Logic Design 106061212 賴傳堯

Lab09

(1) Keyboard – display 0~9、A、S、M when pressed

I/O:

Inout: PS2_DATA

Inout: PS2_CLK

Inout: rst // high active reset

Inout: clk // 100MHz

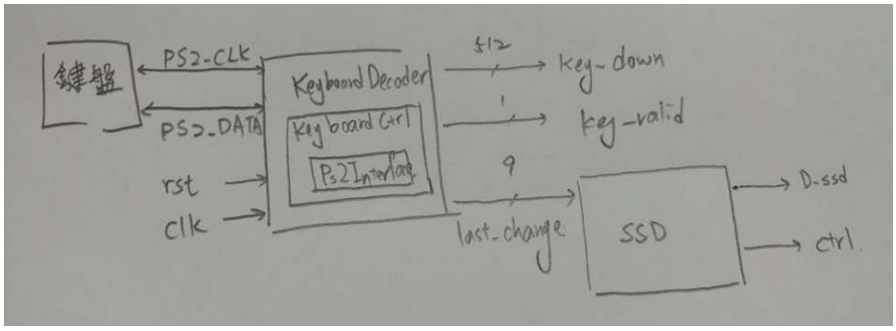
Output: [3:0]ctrl

Output: [7:0]D_ssd

Pin:

W5 - clk	W7 - D_ssd[7]
U18- rst	W6 - D_ssd[6]
B17 - PS2_DATA	U8 - D_ssd[5]
C17 - PS2_CLK	V8 - D_ssd[4]
W4 - ctrl[3]	U5 - D_ssd[3]
V4 - ctrl[2]	V5 - D_ssd[2]
U4 - ctrl[1]	U7 - D_ssd[1]
U2 - ctrl[0]	V7 - D_ssd[0]

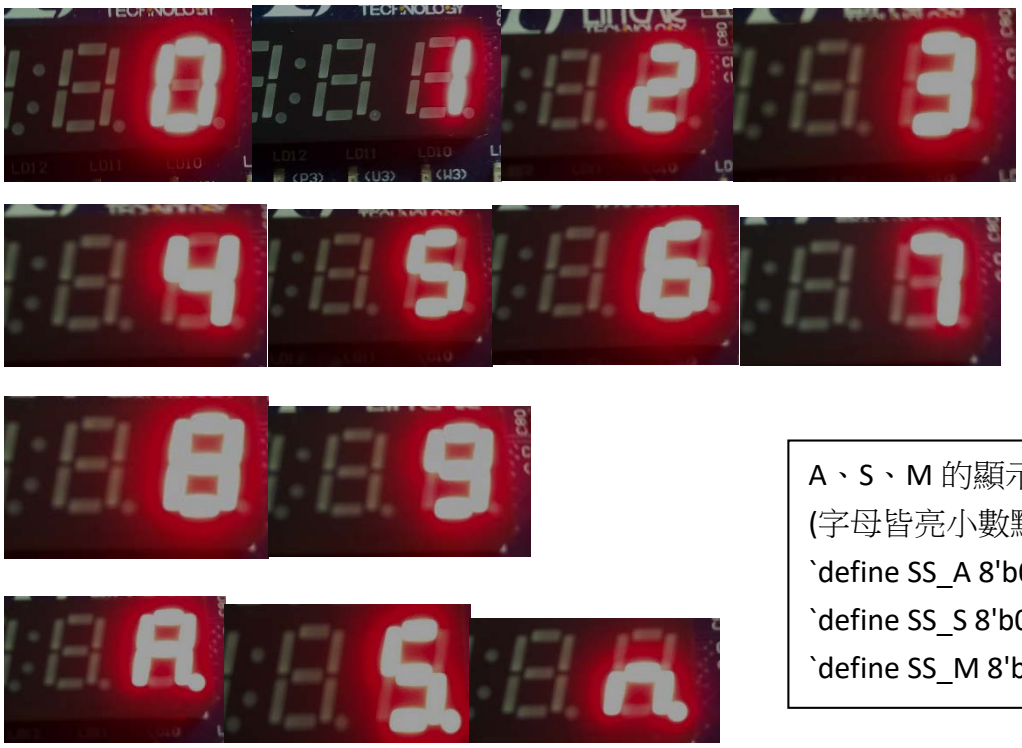
Block diagram:



