1 Finish the time display function supporting 24-hour (00-23).

1.1 Can display as hour:minute and second, and use a push button or DIP switch to switch the display.

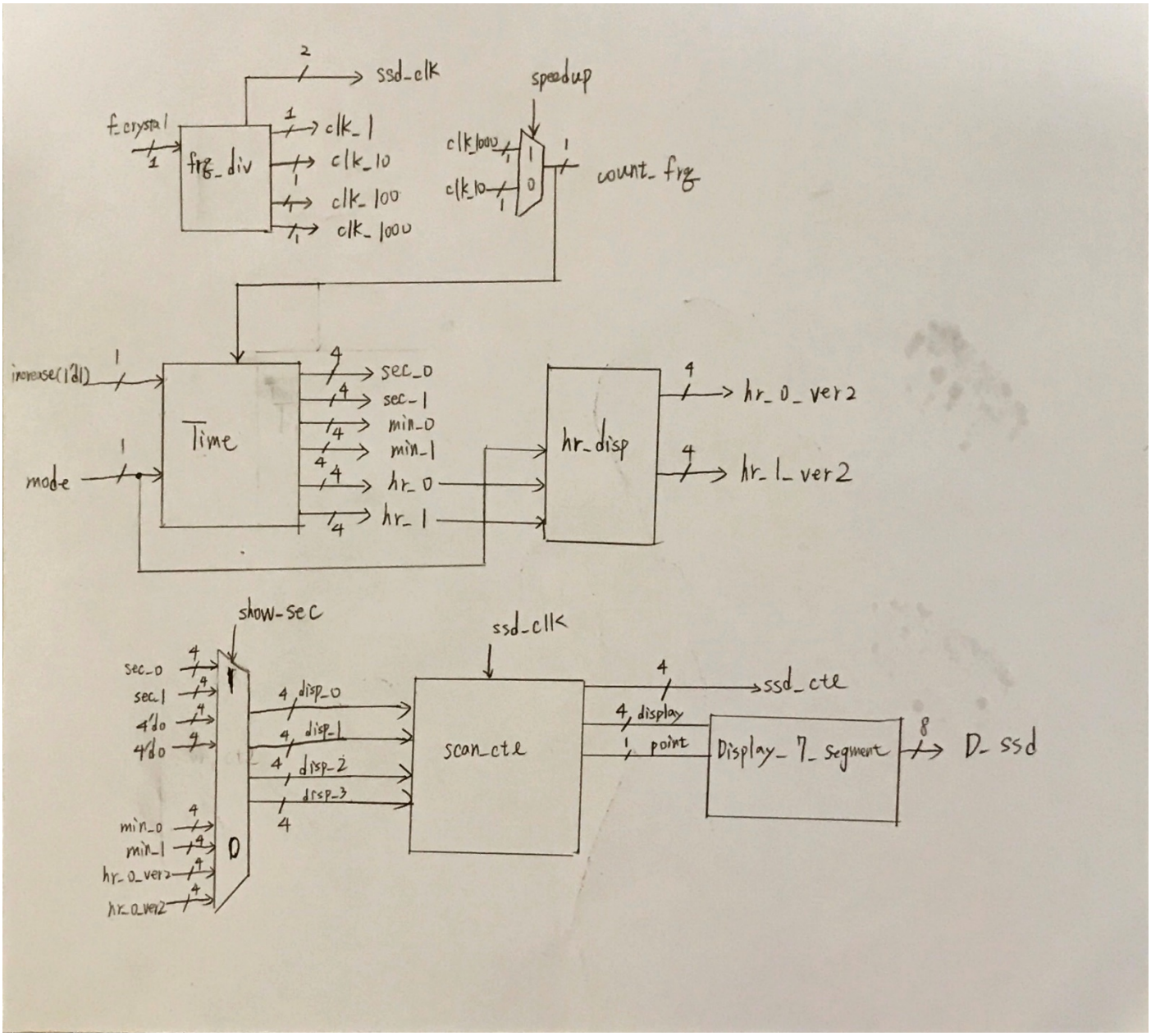
1.2 Support two modes: AM/PM and 24-hour.

