Lab6_1

Design Specification

✓ For a clock with second, minute, and hour:

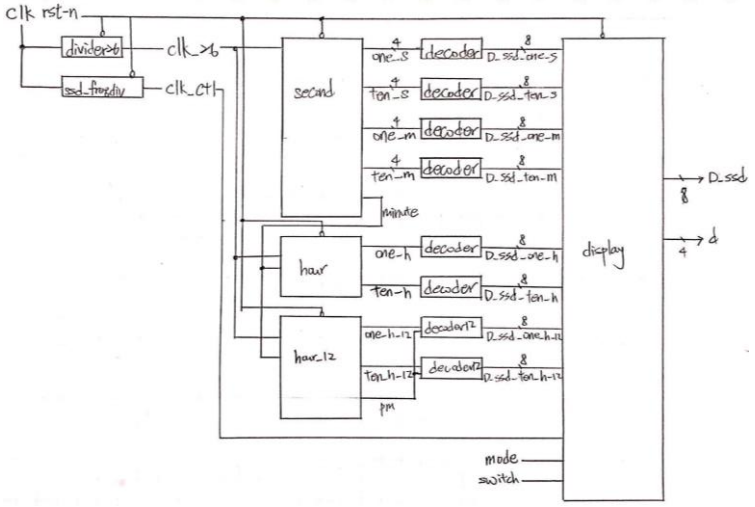
Input:

switch // 時分與秒切換
rst_n // control rst_n button
clk
mode // 24 小時制與 12 小時制切換
clk_c // 控制秒的頻率(非必要，因此無在 block diagram 中顯示)

Output:

D_ssd[7:0] // 7-segment display
d[3:0]

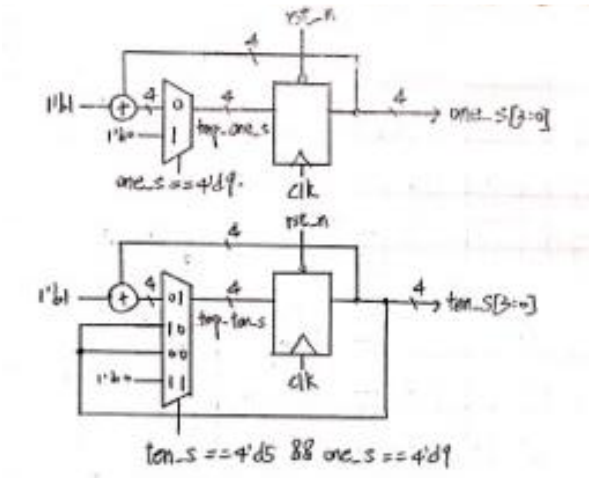
✓ Draw the block diagram of the design.



Design Implementation

✓ 本題由 ssd_freqdiv、divider26、second、hour、hour_12、decoder、decoder12、display 八個 module 組成，其中 second、hour、hour_12 皆為 upcounter，其差異進位條件不同；decoder 與 decoder12 的差異在於 7-segment display 的點有沒有亮。以下將分別說明 second、hour、hour_12、display 的功能，而 ssd_freqdiv、divider26、decoder 則在先前的 lab 即使用過，將不在贅述。

✓ Second、hour、hour_12



Second 的功能為計算秒與分，主要組成為 BCD upcounter。上圖為秒的部份的示意圖。而分

的部分由於結構類似，因此將以文字說明。

- 分的個位數部分

MUX 條件			MUX output
one_s == 4'd9	ten_s == 4'd5	one_m == 4'd9	one_m
default			one_m
1	1	0	tmp_one_m
1	1	1	4'd0

- 分的十位數部分

MUX 條件				MUX output	
one_s == 4'd9	ten_s == 4'd5	one_m == 4'd9	ten_m == 4'd5	ten_m	minute
1	1	1	0	tmp_ten_m	1'b0
1	1	1	1	4'd0	1'b1
default				ten_m	1'b0

十位數部分多加了一個 minute 作為整個 second 的 output，其作用為判斷 hour 要不要進位。

Hour 的部份其結構亦與上圖相似，差異在於 MUX 條件。

- 時的個位數部分

MUX 條件			MUX output
ten_h == 4'd2 && one_h == 4'd3	one_h == 4'd9	minute	one_h
0	1	1	4'd0
1	0	1	4'd0
0	0	1	tmp_one_h
default			one_h