Discussion:

題解&思路&作法解釋:

這題只要在按下按鍵後顯示對應的數字和字母，並每次取代掉前一個即可。我的作法是直接用 case 判斷 last_change 為何，接著 assign 相對應的值給 D_ssd。而因為只需要顯示一位數，所以讓 ctrl 永遠是 1110，只有在按下 Enter 時(last_change[7:0] == 8'h5A)，ctrl 會變 1111，所有燈暗掉(refresh)。



A、S、M 的顯示定:
(字母皆亮小數點)
`define SS_A 8'b00010000
`define SS_S 8'b01001000
`define SS_M 8'b11010100

Conclusion:

第一題很簡單，只用到了 last_change，因此沒什麼問題。加上每次只要覆蓋掉前一次的顯示字母或數字，又不用實際有運算功能，基本上就是簡單的應用以前學會的就能完成。

(2) Keyboard – Single Digit Decimal Adder

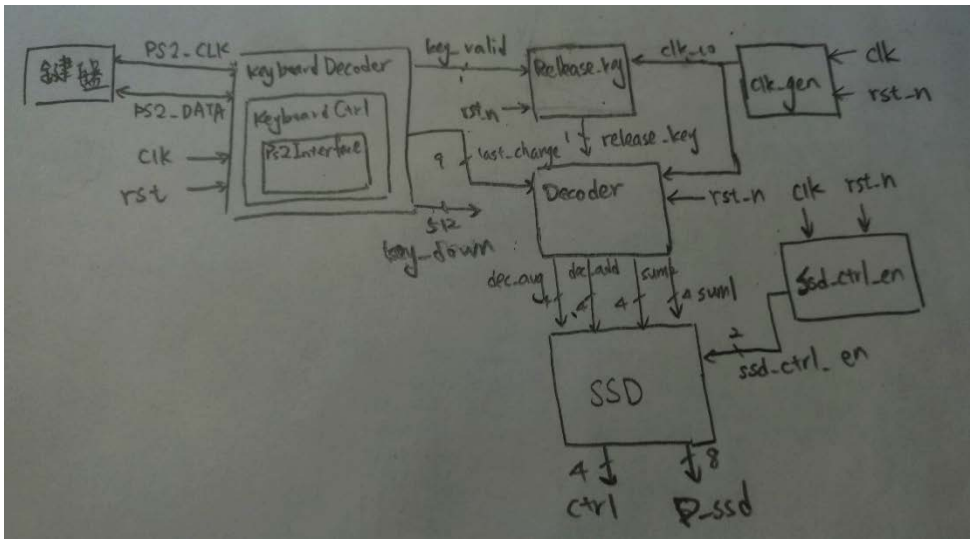
I/O:

Inout: PS2_DATA
Inout: PS2_CLK
Inout: rst // high active reset
Inout: clk // 100MHz
Input: rst_n
Output: [3:0]ctrl
Output: [7:0]D_ssd

Pin:

W5 - clk	W7 - D_ssd[7]
U18- rst	W6 - D_ssd[6]
B17 - PS2_DATA	U8 - D_ssd[5]
C17 - PS2_CLK	V8 - D_ssd[4]
W4 - ctrl[3]	U5 - D_ssd[3]
V4 - ctrl[2]	V5 - D_ssd[2]
U4 - ctrl[1]	U7 - D_ssd[1]
U2 - ctrl[0]	V7 - D_ssd[0]
V17 - rst_n	

