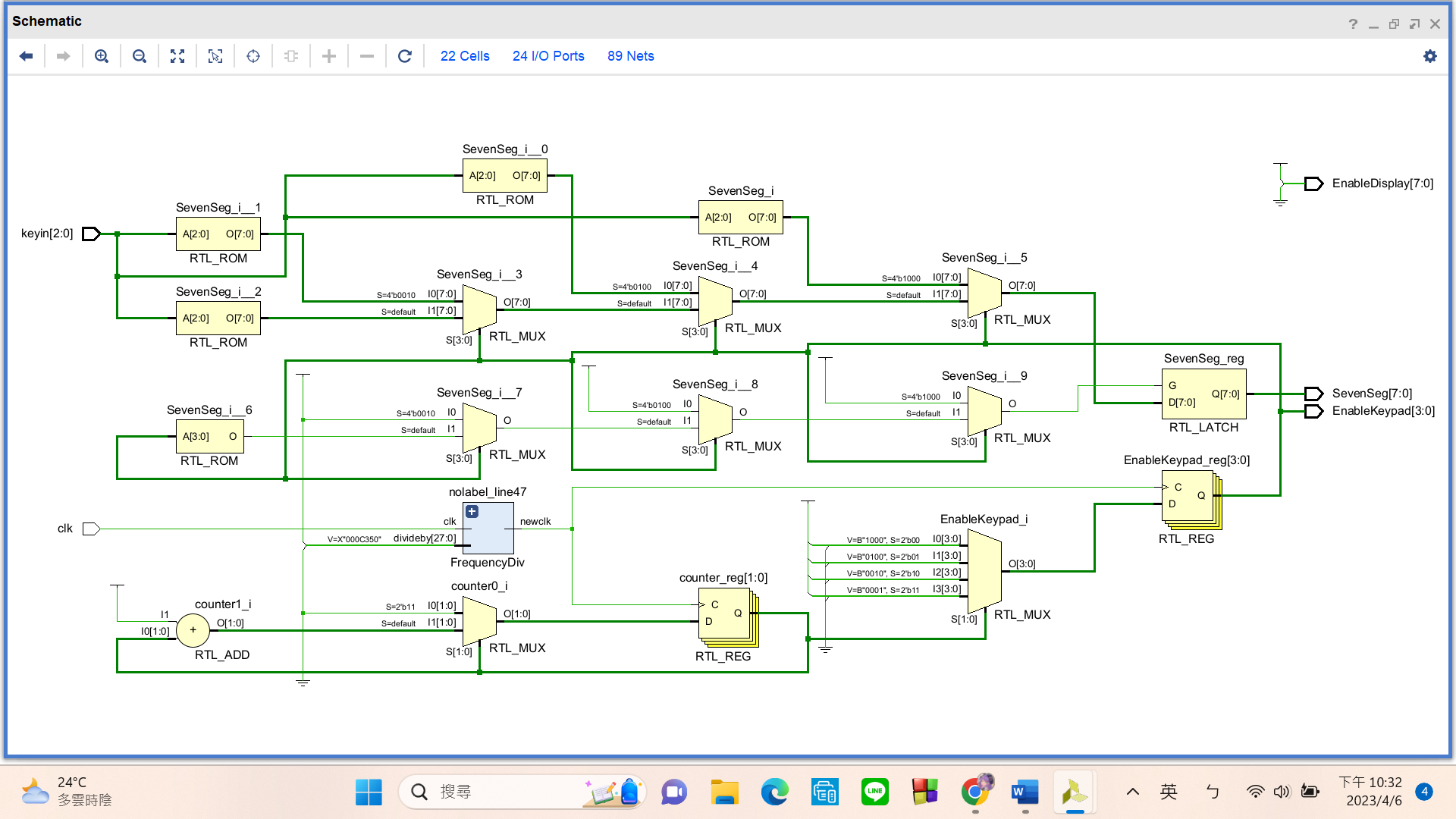


Note: **Press** and **click** are two different actions that can be performed on a computer mouse or a touchpad. Press means to push down on something with force, while click means to make a short and sharp sound by pressing and releasing a button.

sample-602 <https://youtu.be/UqguCZIiKV8>

Experiment Data

RTL Schematic



Simulation results

(Design your own testbench)



The numbers that the rows(EnableKeypad) and columns(keyin) correspond to are correct.

