Lab 8: The Keyboard (Calculator)

1 Implement Keyboard

1.1 Press 0/1/2/3/4/5/6/7/8/9 and show them in the seven-segment display. When a new number is pressed, the previous number is refreshed and over written.

1.2 Press a/s/m (addition/subtraction/multiplication) and show them in the seven segment display as your own defined A/S/M pattern. When you press “Enter”, refresh (turn off) the seven-segment display.

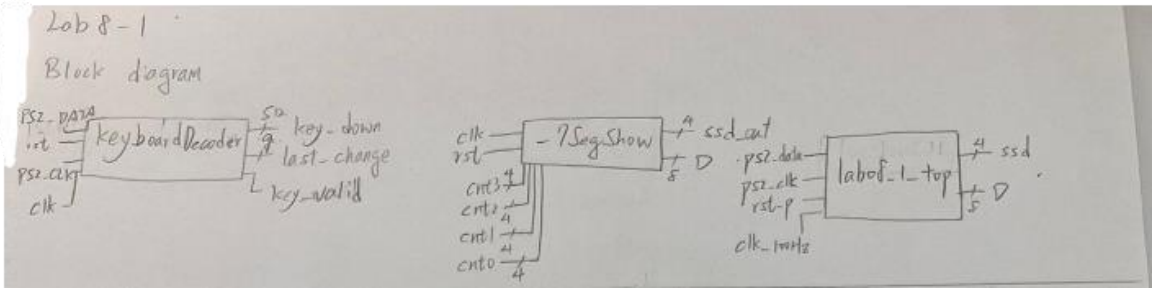
Design Specification

IO 輸出入設定

輸入: ps2_data(1 bit), ps2_clk(1 bit), rst_p(1 bit), clk_100Hz(1 bit)

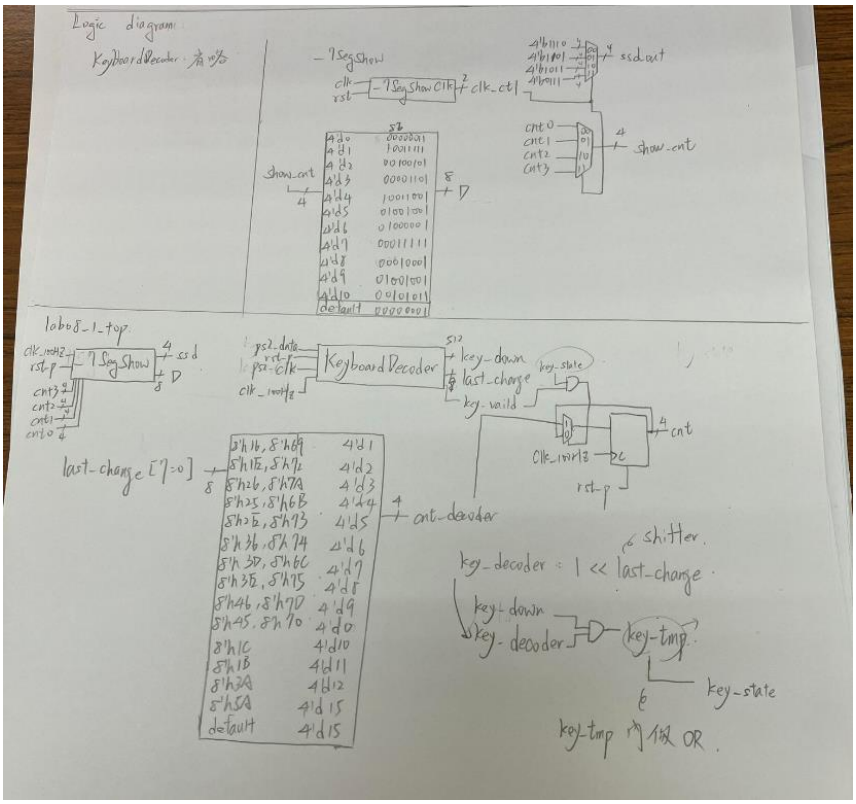
輸出: ssd(4 bits), D(8 bits)

Block diagram



Design Implementation

Logic diagram



如右圖，KeyboardDecoder 是直接使用老師提供的檔案做。而七段顯示器的顯示處理也是直接挪用之前 lab 做過的程式碼拿來處理。唯一我這次要解決的問題便是判斷 keyboard 按鍵。(勘誤:

1. 右圖在 lab08_1_top 裡，_7SegShow 的 cnt3, cnt2, cnt1, cnt0 應接到 cnt
2. _7SegShow 裡的 show_cnt (4 bits) to D(8 bits)的 Decoder 應多加:
   4’d10 → 8’b00010001
   4’d11 → 8’b01001001

4’d12 → 8’b00101011，分別代表 A, S, M)

由於這題的要求是要在按下一個數字鍵時，七段顯示器要顯示該段數字。但 last_change 和 key_valid 不只偵測按下的動作，鬆開按鍵時也會偵測一次。因此假設一個情境，若我按下按鍵 5，但不鬆手。此時七段顯示器顯示的數字是 5。而接著按下 6 時，七段顯示器刷新數字變成 6。但接著我鬆開了數字 5 的按鍵，此時理論上七段顯示器應該顯示 6 才對。然而，若使用 last_change 和 key_valid 判斷的話，七段顯示器的數字便會刷新成 5。因此只偵測按下去的瞬間是我要解決的問題。