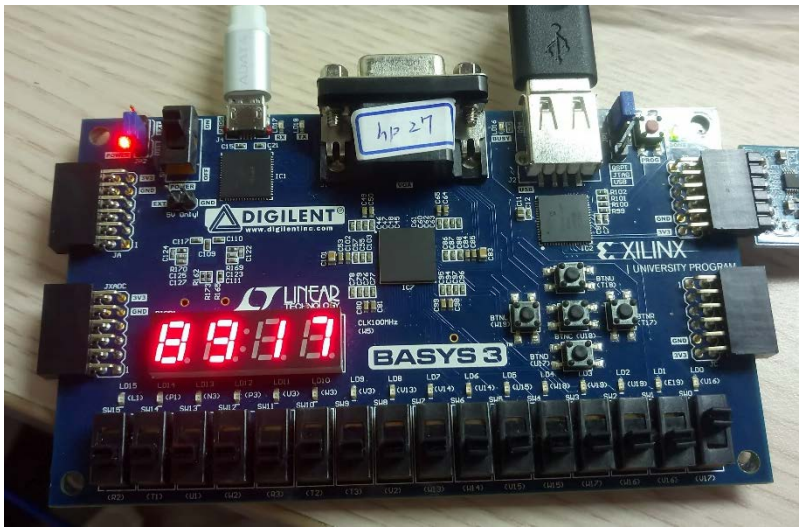
Block diagram:



Discussion:

題解&思路&作法解釋:

這題要做個位數加法。首先先用 fsm，透過 key_valid 在每次按下、放開按鍵時都會送出一個 pulse 的性質，得到按下按鍵與放開按鍵兩個 state。將這個訊號傳給 Decoder(重新名為 release_key)，在 release_key 的 negedge 判斷按下的鍵與對應的數字，同時作另一個 fsm 讓輸入的數字輪流存給加數(addend)、被加數(augend)，之後再用 case 作判斷，轉換為對應的十進位數字(dec_add、dec_aug)。由於題目沒有要求要按 enter，所以這裡設計成在按下數字的同時就會直接顯示目前的和。和的算法是兩個數字相加後得到 sum，十位數(sum2)=sum/10，個位數(sum2)=%10。



執行結果:
8+9=17

實驗困難與問題:

這題比第一題複雜在於有兩個分開的數字要記，所以必須要有 fsm 來記錄現在輸入的訊號要存給哪個變數，要做出這個 fsm，就必須判斷何時有「按下按鍵」的動作，於是就得用到 key_valid，而 key_valid 又是不管按下或放開都輸

出 1，所以還需要另一個 fsm(或是 0、1 的 counter)來記錄現在到底是按下還是放開。最後會得到按下時為 1，沒按時為 0 的訊號。而這麼做實際上有個缺點，就是當鍵盤感應不良的狀況下，key_valid 可能只讀到按下或放開其中一個動作，而只輸出一次 1 的 pulse，這麼一來 fsm 的結果就會完全相反，因此在案按鍵時必須注意，一定要按好案滿之後再放開。在不了解鍵盤與 key_valid 之間訊號是如何處理的情況下，我想可能很難避免這個缺陷。

Conclusion:

第二題承接了第一題的輸入數字，並為第三題更完整的計算功能鋪路，相較於第一題其實沒有難很多，然而相較於第三題則是簡單許多!而要說在這題學到的新東西，就是比上題多用了 key_valid 這個變數。

(3) Keyboard – Single Digit Decimal Adder

I/O:

Inout: PS2_DATA

Inout: PS2_CLK

Inout: rst // high active reset

Inout: clk // 100MHz

Input: rst_n

Output: [3:0]ctrl

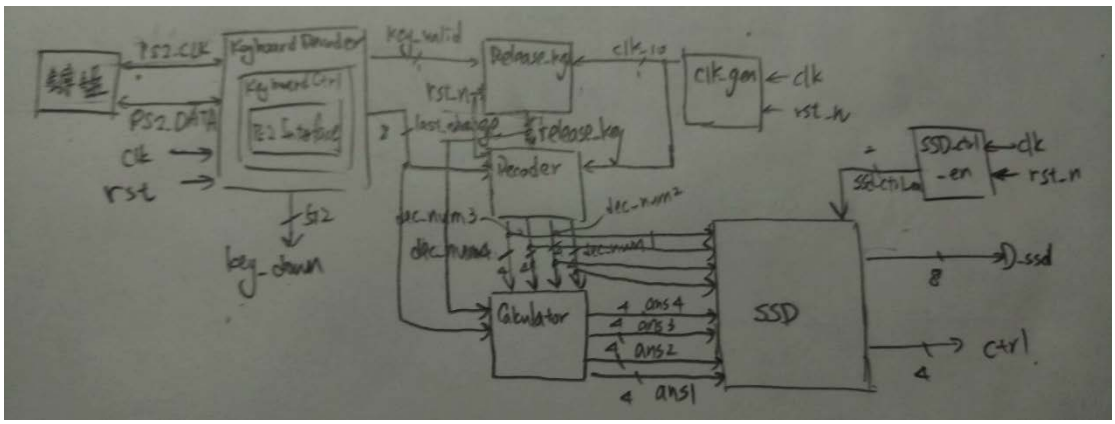
Output: [7:0]D_ssd

Pin:

W5 - clk	W7 - D_ssd[7]
U18- rst	W6 - D_ssd[6]
B17 - PS2_DATA	U8 - D_ssd[5]
C17 - PS2_CLK	V8 - D_ssd[4]
W4 - ctrl[3]	U5 - D_ssd[3]
V4 - ctrl[2]	V5 - D_ssd[2]
U4 - ctrl[1]	U7 - D_ssd[1]
U2 - ctrl[0]	V7 - D_ssd[0]

V17 - rst_n

Block Diagram:



Discussion:

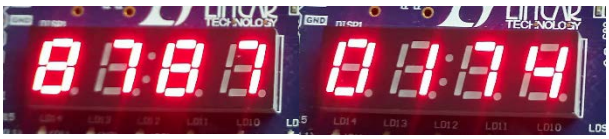
題解&思路&作法解釋:

這題要做二位數的加減乘，數字存取的方法如同上題，透過 fsm 紀錄現在存到哪個數字，並且只有在按下數字鍵的情況下，fsm 的 state 才會有改變(透過 case 判斷)。接著同樣用 case 判斷，將讀進的數值轉為對應的十進為數字，傳給 SSD 以及 Calculator。在 Calculator 中，會透過 case 判斷，如果 last_change 是 A、S、M 其中一個按鍵，會用 state 記住，並且 ans 就會是相對應的運算結果。而後透過運算將 ans 分為四位數(ans4、ans3、ans2、ans1)，同樣傳給 SSD。其中減法因題目沒規定，這裡設定為絕對值的結果，也就是經判斷之後，進行大減小。

```
ans4 = ans/1000;
ans3 = (ans%1000)/100;
ans2 = (ans%100)/10;
ans1 = ans%10;
```

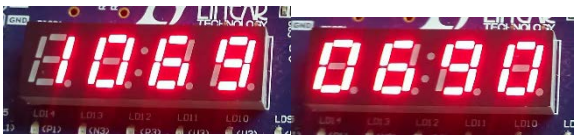
將 ans 轉為四位數的計算方法

在 SSD 中，透過 case 判斷 last_change 是否為 enter 鍵(8'h5A)，是則顯示答案(display4 = ans4，以此類推)，否則顯示運算的數字(display4 = num4，以此類推)。



執行結果-加法:
87+87=174

執行結果-減法:
66-60=6



執行結果-乘法:
10*69=690

實驗困難與問題:

這題說時在的只不過比的二題多了兩個位數、兩個運算子，也就是多一 fsm 要處理，並多一個 enter 的判斷，理論上不該太難。實際寫起來，確實也是比上一題多花了點腦筋就寫完了，但不論怎麼測試，就是有問題:按的一下沒反應，按的二下會顯示上一個按的數字。而且有時正確有時錯誤，有時按一按會回到正確的順序，有時不論怎麼按，顯示慢一拍就是慢一拍，搞的人一個頭兩個大。後來我把讀進數字的@negedge release_key 改成 posedge，這解決了大部分按的二下才出現第一個數字的問題，但偶爾仍會發生一樣的情形。後來經過數不清次數的測試後，發現其實問題很簡單，就是來字上一題討論過的:key_valid 的問題。平時不會大力按壓鍵盤，因此似乎城市常常讀不到按下或放開其中一個訊號，導致 release_key 的判斷亂掉，最終導致錯誤的結果。如果每一下都用力按到底在再放開，就沒問題了。

Conclusion:

要說這題學到的最大教訓，就是絕對不要相信鍵盤的靈敏度，並且用 release_key 判斷實在不是個好點子。不過，完成這題，表示自己已經可以寫出一個有模有樣的計算機了，比起剛開學的自己，總覺得真的進步了不少!