Practical results on the FPGA board

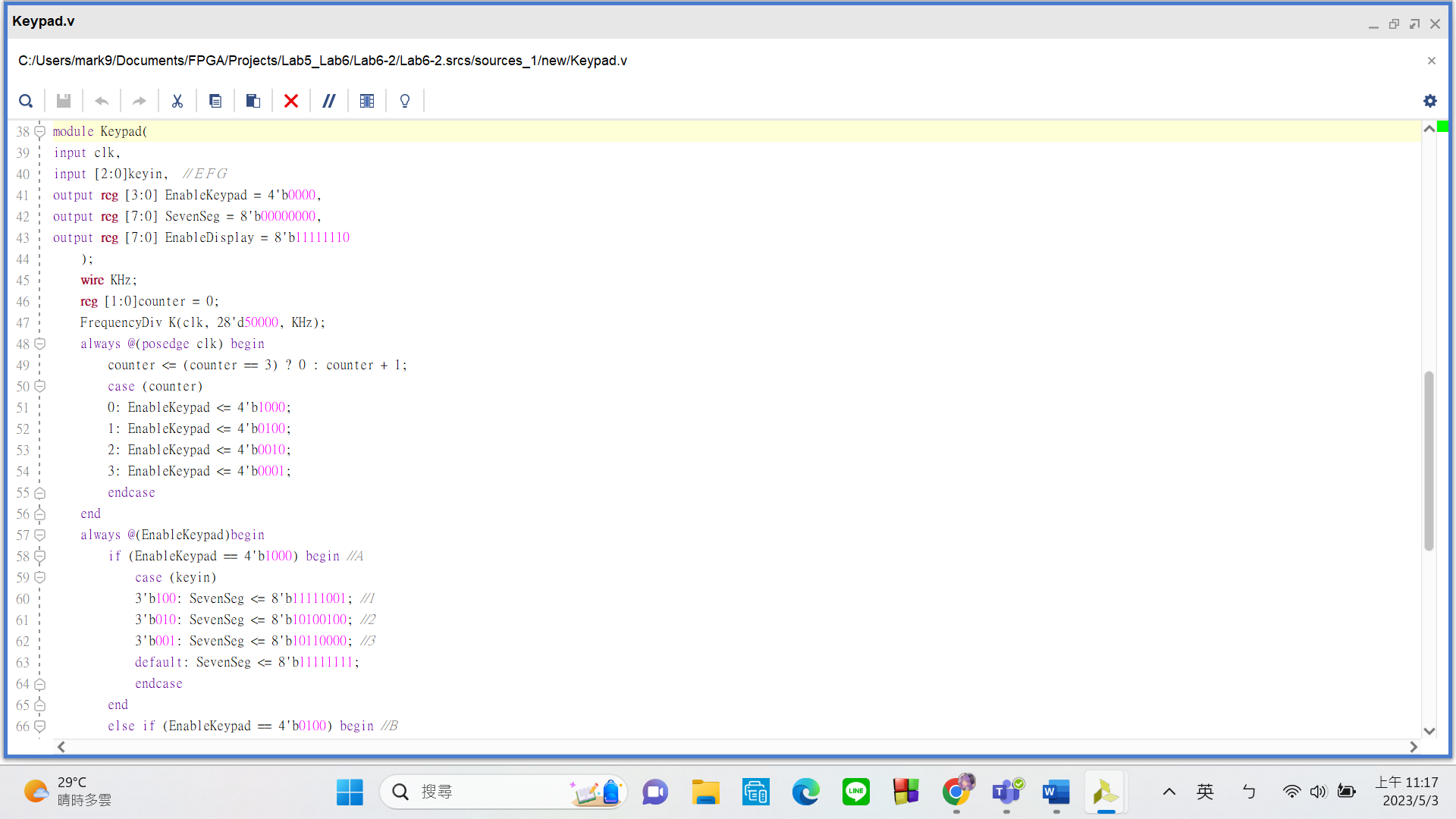
static record (photo, screen capture) or dynamic record (video link)

<https://youtu.be/CHZznBU387Y>

OPTIONAL: design process

**Let’s look at my code in 2 parts:**

1. **Enable Keypad**



My top module – Marquee has

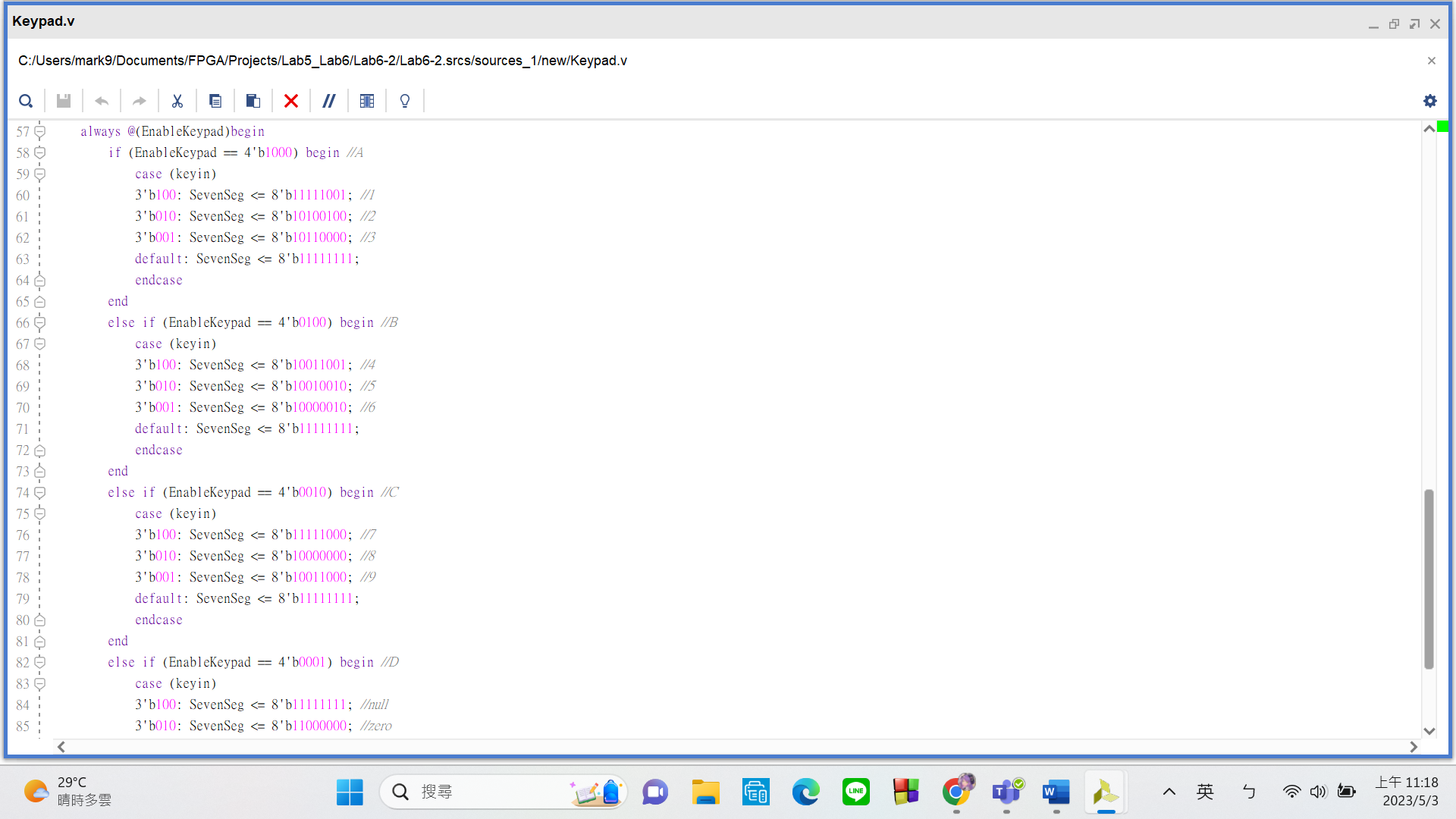
Input:

* clk
* keyin: takes the signal created when we press the keys on the keypad

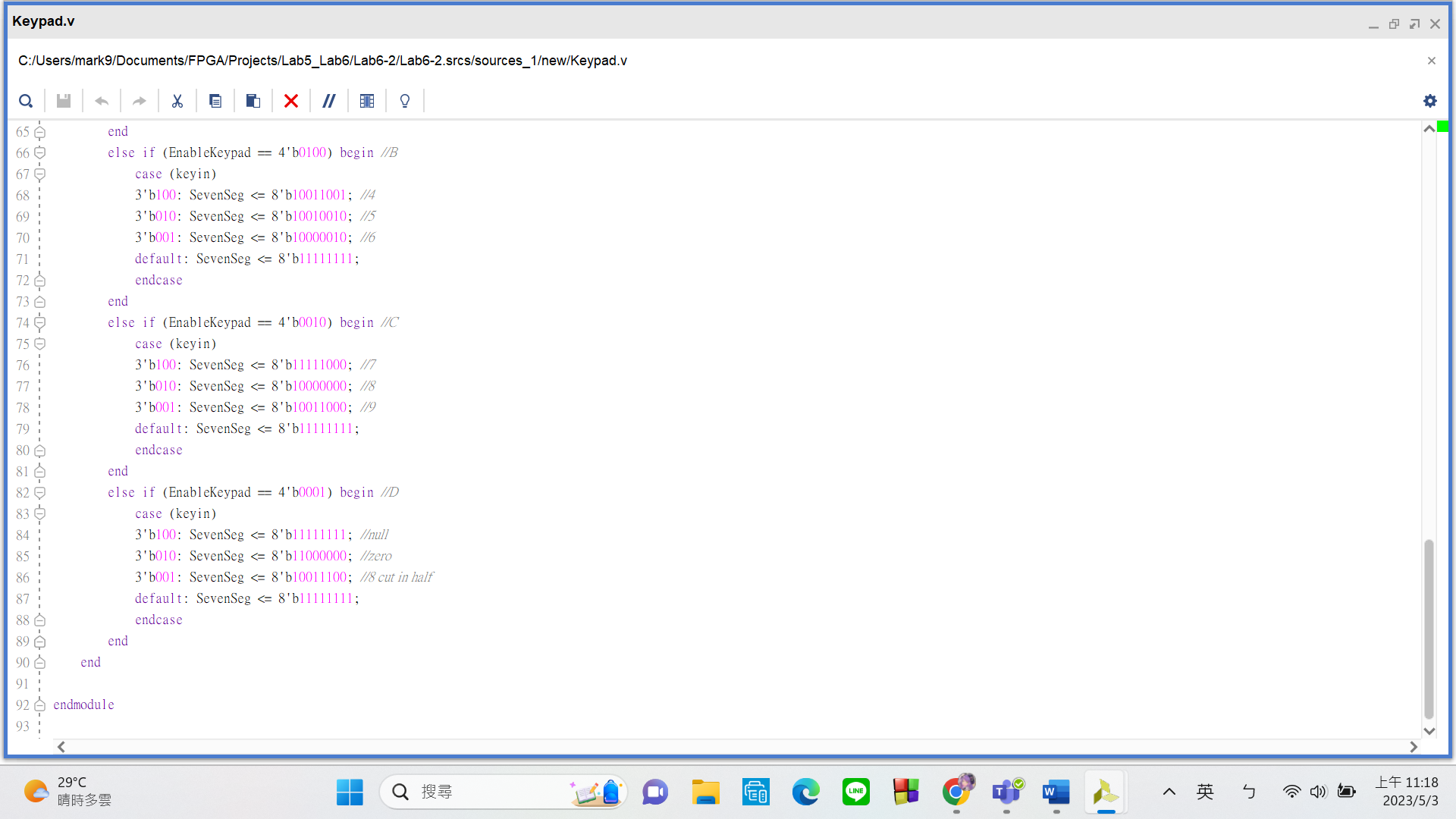
Output:

* EnableKeypad: repeatedly scans through the 4 rows o the keypad so that every row is enabled
* SevenSeg: controls A to G of the seven segment display
* EnableDisplay: turns on the seven segment display

In the always block, I let the **“EnableDisplay” iterate between 4 states at a frequency of 100MHz**. Each state only enables 1 row. Enabling more than one row will prevent us from determining the correct number each key corresponds to.