而在我查看老師提供的程式碼後，知曉了 key_down 是如何運作的以及 index 的形式。但我接下來遇到的問題便是我不確定 reg 可以拿來放在[]中括號裡作為訪問該位元的 index。因此我逆向操作，既然 key_down 的生成是利用 shifter 的概念，那我也用 shifter 和 bitwise 處理。我先依據 make code 生成相對應的 key decode。這個 key decoder 和 key_down 類似，但只有 last_change 那一位元是 1，其餘皆是 0，這麼做是為了等下要單獨訪問這一位元。接著和 key_down 做 AND 後，我們可以確定整串位元都是 0，除了 last_change 位元。last_change 那一格的位元取決於 key_down 那一格的狀態。只要把這個運算結果整個作 OR 操作(查看有沒有 1)，便知道 last_change 的動作是按下按鍵還是鬆開按鍵了。這部分的操作我畫在 Logic diagram 的右下方。

```
"""
always @* begin
// https://elec  Follow link (ctrl + click)  .com/questions/447795/how-to-specify-a-value-for-each-bit-of-the-reg-in-v
// https://nandland.com/reduction-operators/
    key_decoder = 1 << last_change;      // 我們要訪問的index
    key_tmp = (key_down & key_decoder); // 和原本的key_down做AND，使得其他位元是0，保留我們要訪問的位元
    key_state = |key_tmp;               // 將每個位元做OR。由於其餘都是0，因此最後結果取決於要訪問的位元
                                            -->由此可得知是按下還是放開的狀態
end """
```

程式碼裡有附上我參考的連結。

最後，若我們判斷到當前的動作是按下按鍵時，cnt 便會讀去是按下哪一個鍵，這部分我用 case 去做處理。藉由判斷 last_change 的 make code，來決定要顯示哪一個數字或字母。

Pin assignment

IO	clk_100Hz	ps2_clk	ps2_data	rst_p	ssd[3]	ssd[2]	ssd[1]	ssd[0]
Pin	W5	C17	B17	R2	W4	V4	U4	U2
IO	D7	D6	D5	D4	D3	D2	D1	D0
Pin	W7	W6	U8	V8	U5	V5	U7	V7

Discussion

結果如我預期所想，不僅成功地達成題目要求，也避免了我所設想的錯誤。至於如過按下預期外的按鍵(非數字鍵和 A, S, M, Enter 鍵)，那七段顯示器的處理會和按下 enter 鍵一樣熄滅(但右下的小數點會亮，以顯示仍在運作)。除此之外，由於題目沒規定是哪裡的數字鍵，因此上排和右側的數字鍵我皆寫入 case 裡，使得我的 case 有些冗長。

Conclusion

在這一小節裡，我學到了

- 如何偵測 key board 按鍵
- Bitwise 操作

第一次接觸 key board 的操作體驗感覺不錯，有了 key board 可以讓我的程式有更多操作去呈現，我認為某種程度上也大大地感善使用者體驗，把它加進 final project 是不錯的選擇。

另外就是這次的 bitwise 操作，再看到老師提供的程式碼後讓我大為震驚。雖然高中在學競程時就有看過一些人做這樣的操作，可以大大地減少記憶體空間，另一方面也提升了時間複雜度。沒想到可以應用在這裡，讓我備感驚奇。

References

https://electronics.stackexchange.com/questions/447795/how-to-specify-a-value-for-each-bit-of-the-reg-in-verilog

https://nandland.com/reduction-operators/

第一個連結讓我更了解這種 bitwise 的操作，而第二個連結讓我得以處裡一個 reg 每個位元間 OR 的運算，讓我不必去寫一個 for 迴圈來處理。

2 Implement a single digit decimal adder using the keyboard as the input and display the results on the 7-segment display (The first two digit are the addend/augend, and the last two digits are the sum).

