Lab7: Electronic Clock II (Multi-functions)

107061112 王昊文

Experiment

1 Implement a stopwatch function with the FPGA board.

Design Specification

✓ I/O

Input clk; // crystal clock

Input rst_n; // low active reset

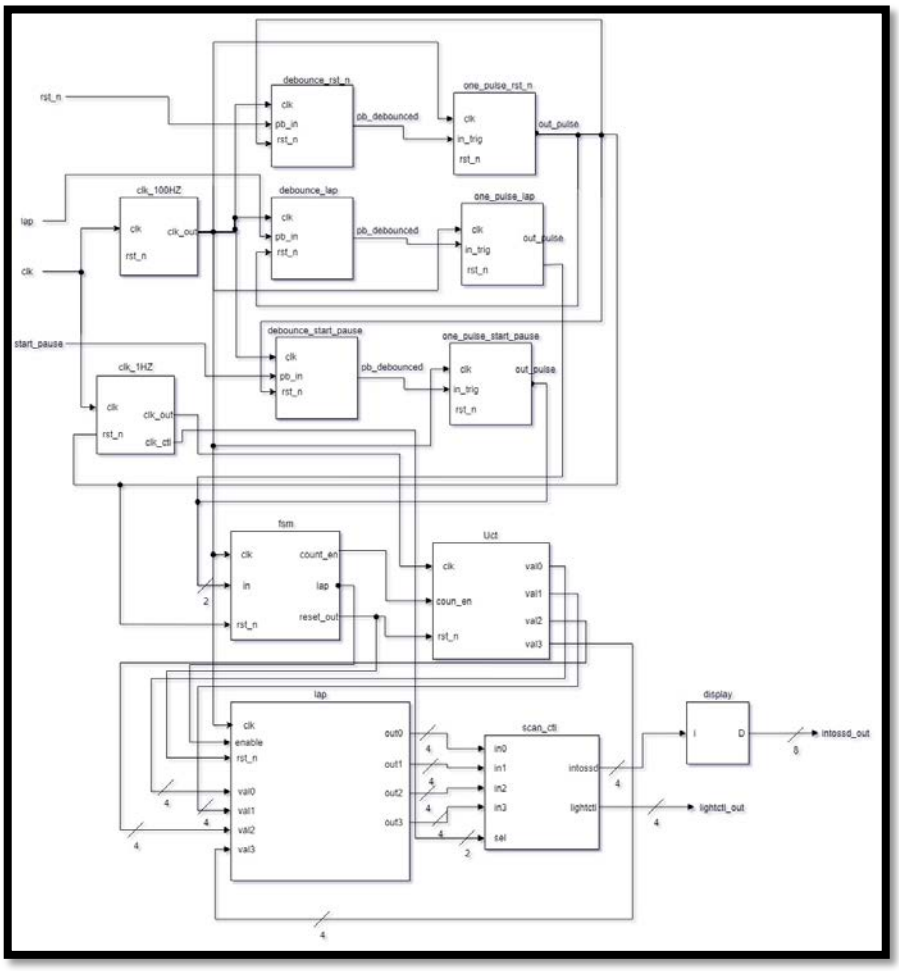
Input start_pause; // select start or pause

Input lap; // select lap

Output [3:0]lightctl_out;

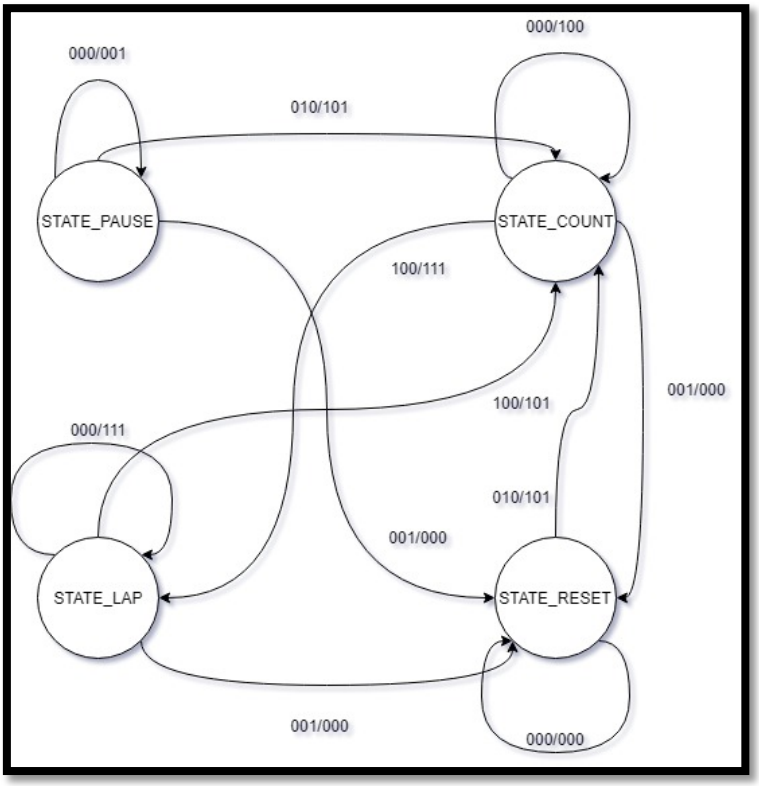
Output [7:0]display;