1. **Correspond Rows and Columns to Number**

I **used 4 if statements that each contains a case statement** to match the activated rows and columns of the keypad to what should be displayed on the seven segment. This code is somewhat difficult to read and lengthy. The **same purpose can be achieved in a better way**, which I implemented in 6-3.



**FPGA Experiment-603: BCD Adder**

**Create a BCD adder which is satisfied the following requirements:**

1. Modeling style: behavior
2. Input: keypad (7-pin or 8-pin)
3. Output: **exactly one** 7 segment display = 1-digit number
4. Here are the arithmetic and display rules:
   1. Store the augend and addend respectively.
   2. Show the summation result in decimal format.
   3. Design a reset function to set all register values to 0."

sample-603

1. <https://youtu.be/IAYxrhJcu3U>

1 + 3 = 4

step1: set regA = 1,

step2: set regB = 3,

step3: press add button, check summation on the 7-segment display,

step4: reset all register

1. <https://youtu.be/28NkH89ExhU>

2 + 9 = 11

step1: set regA = 2

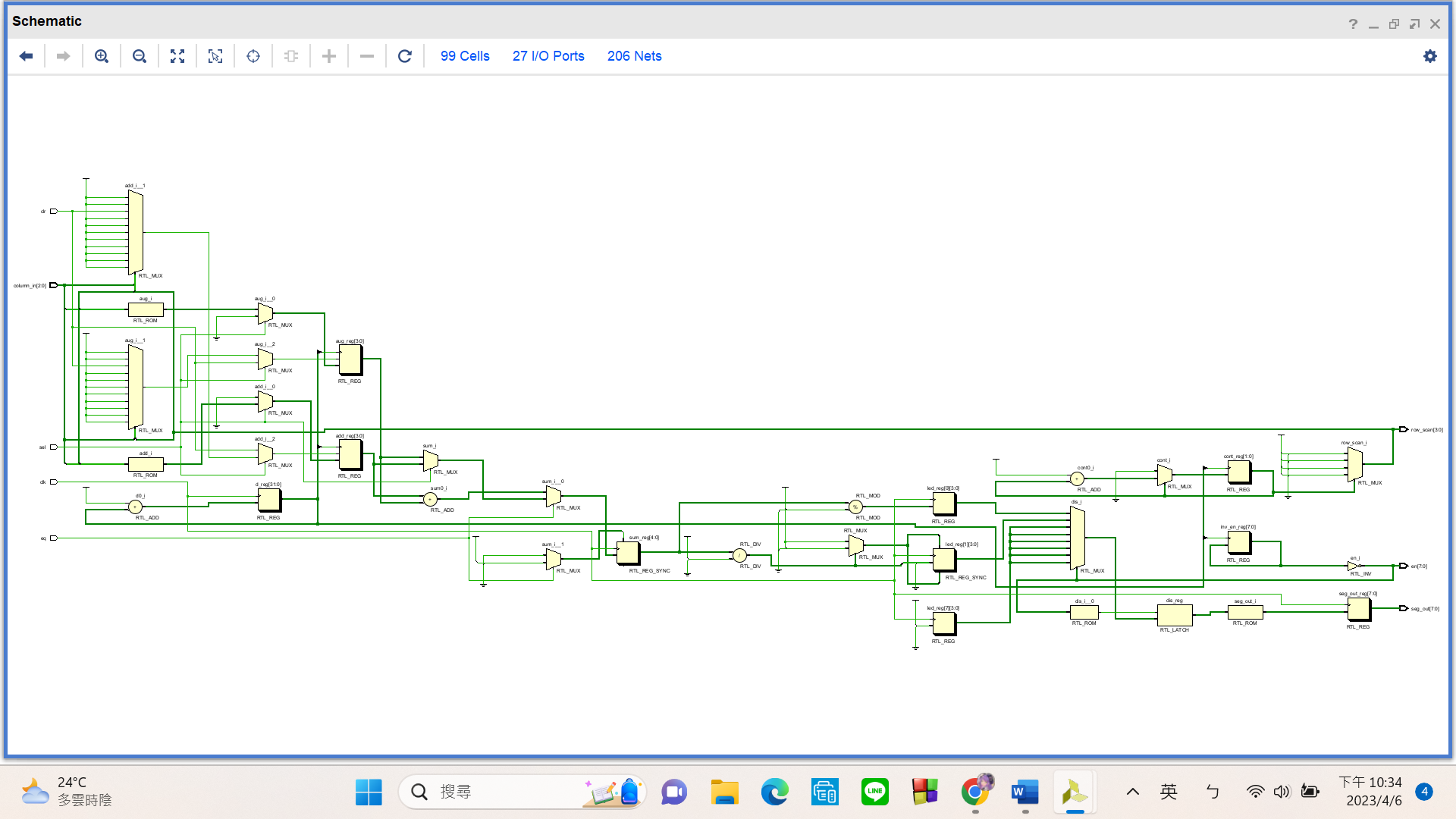
step2: set regB = 9

step3: press add button, check summation on the 7-segment display,

step4: reset all register

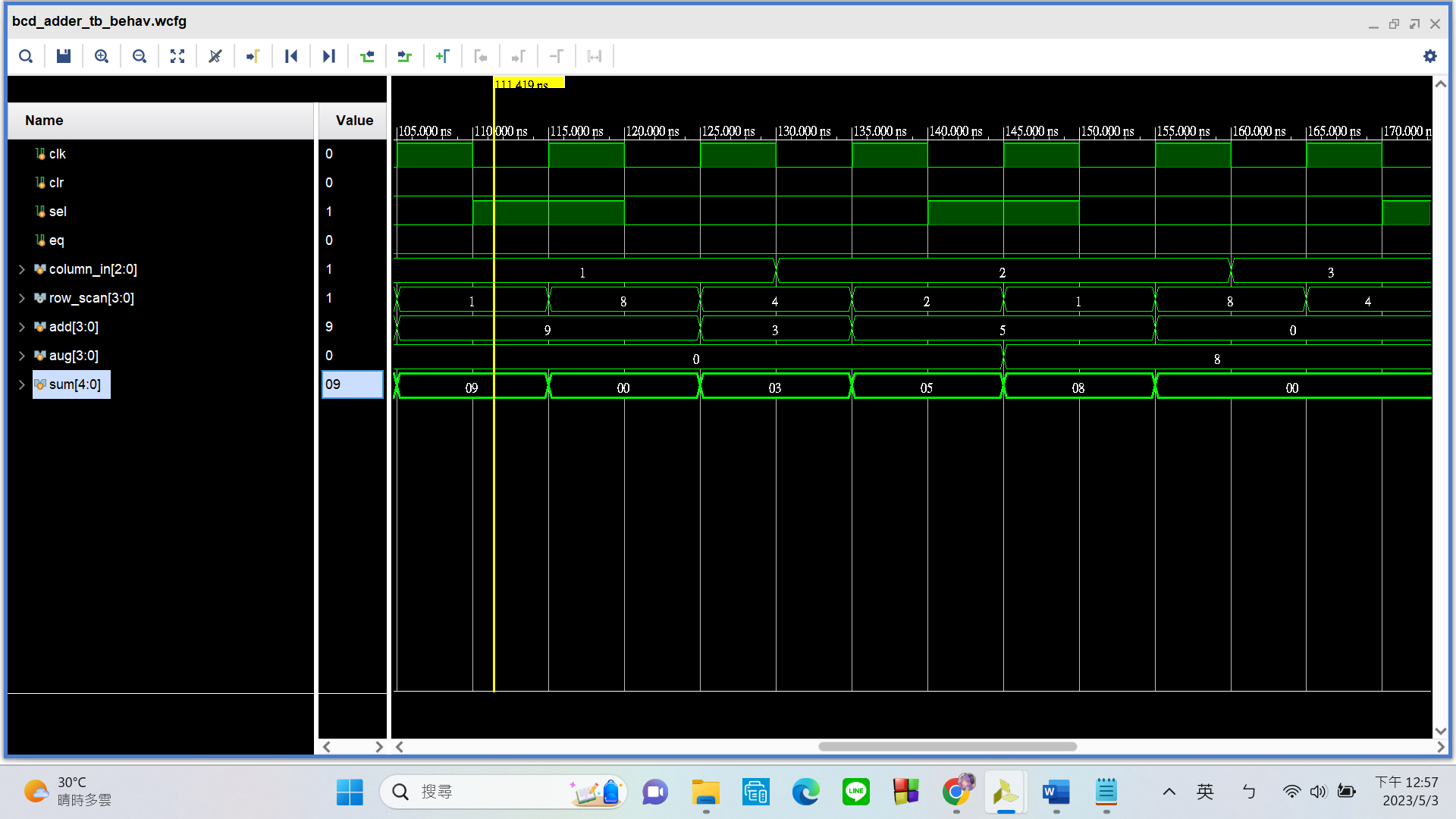
Experiment Data

RTL Schematic



Simulation results

(Design your own testbench)



Notice that the sum has the correct result after adding together the addend and the augend.