其中(ten_h == 4'd2 && one_h == 4'd3)與 one_h == 4'd9 不可能同時成立。

- 時的十位數部分

MUX 條件			MUX output
ten_h == 4'd2 && one_h == 4'd3	one_h == 4'd9	minute	ten_h
0	1	1	tmp_ten_h
1	0	1	4'd0
default			ten_h

Hour_12 為 12 小時制的模式，在我的設計中凌晨 12 點為 00。

- 時的個位數部分
  - 若 minute == 1
    - 若為凌晨 00 點或中午 12 點，則 one_h_12 為 1
    - 若為早上 09 點或晚上 11 點，則 one_h_12 為 0
    - 其餘 one_h_12 加 1
  - 若 minute == 0，one_h_12 維持不變
- 時的十位數部分
  - 其中加入 pm 表示下午
  - 若 minute == 1
    - 若為晚上 11 點或中午 12 點，則 ten_h_12 為 0；pm <= (~pm)
    - 若為早上 09 點，則 ten_h_12 為 1；pm <= pm

- 若 minute == 0，ten_h_12 維持不變；pm <= pm

Input & Output

second		hour		hour_12	
input	output	input	output	input	output
clk	one_s	clk	one_h	clk	one_h_12
rst_n	ten_s	rst_n	ten_h	rst_n	ten_h_12
	one_m	minute		minute	pm
	ten_m				
	minute				

✓ Display

此為控制螢幕顯示時間的裝置，其中 switch 為控制顯示時分或秒，mode 為控制時制。表中的 c 隨頻率改變，每個頻率所顯示的時間位數不同，透過視覺暫留讓我們同時看到不同數字；d 則是控制 4 個 7-segment display 亮。

switch	mode	c	D_ssd	d
1	X	00	D_ssd_one_s	4'd1110
1	X	01	D_ssd_ten_s	4'd1101
1	X	default	8'b11111111	4'd0000
0	1	00	D_ssd_one_m	4'd1110
0	1	01	D_ssd_ten_m	4'd1101
0	1	10	D_ssd_one_h	4'd1011
0	1	11	D_ssd_ten_h	4'd0111
0	1	default	8'b11111111	4'd0000
0	0	00	D_ssd_one_m	4'd1110
0	0	01	D_ssd_ten_m	4'd1101
0	0	10	D_ssd_one_h_12	4'd1011
0	0	11	D_ssd_ten_h_12	4'd0111
0	0	default	8'b11111111	4'd0000

✓ I/O pin

I/O	D_ssd[7]	D_ssd[6]	D_ssd[5]	D_ssd[4]	D_ssd[3]	D_ssd[2]	D_ssd[1]	D_ssd[0]
VOC	W7	W6	U8	V8	U5	V5	U7	V7
I/O	d[3]	d[2]	d[1]	d[0]	clk	rst_n	mode	switch
VOC	W4	V4	U4	U2	W5	V17	V16	W16

Lab6_2

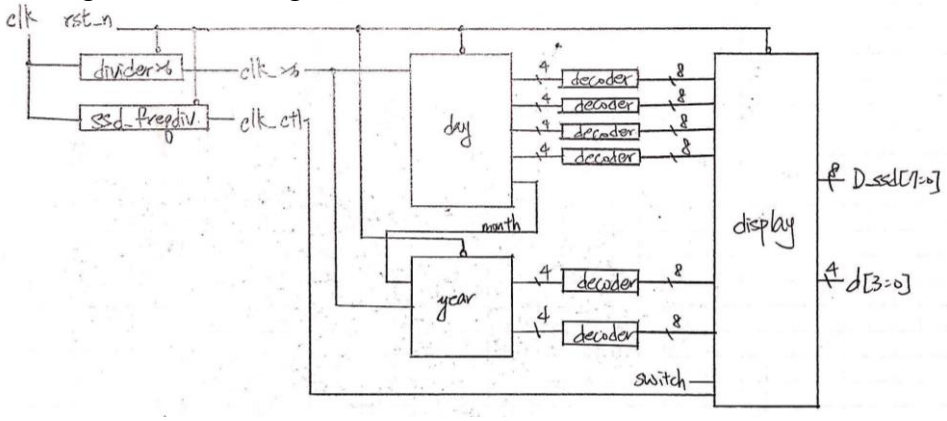
Design Specification

✓ For a calendar:

Input: clk,
rst_n,
switch // 切換年與月日

Output: D_ssd[7:0], d[3:0]

✓ Draw the block diagram of the design.