✓ Logic Block:



這一題button input的部分共有三個，rst_n，start_pause，lap。最前面一樣使用了debounce以及one pulse的訊號處理。訊號進入fsm會根據輸入來做counter的應變。然後lap模組主要是控制現在數字要顯示什麼，後面會有詳細的說明。。然後最後的部分顯示數字用。

Fsm:

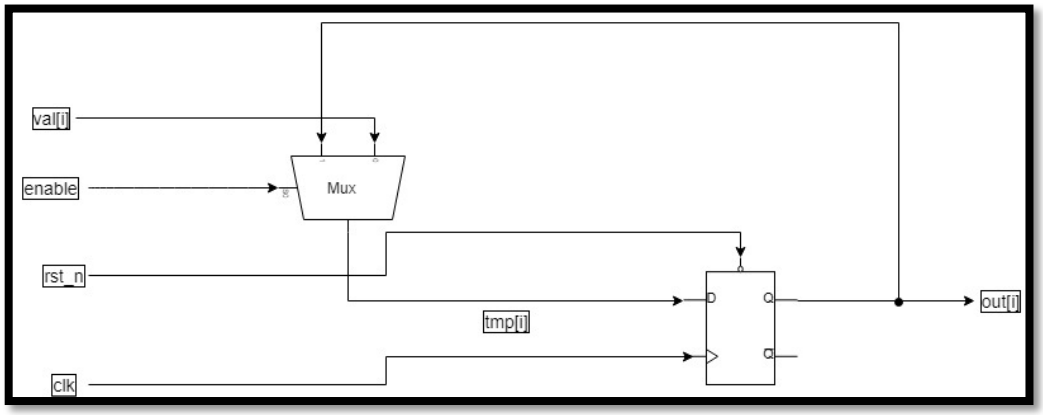


Input: {lap, start_pause, rst_n}

Output: {count_enable, lap_en, reset_out}

- Count_enable: 控制counter計數的訊號
- Lap_en: 控制lap模組的訊號。
- Reset_out: 控制counter是否reset的訊號（跟rst_n有相同功能，只是是combinational logic）

Lap:



在我的設計中共有四個這樣的logic存在lap模組中，分別需要判斷四個bit。上面的圖我用了其中一個bit來作代表，其餘邏輯相同。Val[i]為counter進來的input，為真實counter的值。Enable來自fsm，只要counter現在暫停，enable就會變成1，這時lap模組輸出會是前一個clock cycle儲存的值，直到使用者按下繼續，counter繼續計數之後，才會繼續顯示counter送來的值。

✓ Design Implementation

Input			
clk	*rst_n*	*lap*	*start_pause*
W5	T17	W19	U18

Output ssd							
display[0]	***display[1]***	***display[2]***	***display[3]***	***display[4]***	***display[5]***	***display[6]***	***display[7]***
V7	U7	V5	U5	V8	U8	W6	W7

Output lightctl			
lightctl[0]	*lightctl[1]*	*lightctl[2]*	*lightctl[3]*
U2	U4	V4	W4

✓ Discussion

這一題要做的功能算是相當特殊，不太知道什麼時候會用到。但是算是小有挑戰性，或許功能並沒有很實用，但是要做出來也不太容易，算是對顯示器與clock功能作更大的挑戰。在做的時候發現暫停的時候有時候會有紊亂的信號，counter經常自己亂跳。Debug好久還是找不到錯誤在哪邊。後來發現原來我從剛開始除頻的地方就沒有做好，導致100HZ與1HZ無法對齊，導致counter亂數的狀況。改完了clock總算一切都恢復正常，害我找了好久沒時間作bonus。

✓ Experiment

2 Implement a timer (can support as long as 23:59) with the following functions.

Design Specification

✓ I/O

```
Input clk;          // crystal clock
Input rst_n;        // low active reset
Input in_sec;       // set the second in setting mode
Input in_min;       // set minute in setting mode
Input in_pause;     // pause the counter
Input in_stop;      // stop the counter
```