Practical results on the FPGA board

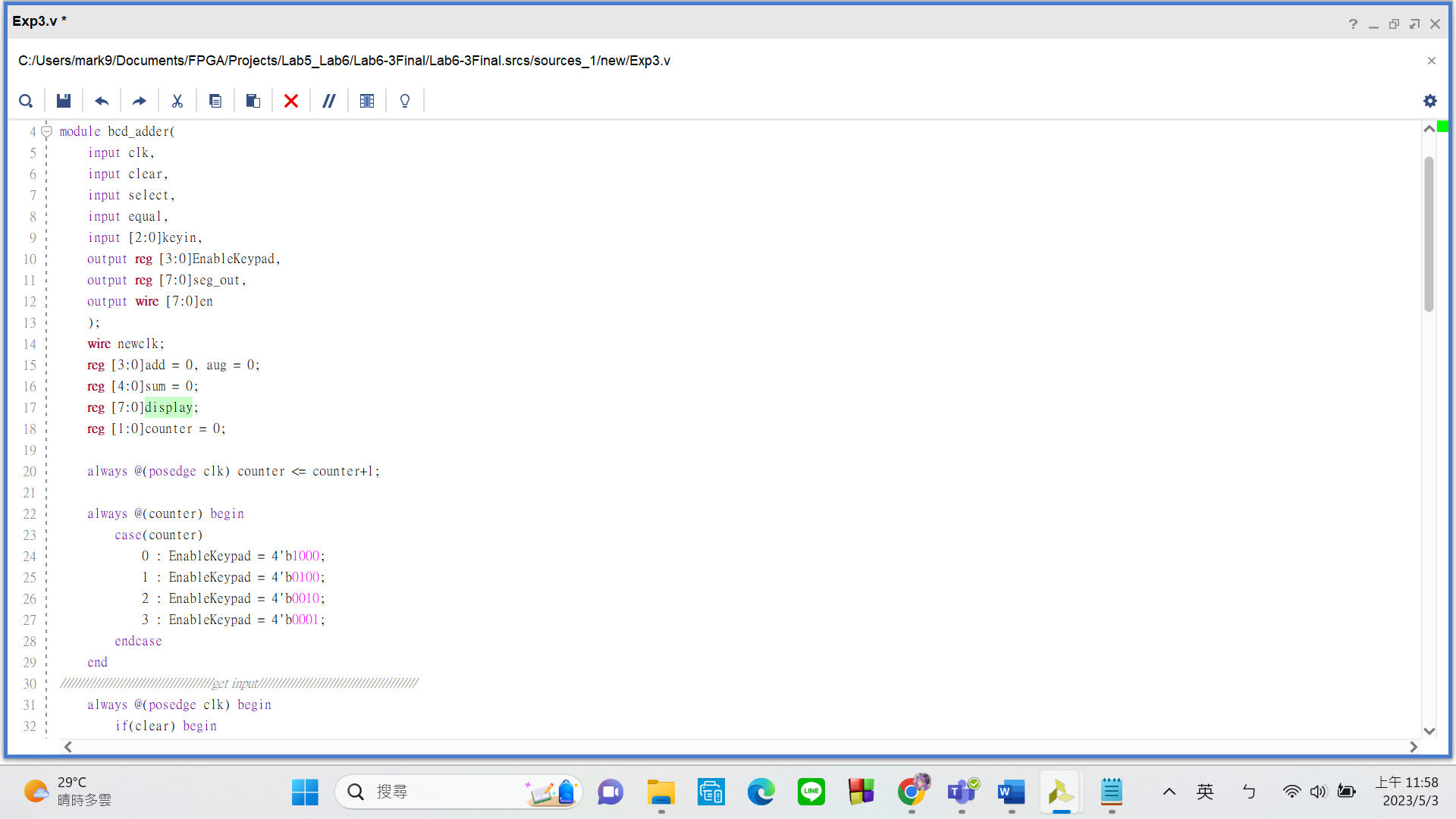
static record (photo, screen capture) or dynamic record (video link)

<https://youtube.com/shorts/btTNGZG-xKg?feature=share>

OPTIONAL: design process

**Let’s look at my code in 3 parts:**

1. **Enable Keypad**



My top module – bcd-adder has

Input:

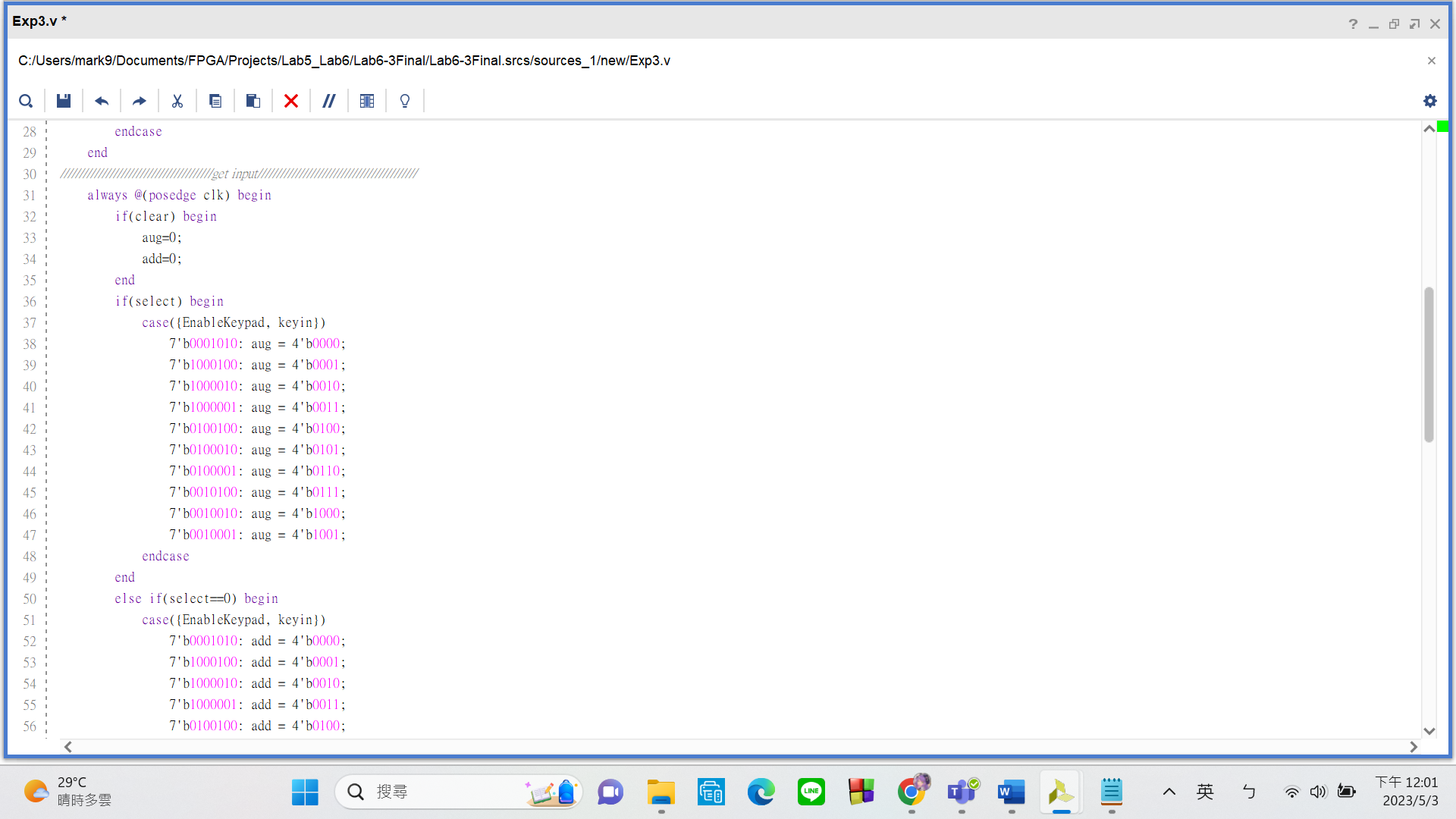
* clk
* clear: clears the numbers stored
* select: select between which reg to give value to, the augend or the addend
* keyin: takes the signal created when we press the keys on the keypad