(4) Keyboard – Single Digit Decimal Adder

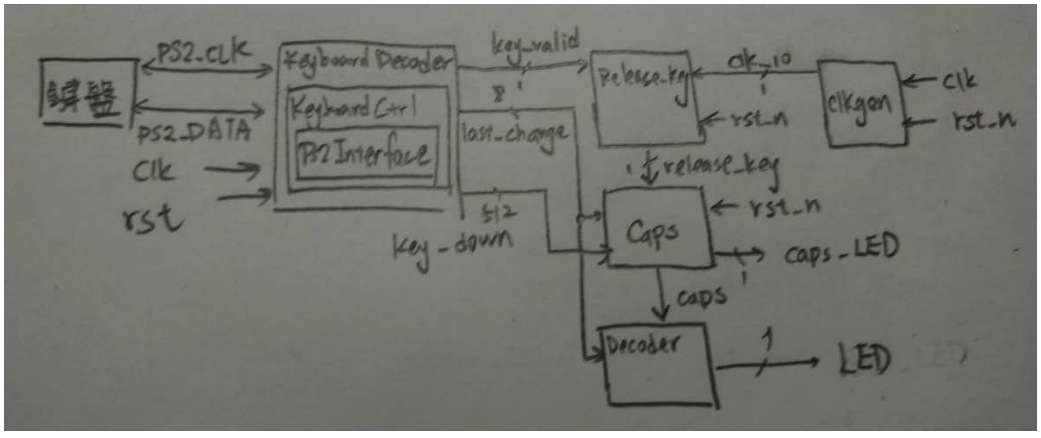
I/O:

Inout: PS2_DATA
Inout: PS2_CLK
Inout: rst // high active reset
Inout: clk // 100MHz
Input: rst_n
Output: [3:0]ctrl
Output: [7:0]D_ssd

Pin:

W5 - clk	U14 - LED[6]
U18- rst	U15 - LED[5]
B17 - PS2_DATA	W18 - LED[4]
C17 - PS2_CLK	V19 - LED[3]
V17 - rst_n	U19 - LED[2]
L1 - caps_LED	E19 - LED[1]
P1 - shift_LED	U16 - LED[0]

Block Diagram:



Discussion:

題解&思路&作法解釋:

這題要做英文字母大小寫切換，並以 7bit 二進位 LED 燈顯示。第一步判斷 Caps 是否開起，一樣用 fsm 記錄其 state(紀錄在變數 caps_on，固定大寫為 1，反之為 0)，再來判斷是否按下 shift，不同的是這裡是拿 key_down 來判斷，有按則 shift=1，反之為 0。最後，讓最終輸出 caps = (caps_on & ~shift) | (~caps_on & shift)。

caps_on	shift	caps
0	0	0
0	1	1
1	0	1
1	1	0

把 caps 傳給 Decoder，在裡面首先判斷 last_change 是哪個英文字母，以及現在的 caps 是大寫還是小寫，透過 case 將對應的值 assign 給 LED，就得到所要的結果。

小寫 a

按住 shift，左二燈亮，大寫 A

按下 Caps 後，最左燈亮，大寫 A

按了 Caps 且按住 shift，兩燈皆亮，小寫 a

實驗困難與問題:

實驗到這，第一次遇到組合鍵，才第一次使用到 key_down。看老師的講義也不明白 ket_down 究竟是甚麼，問過同學才知道，裡面是以 one-hot 的形式紀錄按鍵的位址，也就是對 shift 來說，key_down[8'h12]會等於 1。除了這部分在實驗過程中卡住之外，其他都比的三題要來的簡單，沒太大問題。

Conclusion:

這次實驗又是需要用到老師提供寫好的模組，同樣也是得了解這些模組輸出的訊號功能為何，基本上講義對於 key_valid 和 last_change 的描述都算清楚，讓人易懂，唯獨 key_down 的解釋讓人看了還是不大明白，才導致的四題過程停頓，而這再次說明了當以後在合作寫程式時，清楚溝通彼此的分工及功能的重要性。

Reference:老師講義 09，包含鍵盤處理模組，以及輸入、出功能說明。