Design Implementation

✓ 此設計與 lab6-1 相似，由 ssd_freqdiv、divider26、day、year、decoder、display 六個 module 組成，其中 day 與 year 皆為 upcounter，其差異進位條件不同。以下將分別說明 day 與 year 的功能，其餘與 lab6-1 相同，將不再贅述。

✓ day 與 year

兩者與第一題的 second 結構相似，因此在此不再作圖，將以文字敘述 MUX 的判斷。

day：

在這裡我加入了 month_ctl_one 與 month_ctl_ten 作為月份天數的判斷，用以判斷天數是否進位或改變。當 one_d == month_ctl_one 且 ten_d == month_ctl_ten 時，代表過了一個月。

- 日的個位數部分

MUX 條件			MUX output
one_d == month_ctl_one	ten_d == month_ctl_ten	one_d == 4'd9	one_d
1	1	X	4'd1
X	0	1	4'd0
0	X	1	4'd0
default			tmp_one_d

- 日的十位數部分

MUX 條件			MUX output
one_d == month_ctl_one	ten_d == month_ctl_ten	one_d == 4'd9	ten_d
1	1	X	4'd0
X	0	1	tmp_ten_d
0	X	1	tmp_ten_d
default			ten_d

- 月的個位數部分

MUX 條件				MUX output
one_d == month_ctl_one	ten_d == month_ctl_ten	one_m == 4'd2 && ten_m == 4'd1	one_d == 4'd9	one_m
1	1	1	X	4'd1
1	1	0	1	4'd0
1	1	0	1	4'd0
1	1	0	0	tmp_one_m
default				one_m

- 月的十位數部分
  - 十位數部分多加了一個 month 作為整個 day 的 output，其作用為判斷 year 要不要進位。
  - 若為 12 月且過了一個月，ten_m 為 0，month 為 1
  - 否則若為 9 月且過了一個月，ten_m 為 1，month 為 0
  - 其餘狀況 ten_m 維持不變，month 為 0

year：

- 年的個位數部分

MUX 條件		MUX output
one_y == 4'd9	month	one_y
0	1	tmp_one_y
1	1	4'd0
X	0	one_y

- 年的十位數部分

MUX 條件			MUX output
ten_y == 4'd9	one_y == 4'd9	month	ten_y
1	1	1	4'd0
0	1	1	tmp_ten_y
X	X	0	ten_y

年的部份較簡單，只要藉由 month 判斷是否進位即可，其餘為 BCD counter 的應用。

✓ I/O pin

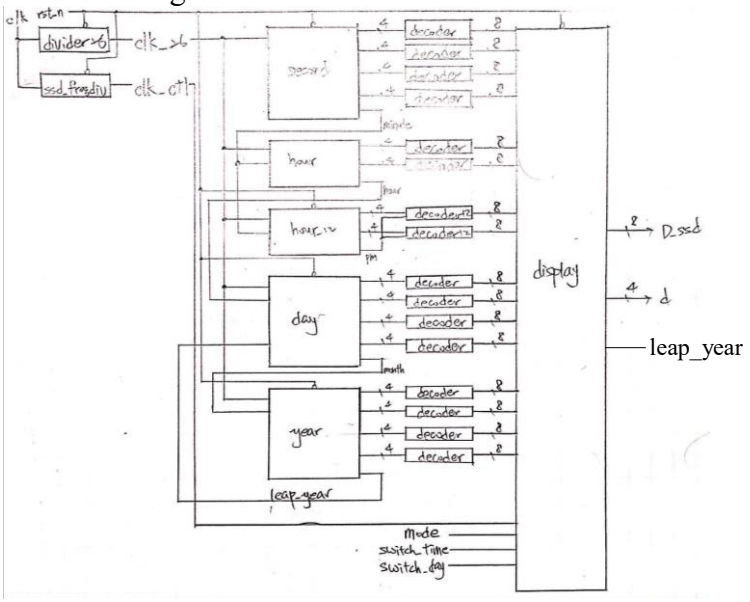
I/O	D_ssd[7]	D_ssd[6]	D_ssd[5]	D_ssd[4]	D_ssd[3]	D_ssd[2]	D_ssd[1]	D_ssd[0]
VOC	W7	W6	U8	V8	U5	V5	U7	V7
I/O	d[3]	d[2]	d[1]	d[0]	clk	rst_n	switch	
VOC	W4	V4	U4	U2	W5	V17	V16	

Lab6_3
Design Specification

✓ Input: clk,
rst_n,
switch_day // 切換年與月日
switch_time // 切換秒與時分
mode // 切換時制

Output: D_ssd[7:0], d[3:0], leap_year

✓ Draw the block diagram of the design.