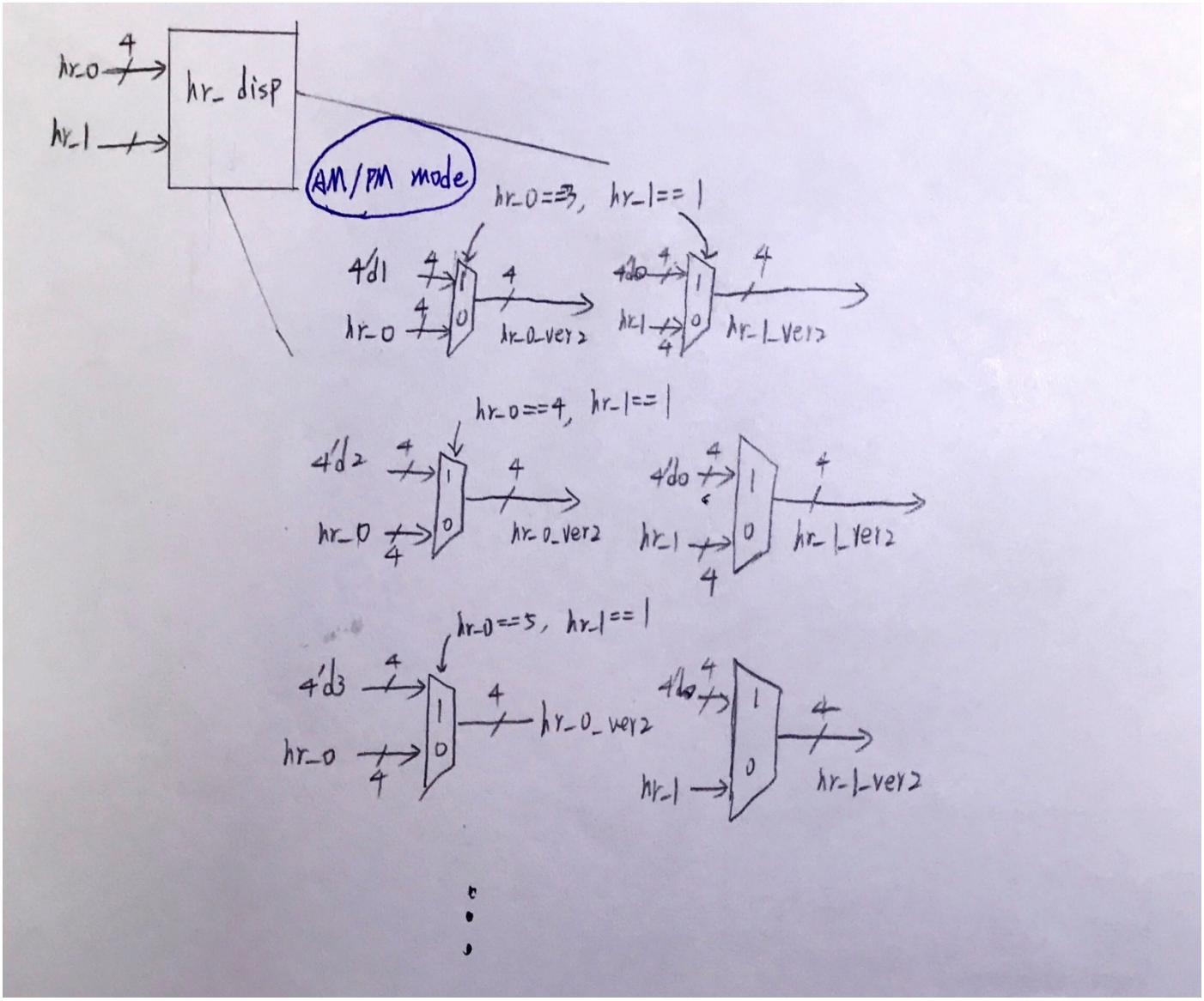
Design Specification:

Input: f_cyrstal, rst, show_sec, mode, speedup

Output: ssd_ctl, D_ssd

Block Diagram:





In hr_disp module, I didn't draw the whole detail as the structure is similar and repeated.