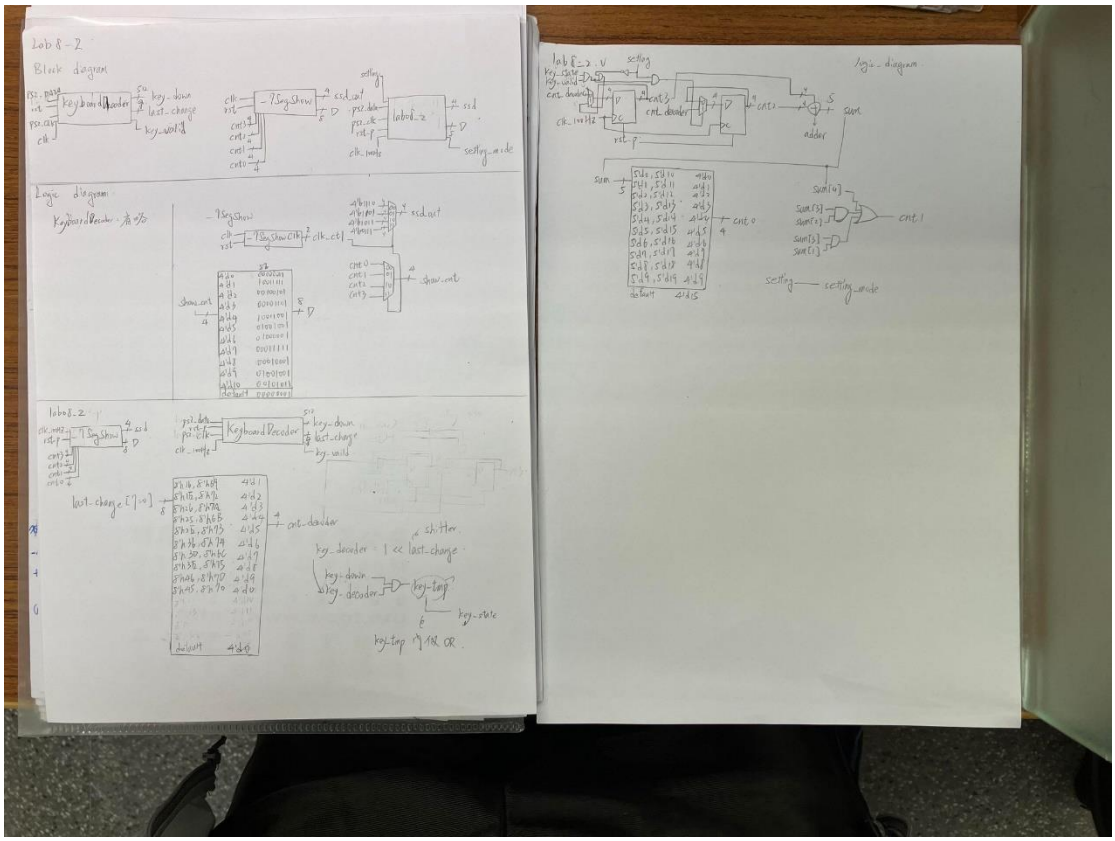
Design Specification

IO 輸出入設定
輸入: ps2_data(1 bit), ps2_clk(1 bit), rst_p(1 bit), setting(1 bit), clk_100Hz(1 bit)
輸出: setting_mode(1 bit), ssd(4 bits), D(8 bits)

Block diagram(下圖，和 Logic diagram 一起)

Design Implementation

Logic diagram



基本上挪用前面 lab8-1 的內容，少部分的改動如下說明。
首先是顯示的部分，七段顯示器的設定如往常一樣，唯一要變的是輸出的數字，因此在 top module — lab08_2.v 裡處理。
先談輸入的數字，用撥桿(setting V17)決定現在要輸入的是被加數還是加數。預設是先輸入最左邊的數字(被加數)。而在輸入的過程中就會自動作加法的動作。

這裡我就沒有做加法器了，直接用+符號進行運算。算完後先用一個 5bits 的 reg(sum)儲存。接著再用 case 來把它轉成 BCD 的個位數。而十位數只要偵測幾個特定的 case 就好，用簡單的組合邏輯即可。

輸入的過程判定和 lab8-1 一樣，但多了 setting 的條件判斷。若按鍵判斷是按下動作且 setting 為 0，就是設定被加數，反之則是設定加數。而 setting_mode(U16)會顯示當前處於哪個狀態。

Pin assignment

IO	clk_100Hz	ps2_clk	ps2_data	rst_p	ssd[3]	ssd[2]	ssd[1]	ssd[0]
Pin	W5	C17	B17	R2	W4	V4	U4	U2
IO	D7	D6	D5	D4	D3	D2	D1	D0
Pin	W7	W6	U8	V8	U5	V5	U7	V7
IO	setting_mode	setting						
Pin	U16	V17						

Discussion

基本上此題的操作只是延續上一題，基本的按鍵判斷寫好後，後續只是一些簡單的時序邏輯的變化而已。大致上的架構也是直接使用第一題所用到的程式碼，把一些細節修改或註解掉而已。

Conclusion

在本小節裡，我學到了

- Key board 的實際運用

再了解如何偵測鍵盤按鍵後，輕鬆地寫完一個簡單的小程式。過程上大概只花了十幾分鐘解決，也讓我對鍵盤控制愈來愈熟練。

3 Implement a two-digit decimal adder/subtractor/multiplier using the right-hand-side keyboard (inside the red block). You don’t need to show all inputs and outputs at the same time in the 7-segment display. You just need to show inputs when they are pressed and show the results after “Enter” is pressed.