Design Implementation

✓ 第三題為第一題與第二題的結合，其中有二差別。一為日期需要利用小時去判斷是否加一，我將兩個之間利用 hour 連結作為 hour 的 output 與 day 的 input，藉此作為判斷依據。另一差別為第三題的年為 2000~2200，且需判斷閏年。以下將說明我的作法

✓ Year

Input：clk, rst_n, month

Ouput：one_y, ten_y, hun_y, thu_y, leap_year

- 其中年的千為數恆為 2，百位數、十位數、個位數為 BCD counter，跟第二題部分相似，並在 2200 年時回到 2000 年重新計算。
- Leap_year 作為閏年的判斷。其中我設了一個 11 bits 的變數 leap_cnt 作為計數器，隨著年份增加而增加(即為年的 binary 形式)。判斷閏年的條件為

leap_year = leap_cnt % 4 == 0 && (leap_cnt % 400 == 0 || leap_cnt % 100 != 0)

利用and與or的性質，加上很厲害的邏輯，就可以利用一條式子就判斷是否為閏年。

✓ I/O pin

I/O	D_ssd[7]	D_ssd[6]	D_ssd[5]	D_ssd[4]	D_ssd[3]	D_ssd[2]	D_ssd[1]	D_ssd[0]
VOC	W7	W6	U8	V8	U5	V5	U7	V7
I/O	d[3]	d[2]	d[1]	d[0]	clk	rst_n	leap_year	mode
VOC	W4	V4	U4	U2	W5	V17	U16	V16
I/O	switch_time	switch_day						
VOC	W16	W17						

Discussion

這次的 lab 都是 divider、BCD counter、decoder、display 由組成。其中 BCD counter 的 MUX 判斷為三題中的主要差異。我認為除了 MUX 的條件判斷為本次實驗重點外，還有另外一個重點為進位的條件判斷。

在第一題的時候，我一開始將分與秒分開做，但發現會遇到無法處理兩者的連結，所以之後將二者放在同一個 module。但如果將時分秒都放在一起感覺很複雜，很難 debug，所以之後又想到可以藉由 output 一個進位的 enable 來作為判斷，因此在 hour 的時候就分開做了。

在做第二題時，原本我並沒有加入判斷每個月天數的變數，因此在寫日期進位條件的時候

遇到了很大的麻煩，很容易出錯。因此我加入了 month_ctl_one 與 month_ctl_ten 幫助判斷是否進位，有效解決了判斷式複雜容易出錯的問題。另外，我原本想利用上題的方式，利用 enable 判斷月份是否加一，但卻發生了月份改變會慢一個 clk 的問題。就其原因為一開始的設計為在月份要改變的當下 enable 才會為 1，因此會到下個 clk 時月份才改變。因此最後還是利用原本的將月份與日期做在同一個 module，並利用 month_ctl_one 與 month_ctl_ten 判斷月份是否改變。

最後是第三題閏年的判斷。其判斷方式有很多種，例如先判斷是否為 4 的倍數，再判斷是否為 100 的倍數，最後判斷是否為 400 的倍數；藉由三次判斷得出答案。此方法雖然很容易理解，但需要經過三次判斷，較沒效率。因此我利用邏輯結構，藉由一個條件即可判斷結果，雖然較不直觀，但卻可以練習邏輯判斷，真的很方便。另外一個差異是，是否需要多設置一個變數紀錄年份用來判斷。我認為多加一個變數會讓程式看得比較清楚，缺點是須注意其值什麼時候要改變，整個程式也會比較冗長。

Conclusion

這次的實驗著重於細節的判斷，若沒注意年月日時分秒的進位，就很容易出問題，因此需要很仔細的思考。另外一個細節是年月日時分秒的連結，須注意是否會快或慢一個 clock，需要清楚自己 enable 的設計以免出錯。