Design Implementation:

I/O pin assignment:

ssd_ctl[3]—W4

ssd_ctl[2]—V4

ssd_ctl[1]—U4

ssd_ctl[0]—U2

D_ssd[7]—W7

D_ssd[6] — W6

D_ssd[5] — U8

D_ssd[4] — V8

D_ssd[3] — U5

D_ssd[2] — V5

D_ssd[1] — U7

D_ssd[0] — V7

f_crystal — W5

show_sec — T1

mode — U1

speedup — V17

rst — R2

To decrease the time display function, it's supposed to count just like clock in our daily life. So I assigned 1'b1 to 'increase' signal of Time (module). However, there is a limit for time in a day, which is 23:59. I used a MUX to check if time has reached the limit. If it's the case, I reset time so that it would starts from 00:00 again.

The hr_disp is supposed to be a big MUX but I drew it in many small MUXs in block diagram to express my idea. If it's AM/PM mode, and hour input is large than 12 than I re-assign it to be between 1~12. Otherwise, output would be just as the same as the input.

Also, according to signal of DIP switch to decide if 7-segment display shows the second or not. And as it would take a long time to count when using 1Hz clock frequency, I add other DIP function ('speedup' in block diagram) to choose faster clock.

Discussion:

In this experiment, the most important part is to support AM/PM and 24-hour function since we have done other parts of functions in previous Labs.

To support it, I used the most straight-forward method. If it's AM/PM mode and hour output is greater than 12, I reassign the output as the original output minus (-) 12 on two digits of hour respectively as showed in the block diagram.

Also, I note that if it's AM/PM mode and the time display is reset, it should be reset as 12 instead of 00.

2 For the date functions in clock (no leap year), we have the following functions:
- Day (Jan/March/May/July/Aug/Oct/Dec: 1-31, Feb: 28, Apr/June/Sept/Nov: 30),
- Month (1-12),
- Year (00-99).

Implement the following functions:

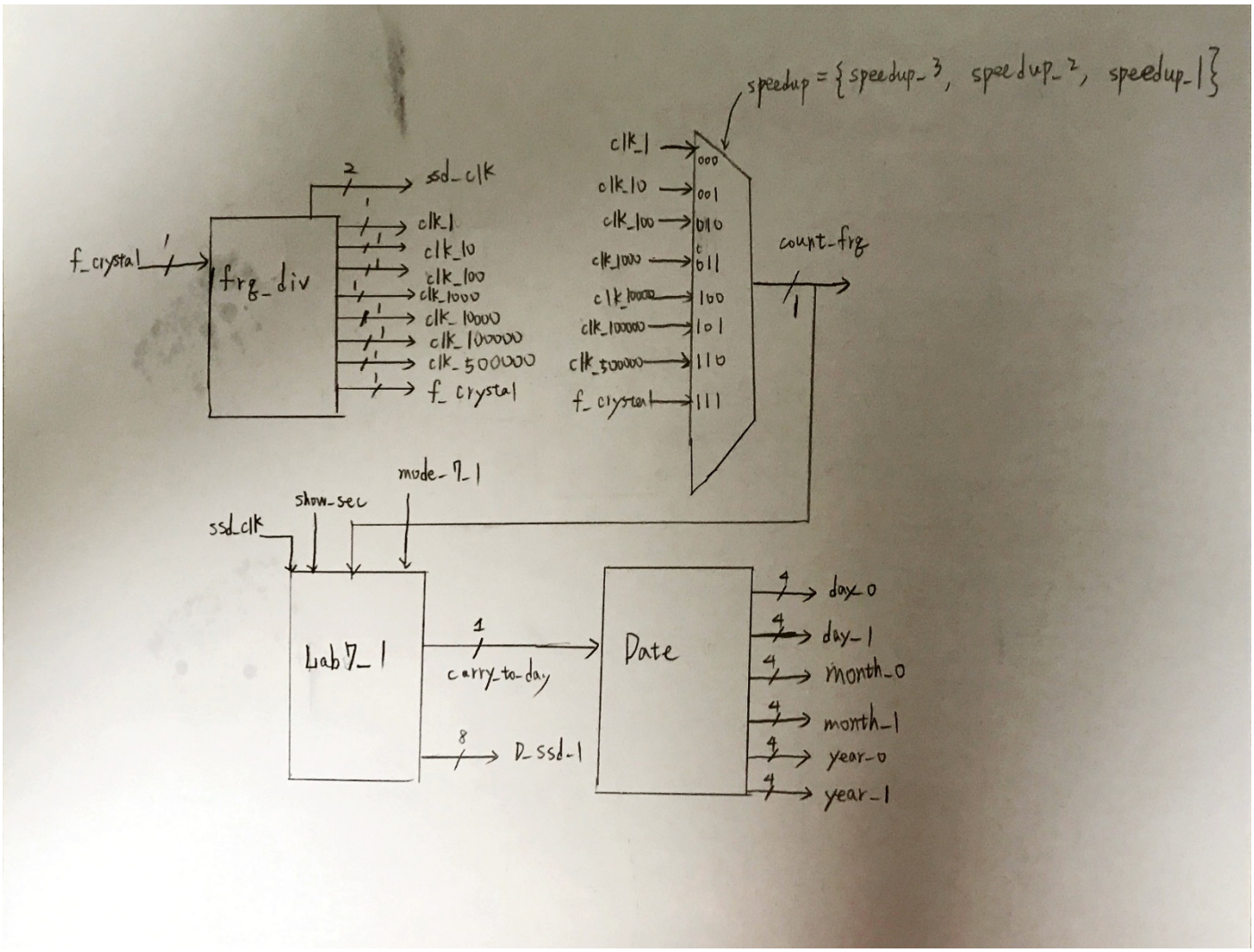
2.1 Month-Day function display in the 4 7-segment displays.
2.2 Combine the Year and 1.1 to finish a Year-Month-Day, and use one DIP switch to select the display of Year (2 Seven-Segment Displays, SSDs) or Month-Day (4 SSDs).