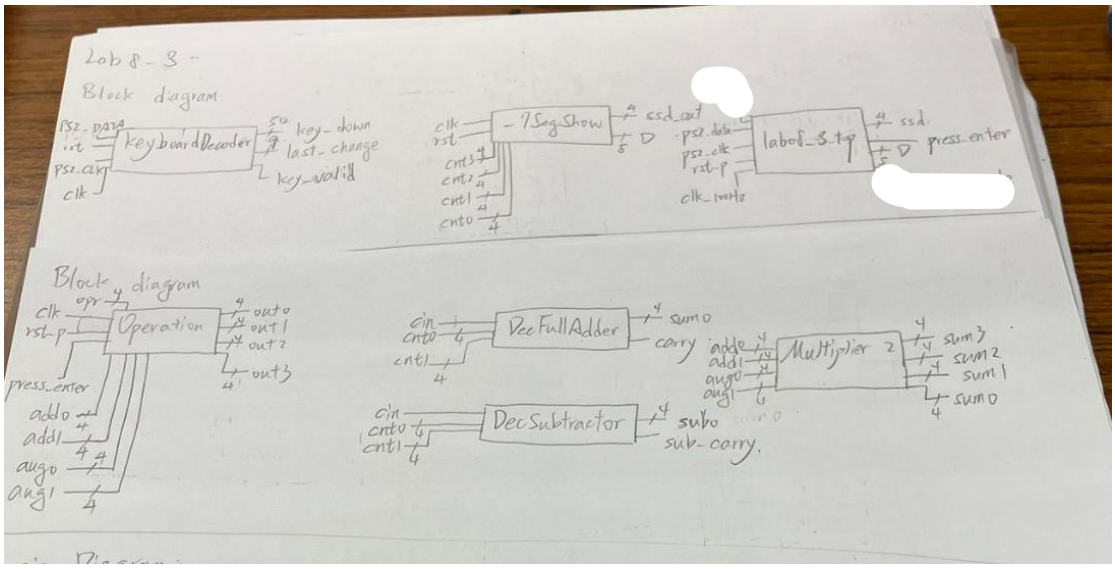
Design Specification

IO 輸出入設定

輸入: ps2_data(1 bit), ps2_clk(1 bit), rst_p(1 bit), setting(1 bit), clk_100Hz(1 bit)

輸出: ssd(4 bits), D(8 bits), press_enter(1 bit)