Output:

* EnableKeypad: repeatedly scans through the 4 rows o the keypad so that every row is enabled
* seg\_out: controls A to G of the seven segment display
* en: 4turns on the seven segment display

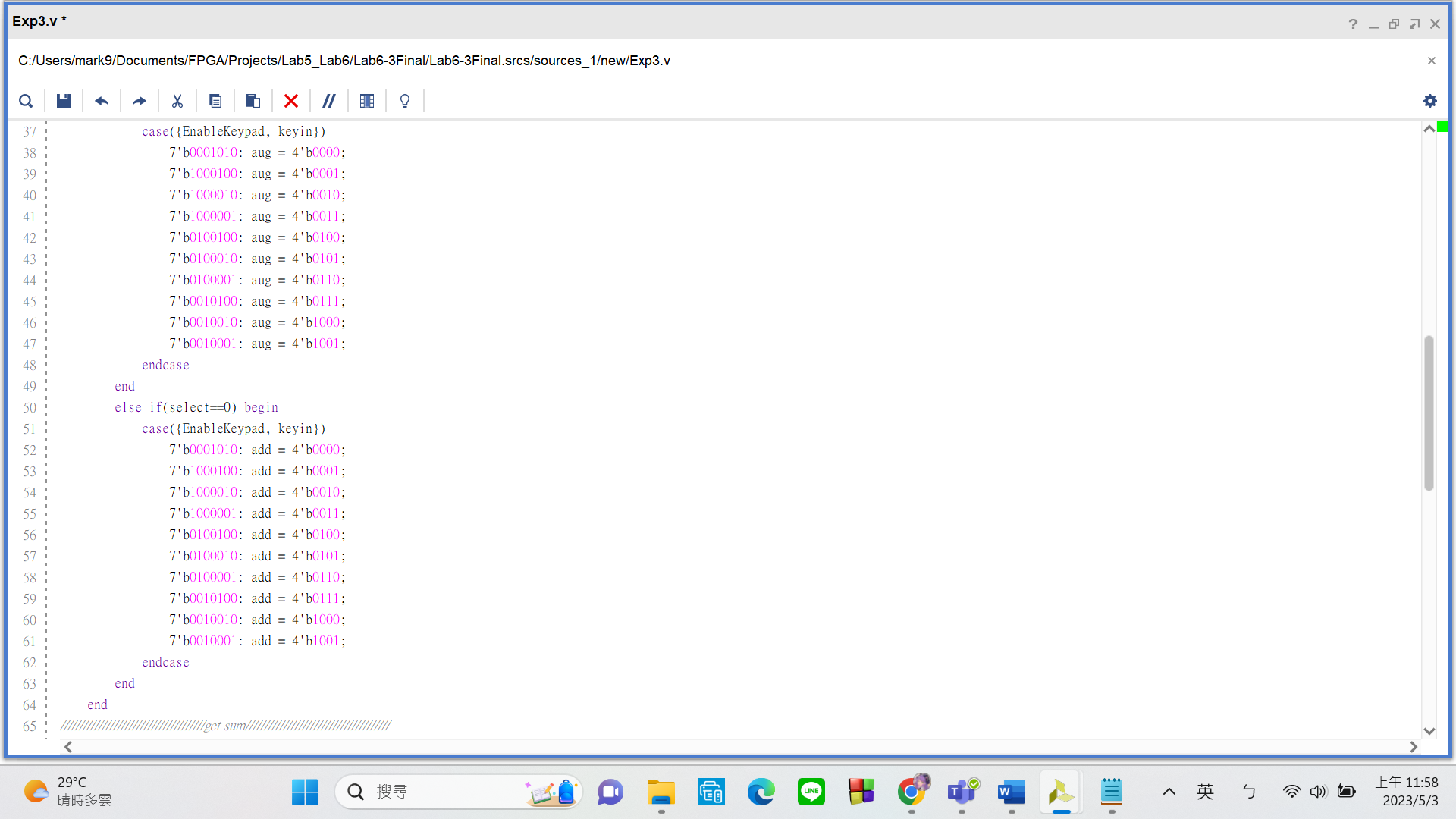
Same as 6-2, in the always block, I let the “EnableDisplay” iterate between 4 states at a frequency of 100MHz.