Input in_set; // switch to setting mode

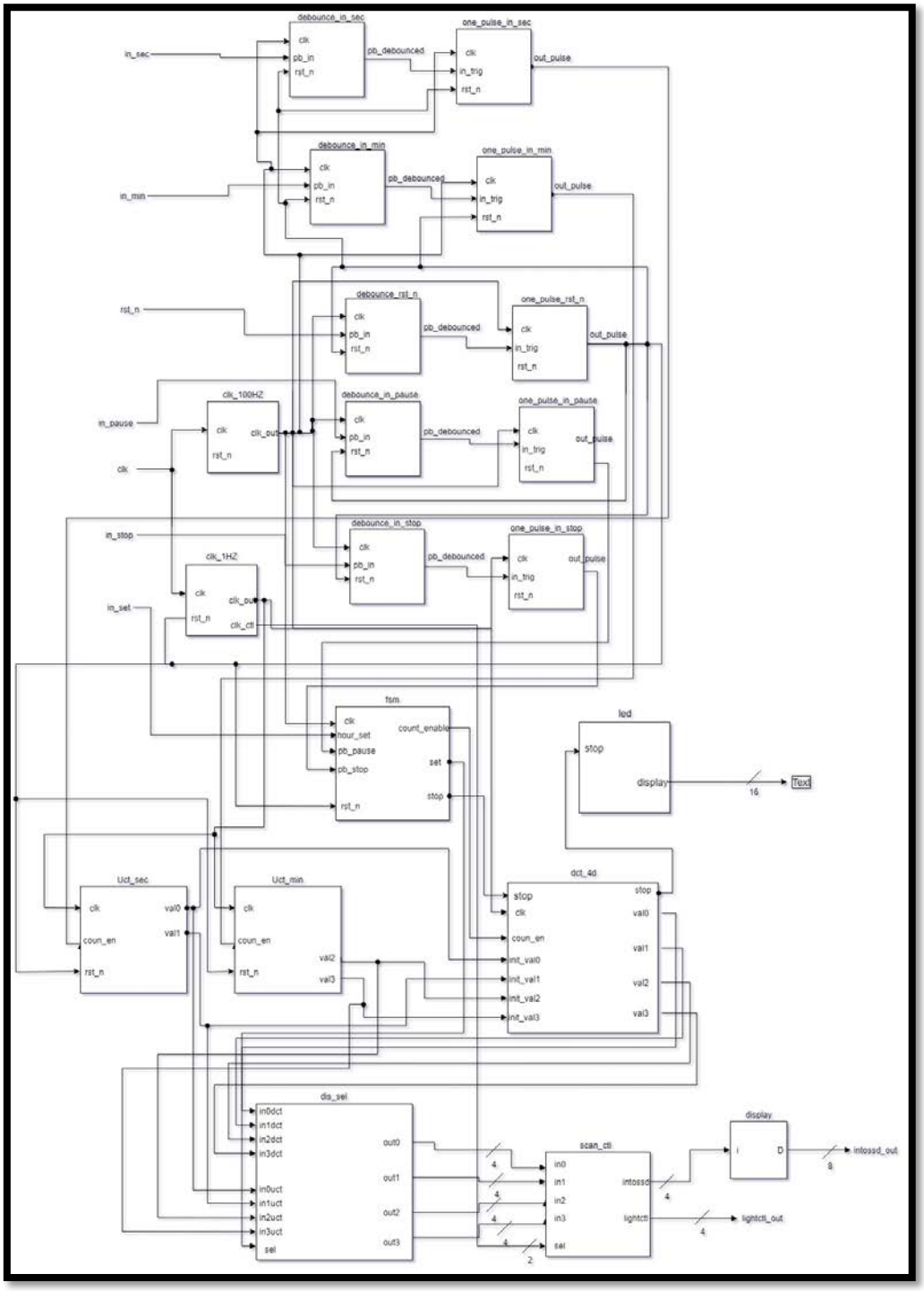
// display outputs

Output [3:0]lightctl_out;

Output [7:0]display;