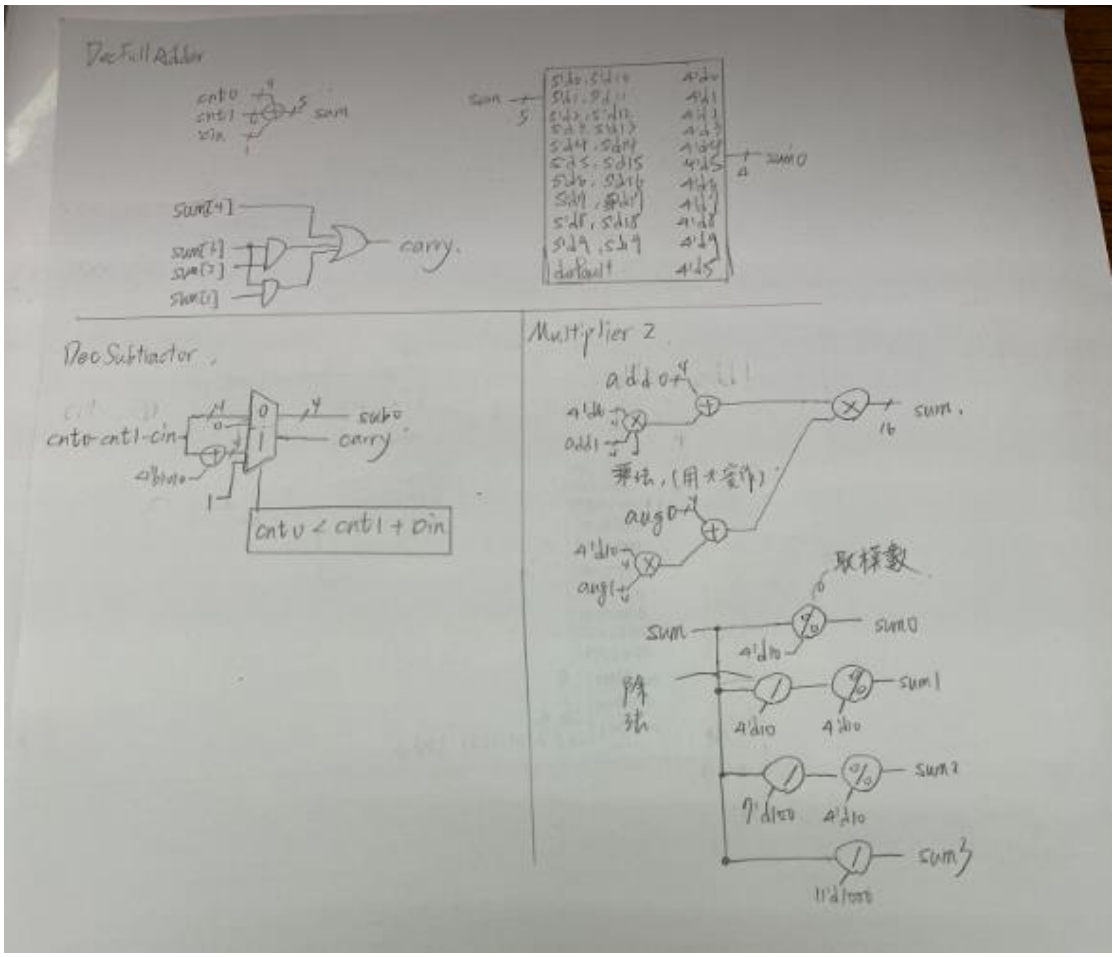
Block diagram



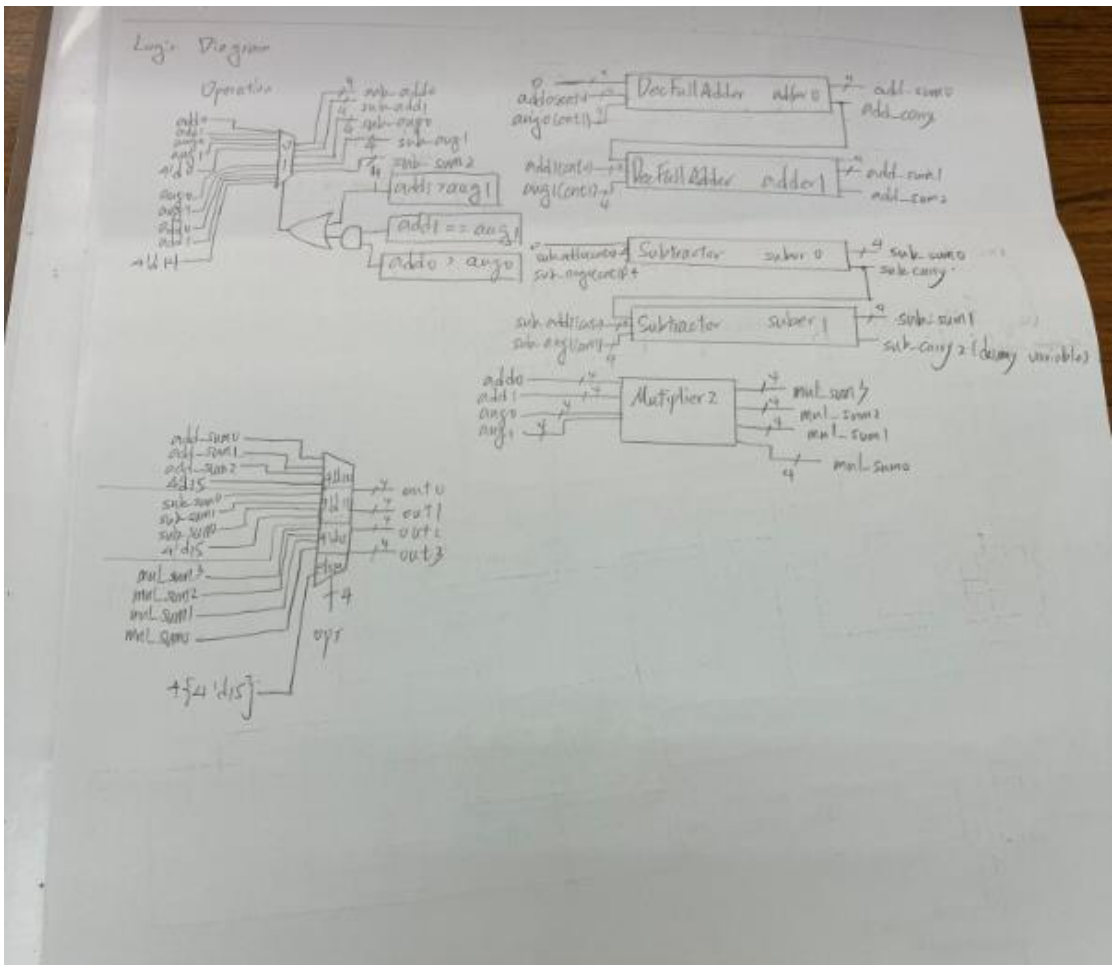
Design Implementation

Logic diagram

Some adder, subtractor, multiplier



Operation.v



由於這部分有點複雜，我為了嘗試說明每個模組的功能，因此先說明這部分的模組。

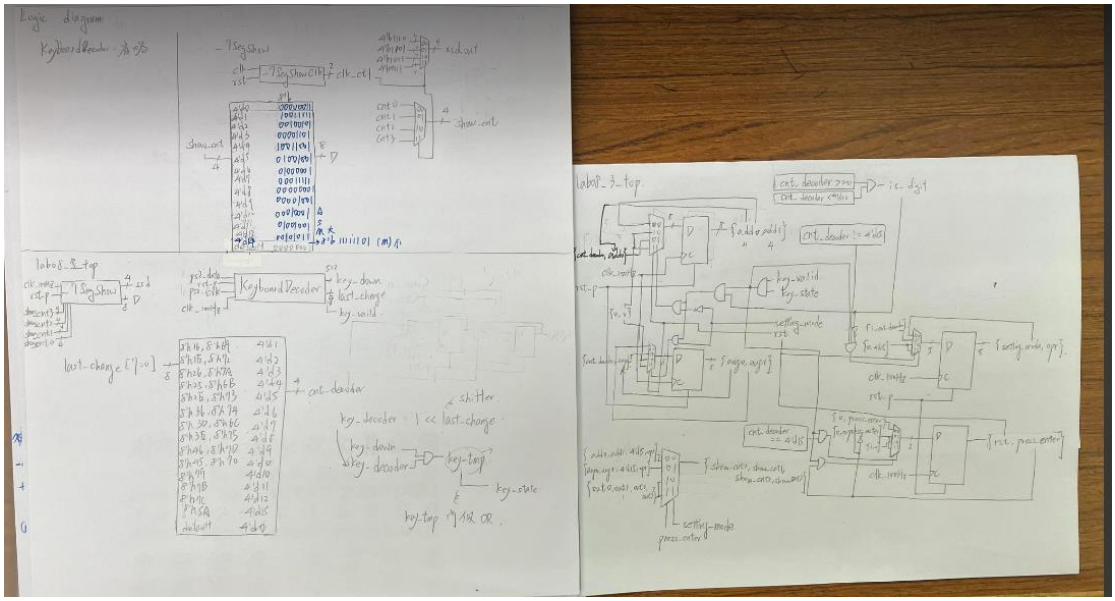
首先先看 top module 裡的 Operation.v，這個模組是用於處裡數學計算的模組。裡面大致上又塞了三種模組—加法器、減法器、乘法器。乘法器非常地直觀，我就直接用 verilog 支援的數學運算子來處理，這部分寫法如同 C 一樣。不過這並不是我一開始的寫法，原始寫法(程式碼中的註解)就留到 Discussion 再討論。