1. **Correspond Rows and Columns to Number**



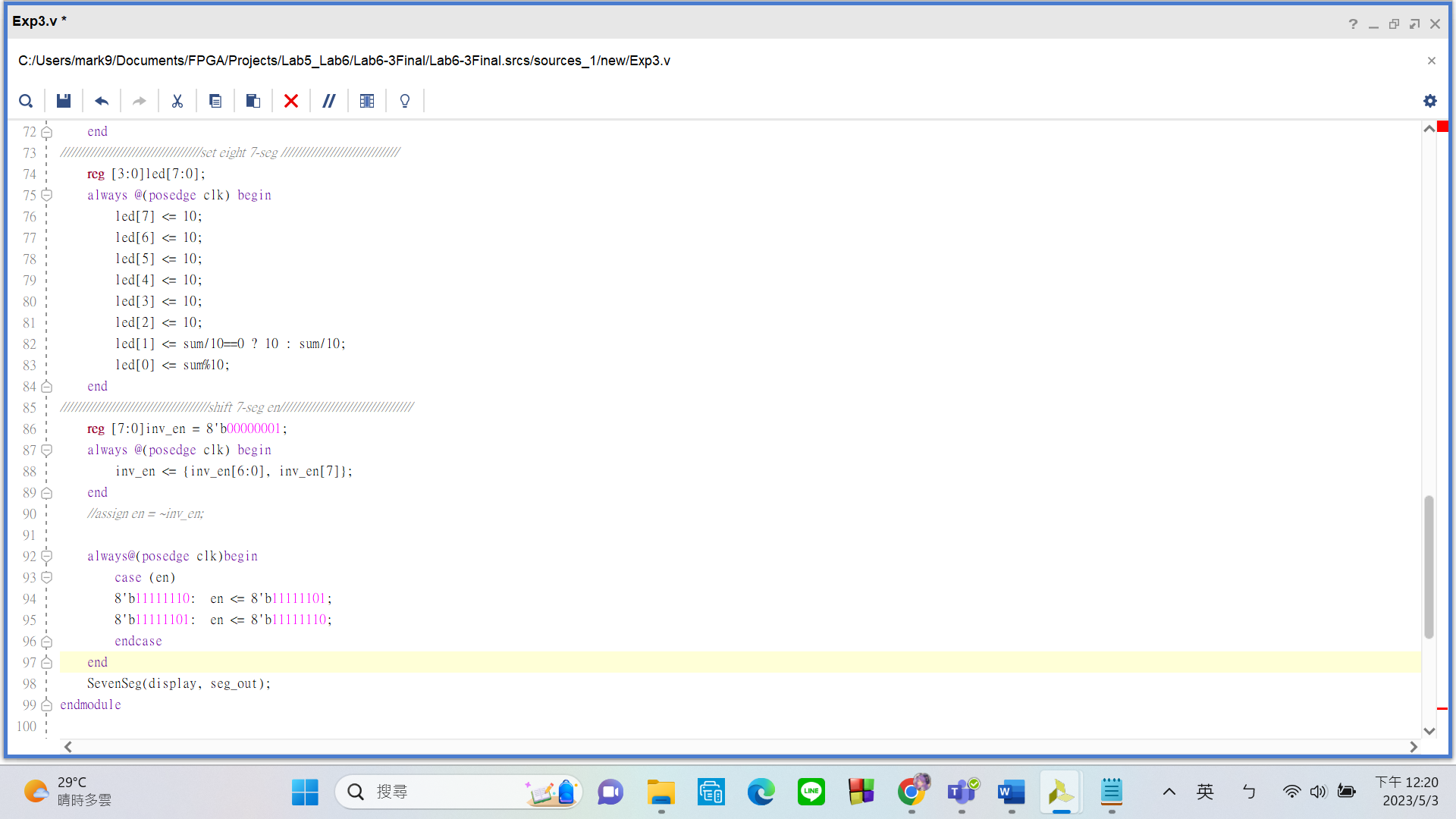
Remember what I said about there being a better method in 6-2. Here it is!

Please look at the case statement inside this always block. I used these curly brackets to **group together the rows register and columns register of the keypad{EnableKeypad(ROWS), keyin(COLUMNS)}**. Please refer to the following picture for

clarification. Once I group the regs together, I can use a **case statement** to determine which key was pressed. **For example, “1” would be 7’b1000100, “2” would be 7’b1000010, “4” would be 7’b0100100 etc.**

1. **Get Sum and Ouput**

This always block adds the addend and augend together and stores result in a 5-bit-reg called “sum”.



This always block quickly toggles between enabling the unit digit seven seg and the tens digit seven seg. Allowing us to give them different numbers.

The SevenSeg submodule works as previously explained, it translates numbers to what is shown on the displays.