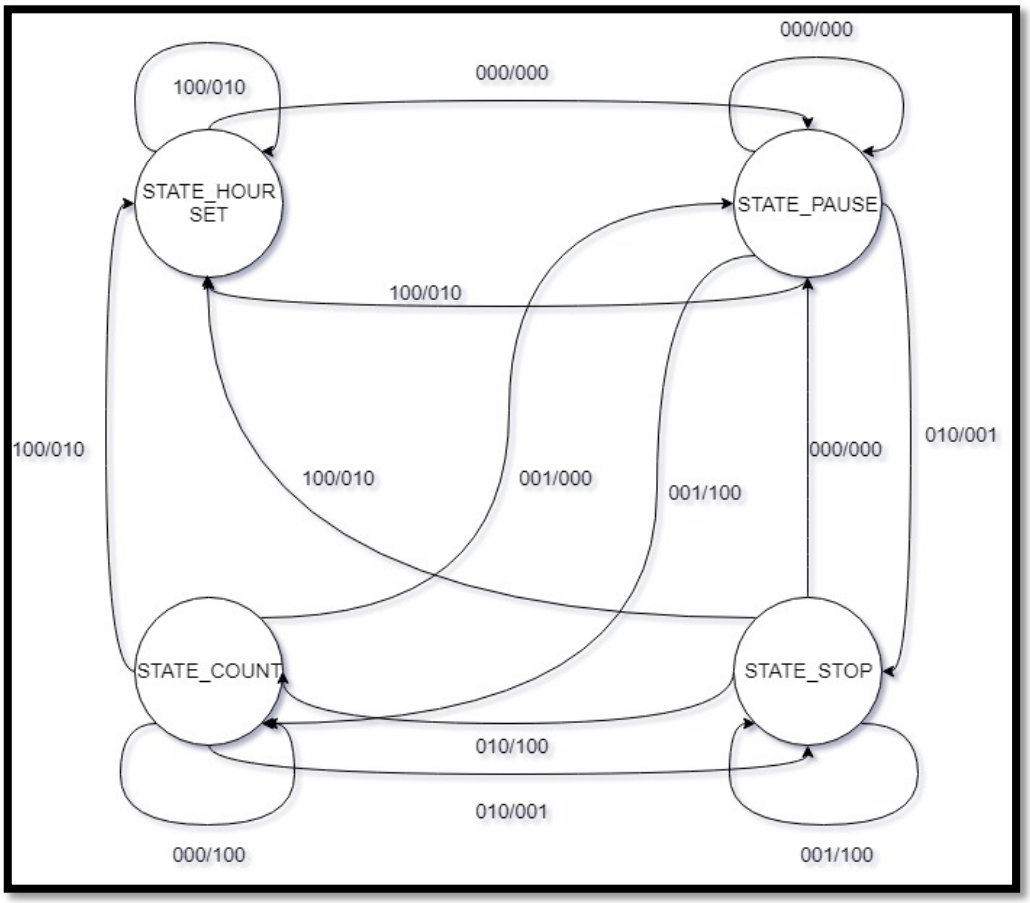
Output [15:0]led

✓ Logic Block:



輸入共有五個按鍵需要做訊號處理，in_set為dip-switch。當in_set為0時，counter會做正常的計數，暫停，停止功能...前面的lab已經做很多次，不再多敘述。當in_set為1時啟動setting模式，這時in_sec，in_min作為upcounter的count_enable。Up counter共有兩個，一個當作分，另外一個當作秒，每按一下加一，然後這些值會連動到down counter的initial value。在setting模式下，後面的dis_sel模組會切換到顯示upcounter，類似mux的功能去做選擇，使用者就可以到目前initial value被設定到多少。離開setting模式，就會回歸顯示downcounter。

FSM



Input: {hour_set, pb_stop, pb_pause}

Output: {count_enable, set, stop}

- Count_enable: 控制down counter的計數。
- Set: 交給後面的dis_sel模組，決定是否顯示up counter還是顯示down counter。
- Stop: 會傳進down counter中，功能類似reset counter，但是是combinational logic。

✓ Design Implementation

Input						
clk	rst_n	in_pause	in_stop	in_set	in_min	in_sec

W5	T17	U18	W19	R2	T18	U17

Output ssd							
display[0]	***display[1]***	***display[2]***	***display[3]***	***display[4]***	***display[5]***	***display[6]***	***display[7]***
V7	U7	V5	U5	V8	U8	W6	W7

Output lightctl			
lightctl[0]	*lightctl[1]*	*lightctl[2]*	*lightctl[3]*
U2	U4	V4	W4

LED(略)

✓ Discussion

這一題剛開始做的時候一點想法都沒有。關於設定的部分我想了好久，希望可以找出一個錯誤率比較低的作法。想了好久才想到用 upcounter 來設定的方法。這一題在想法上不難，但是模組數量龐大，而且 fsm 的 state 越來越多，如果一次做完容易造成有 bug 不好找。剛開始時我也是一次全部打完，後來 bug 真的好多好難找。後來是同學加上分成好多模組，才終於找出錯誤的地方。這次的 lab 我寫了很多 testbench 來驗證我的模組，發現了很多 vivado 好用的功能，譬如將其他模組的 behavior 同時拉進來看等等功能…總之 verilog debug 真的事好一門學問。

✓ Conclusion

這次的 lab 題目都類似，但是難度逐漸提升，複雜度也提高很多。Fsm 越來越複雜，打起來也越來越吃力…。看到下次的 lab 總算換主題了，期許可以增添更多的樂趣。