接著看加法器，加法器我寫成一位 BCD 的加法器，所以在 Operation 裡放了兩個 DecFullAdder。處理完後的結果會自動再轉成 BCD 格式。
最後看減法，處理減法時，先在 Operation 裡判斷被減數和減數的大小，以決定第三位 sub_sum2 是否要顯示負號。接著再丟入減法器裡，利用 verilog 支援的減法運算處裡，這裡方便的是不用處理 BCD 轉換的問題，因為一定會小於 4’d10。

回到 Operation，最後我們再利用判斷 opr(使用者輸入的運算符)代表當前要顯示哪個運算的結果。因此把所有運算程式都寫完之後，我們只要把數字和運算子

丟進去就可以了。非常直觀，對吧! (注: 在程式裡，add 代表被加入/被減數/被乘數，而 aug 代表加數/減數/乘數)

接著是 Top module, _7SegShow.v(基本上同前面)
KeyboardDecoder 和_7SegShow 功能和程式碼都跟前面一樣，就不贅述。



Top module 裡涉及到多個狀態，我接著慢慢說明。
偵測按鍵的功能如同前面，因此我們只要處理一些 FSM 狀態的問題。
第一個問題是: 當使用者輸入第一個數字時，如何做到像計算機那樣直觀的輸入，例如: 當我要輸入 43，是先輸入 4 再輸入 3。
解決方法是預設皆設為 0，當我輸入數字時，是輸入到個位數，而原本的個位數位移到十位數。所以若輸錯數字或想重新輸入數字，就直接輸入你想輸入的二位數就好，因為程式自動會覆蓋掉原本的數字(倘若只輸入個位數的話要重新輸入 0，例如: 只要個位數 3，便輸入 03)。

接著是輸入運算子和輸入加數/減數/乘數時的狀態切換，這部分由 setting_mode 變數控制，若其為 0 代表輸入被加數，1 則是加數。若完成輸入被加數後準備要輸入加數時，就只要先輸入運算子(opr)。此時運算子會顯示在最左邊的七段顯示器上(A: 加法; S: 減法; M: 乘法)，之後可以再改運算子(在按下 enter 之前)。再來就可以輸入加數了，加數的輸入法和被加數一樣，同樣操作即可。
程式裡的操作是在偵測輸入 opr(非數字按鍵且非 enter 鍵，但我沒有做按下非預期輸入的處理如:字母)後，setting_mode 會切換成 1(且永遠是 1 直到我們觸發 reset 機制)。

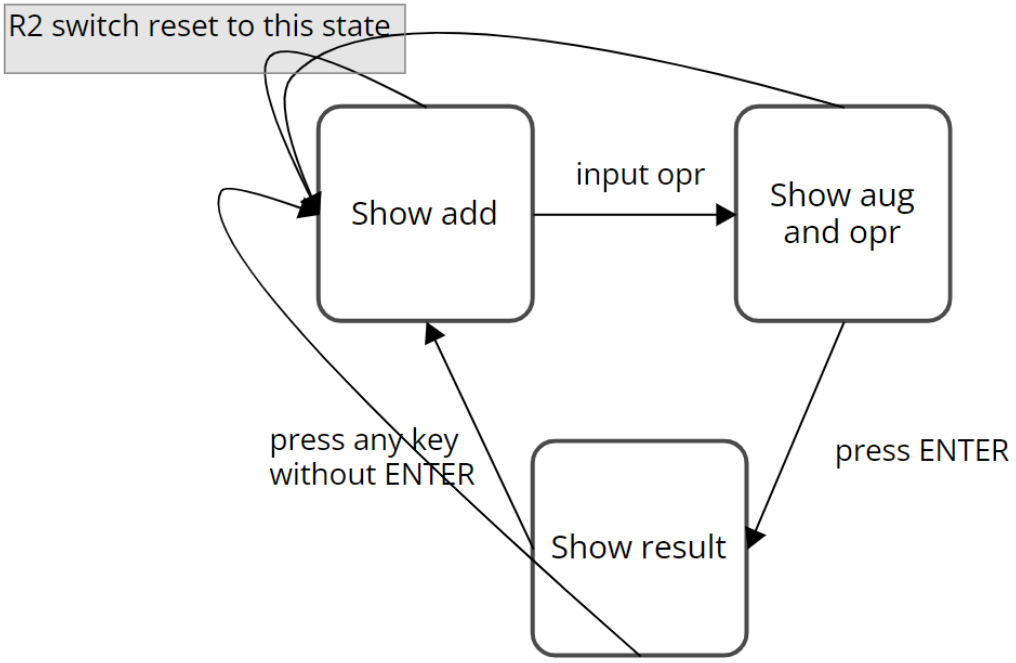
最後按下 enter 後，press_enter 會切換成 1(原本是寫成~press_enter，但若再按

一次 enter 會回到前面狀態，經助教提點後修正(原本寫法呈現在 **Logic diagram** 裡))，代表已按下 enter 鍵進入顯示結果狀態。接著就會顯示計算後的結果，這裡有一個小 bug 是由於我的減法器有計算判斷負號的問題，但若是 0 - 0 會出現”-00”的情況。不過其餘只要出現小減大的問題都能正確出現負號。

最後，再次按下 enter 還是會處於顯示結果的狀態。若要再次計算就要觸發 reset 條件，有兩個方法: 一是撥動 R2 switch 撥桿，觸發整個系統的重設，二是按下任意鍵(除了 enter 鍵)，也會做 reset 的動作，但此時的動作就只會做 reset 而已，所以如果按下數字鍵作為重設，就只會重設而已，還要再重新輸入被加數。

若在未輸入運算子 opr 的狀態下按下 enter，七段顯示器會顯示小數點而已，代表 error。

State diagram(注: 應該是 except ENTER 而不是 without ENTER(文法錯誤))



Pin assignment

IO	clk_100Hz	ps2_clk	ps2_data	rst_p	ssd[3]	ssd[2]	ssd[1]	ssd[0]
Pin	W5	C17	B17	R2	W4	V4	U4	U2
IO	D7	D6	D5	D4	D3	D2	D1	D0
Pin	W7	W6	U8	V8	U5	V5	U7	V7

Discussion

程式基本上都能完成題目的指示，這裡就來談談我原本寫的 Multiplier。我把它保留在 project 裡，不過我的 top module 是接到 Multiplier2(後來的寫法)。原本我沒有想到 verilog 的除法和取餘數的操作，因此我的方法是先把輸入的 BCD 碼轉成 binary code。再用 verilog 支援的乘法運算後，想辦法處裡 binary 轉 BCD 的問題。但這裡遇到一個困難點是我不知道怎麼將多 bits 的 binary code 轉成 BCD 碼。經過我上網查詢後發現了一個叫 Double Dabble 方法，能夠將多位元的二進位數轉成 BCD。但後續又遇到一個問題，就是它會用到 shifter，所以要嘛要寫一個複雜的迴圈執行，或是用正反器去執行。但若用正反器執行的話，如果數字是固定的話那還好，但要是處理到一半時使用者更改數字，那還要處裡判斷終止位移並重新轉換。所以在我得知可以使用除法和模數(取餘數)的操作後，便放棄這樣的寫法，改為後來的寫法。

其餘的的模組就還蠻簡單的，過程上沒遇到什麼困難。

Conclusion

在這一小節裡，我學到了

- 乘法器的操作
- 如何處理多位元的 bits 轉成 BCD

過程中花費了許多時間處裡轉成 BCD 的問題，最後卻用了更簡單的方式完成。雖然感覺有點浪費時間，不過學到這個方法也算讓我獲益良多，搞不好將來哪一天會用到也說不定。

References

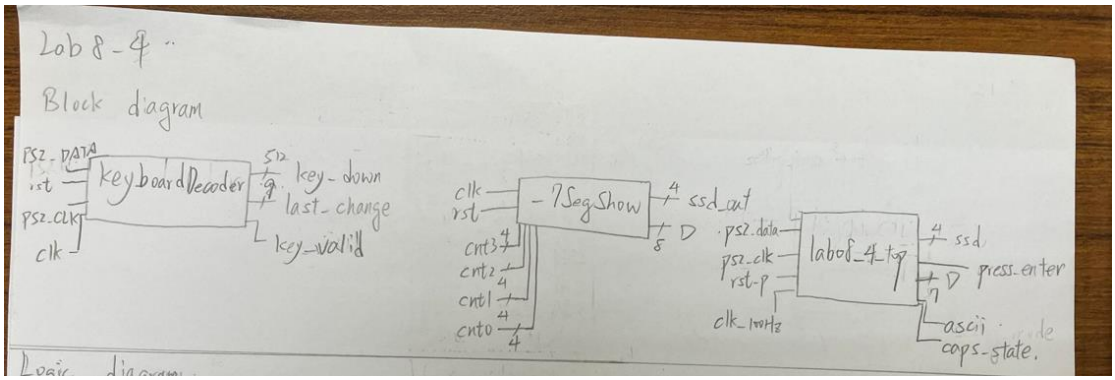
https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Unsigned-16x16-Bit-Multiplier-Coding-VHDL-Example
關於 verilog 可以直接支援乘法的操作
https://nandland.com/binary-to-bcd-the-double-dabbler/
https://en.wikipedia.org/wiki/Double_dabble
上面兩個是關於 Double Dabble 的方法

4 Implement the “Caps” control in the keyboard. When you press A-Z and a-z in the keyboard, the ASCII code of the pressed key (letter) is shown on 7-bit LEDs.
4.1 Press “Caps Lock” key to change the status of capital/lower case on the keyboard. Use a led to indicate the status of capital/lowercase in the keyboard and show the ASSCII code of the pressed key one 7-bit LEDS.
4.2 Implement the combinational keys. When you press “Shift” and the letter keys at the same time. The 7-bit LEDs will show the ASCII code of the uppercase/lowercase of the pressed letter when the “Caps Lock” is at the lowercase/uppercase status.

Design Specification

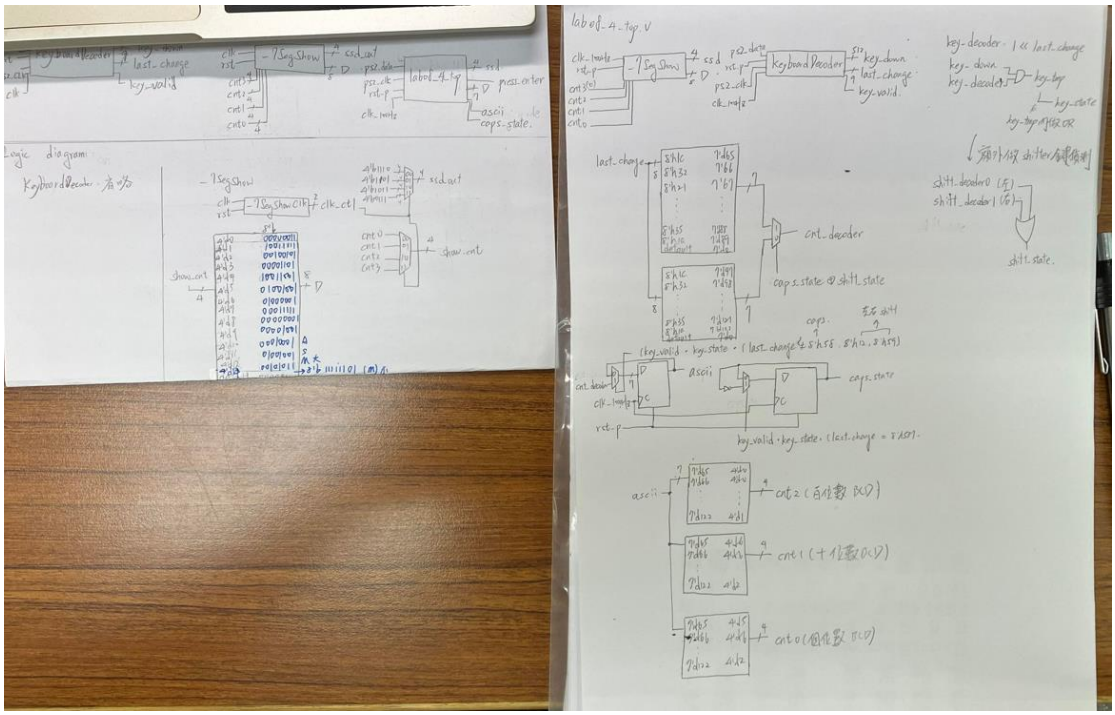
IO 輸出入設定
輸入: ps2_data(1 bit), ps2_clk(1 bit), rst_p(1 bit), (1 bit), clk_100Hz(1 bit)
輸出: ssd(4 bits), D(8 bits), ascii (7 bits), caps_state(1 bit)

Block diagram



Design Implementation

Logic diagram



這題架構相較簡單，只需要處理許多 case 就可以了。七段顯示器和偵測按下按鍵的程式碼皆一樣。只是這次多了 shift_state 來查看 shift 是否處於被按下的狀態，因此它的偵測我另外寫，原理還是一樣。且由於有兩個 shift 按鍵，所以要寫兩個 decoder(兩個 make code 不一樣)

然後多寫一個 caps_state 來模擬鍵盤上的大小寫指示燈，並偵測當 caps 鍵按下時便切換狀態。剩下了字母判斷就非常簡單了，只需要看當前應輸出大寫還是小寫，再用 decoder，如同數字鍵處理方式即可。而大小寫的時機就利用 shift_state 和 caps_state 的 XOR 結果判斷，非常簡單。

Pin assignment

IO	clk_100Hz	ps2_clk	ps2_data	rst_p	ssd[3]	ssd[2]	ssd[1]	ssd[0]
Pin	W5	C17	B17	R2	W4	V4	U4	U2
IO	D7	D6	D5	D4	D3	D2	D1	D0
Pin	W7	W6	U8	V8	U5	V5	U7	V7
IO	caps_state	ascii[6]	ascii[5]	ascii[4]	ascii[3]	ascii[2]	ascii[1]	ascii[0]
Pin	U16	L1	P1	N3	P3	U3	W3	V3

Discussion

程式一樣可以達成題目的所有要求，若按下預期外的話會歸零。至於為何我要多寫七段顯示器呢? 是因為我一開始看錯題目的要求了。不過有了七段顯示器，在使用上便平易近人許多，不需要花許多時間看 binary code!

Conclusion

在本小節裡，我學到了

- 複合按鍵的寫法

由於一開始 lab1 在寫偵測按鈕時便使用的了 key_down，因此到了 lab4 使用 key_down 操作對我來說輕鬆許多。一下子就完成該小題了。