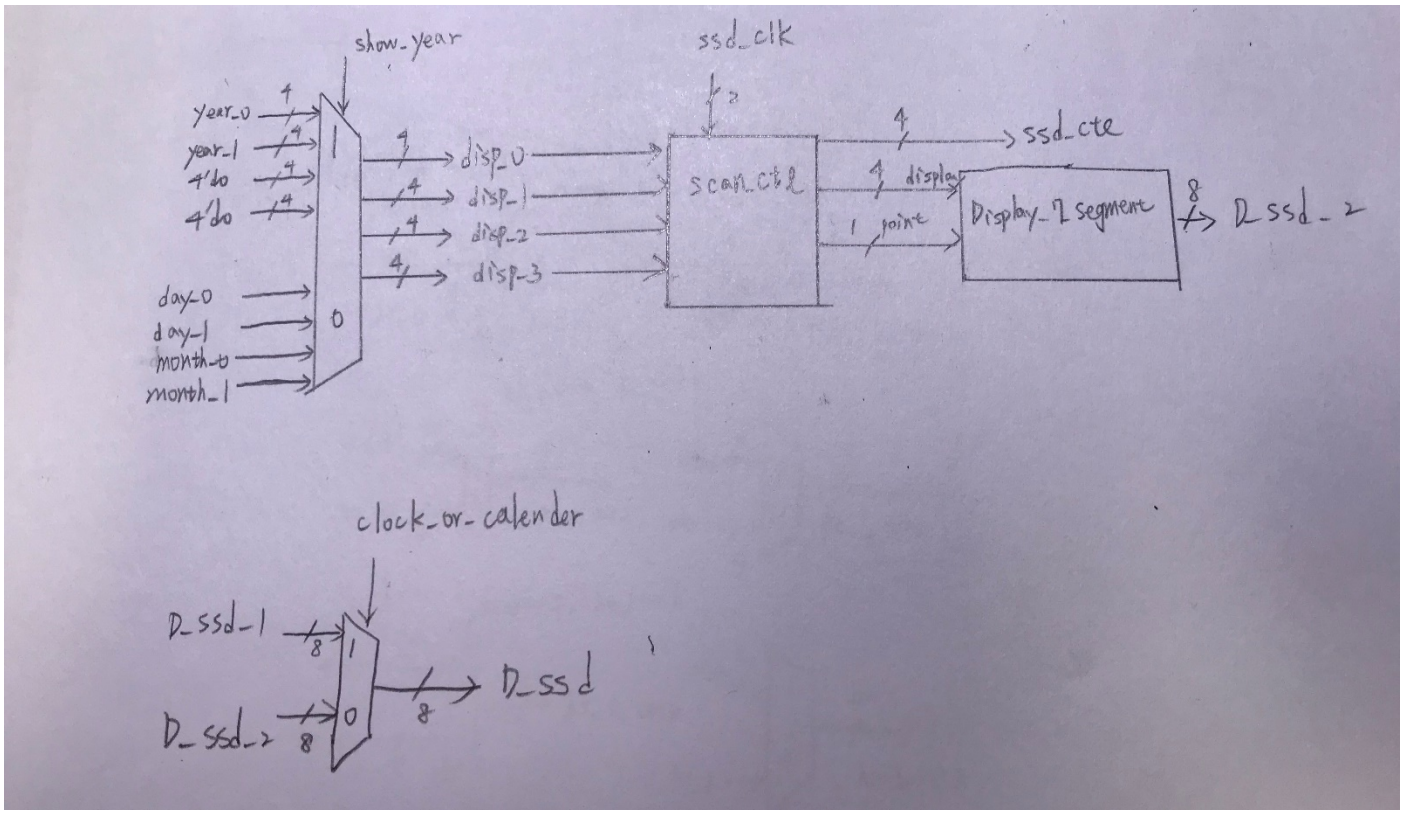
Design Specification:

Input: f_cyrstal, rst, show_year, speedup_1, speedup_2, speedup_3, show_sec, mode7_1, clock_or_calender.

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

Block Diagram:


speedup = {speedup_3, speedup_2, speedup_1}
f_crystal
frq_div
ssd_clk
clk_1
clk_10
clk_100
clk_1000
clk_10000
clk_100000
clk_500000
f_crystal
clk_1
clk_10
clk_100
clk_1000
clk_10000
clk_100000
clk_500000
f_crystal
000
001
010
011
100
101
110
111
count_frq
mode_7_1
show_sec
ssd_clk
Lab7_1
carry_to_day
D_ssd_1
Date
day_0
day_1
month_0
month_1
year_0
year_1


show_year
ssd_clk
year_0
year_1
4'd0
4'd0
day_0
day_1
month_0
month_1
disp_0
disp_1
disp_2
disp_3
scan_ctl
ssd_cte
display
point
Display_7 segment
D_ssd_2
clock_or_calender
D_ssd_1
D_ssd_2
D_ssd

Design Implementation:

I/O pin assignment:

ssd_ctl[3]—W4
ssd_ctl[2]—V4
ssd_ctl[1]—U4
ssd_ctl[0]—U2
D_ssd[7]—W7
D_ssd[6] — W6
D_ssd[5] — U8
D_ssd[4] — V8
D_ssd[3] — U5
D_ssd[2] — V5
D_ssd[1] — U7
D_ssd[0] — V7
f_crystal — W5
rst — R2
show_year — T1
speedup_1 — V17
speedup_2 — V16
speedup_3 — W16
show_sec — U1
mode7_1 — W2
clock_or_calender — W17

First of all, in this experiment we need clock for counter to count second, minute, hour, day, month and year. It is going to take a lot of time if the clock frequency is not high enough. Therefore, I design a frequency divider with 8 different clock frequency outputs controlled through 3 DIP switches to choose fast or slow clock frequency.

And as mentioned above, time would never stop. Therefore I assign 1'd1 for 'increase' signal of LSB of second (秒). And we have done hour, minute and second Lab7_1. I need to take its carry (carry_to_day in the block diagram) to the next level, which is day.

As for date (day, month, year), since each month has different days (ex: February has only 28 days but June has 30 days), I write a big MUX to set limit of counter accordingly for each month. And my counter design is to deal with each digit respectively (i.e. second, minute, hour, day, month, year all have 2 digits. Digit will pass carry to next digit.) So when being reset (ex: 23:59 (hour:second), 12:31 (month:day)…), I assign reset value to Flip Flop input so that Flip Flop could take it when next clock edge comes. At the same time, it passes carry to next digit. Therefore, I can use increase (carry from previous digit) AND reset to reset all other digit when the next clock edge comes.

Discussion:

The essential part of this experiment is how to pass the carry correctly to the next digit level. Because month ends in December (12) instead of 19 and day usually ends in 30/31 instead of 39, I am not able to just set limit for each digit like what we did in previous Lab so that the digit would pass carry automatically

when it reaches each its limit. Instead, I need to discuss different condition for each month respectively so that the function is consistent with what we see in our daily life.

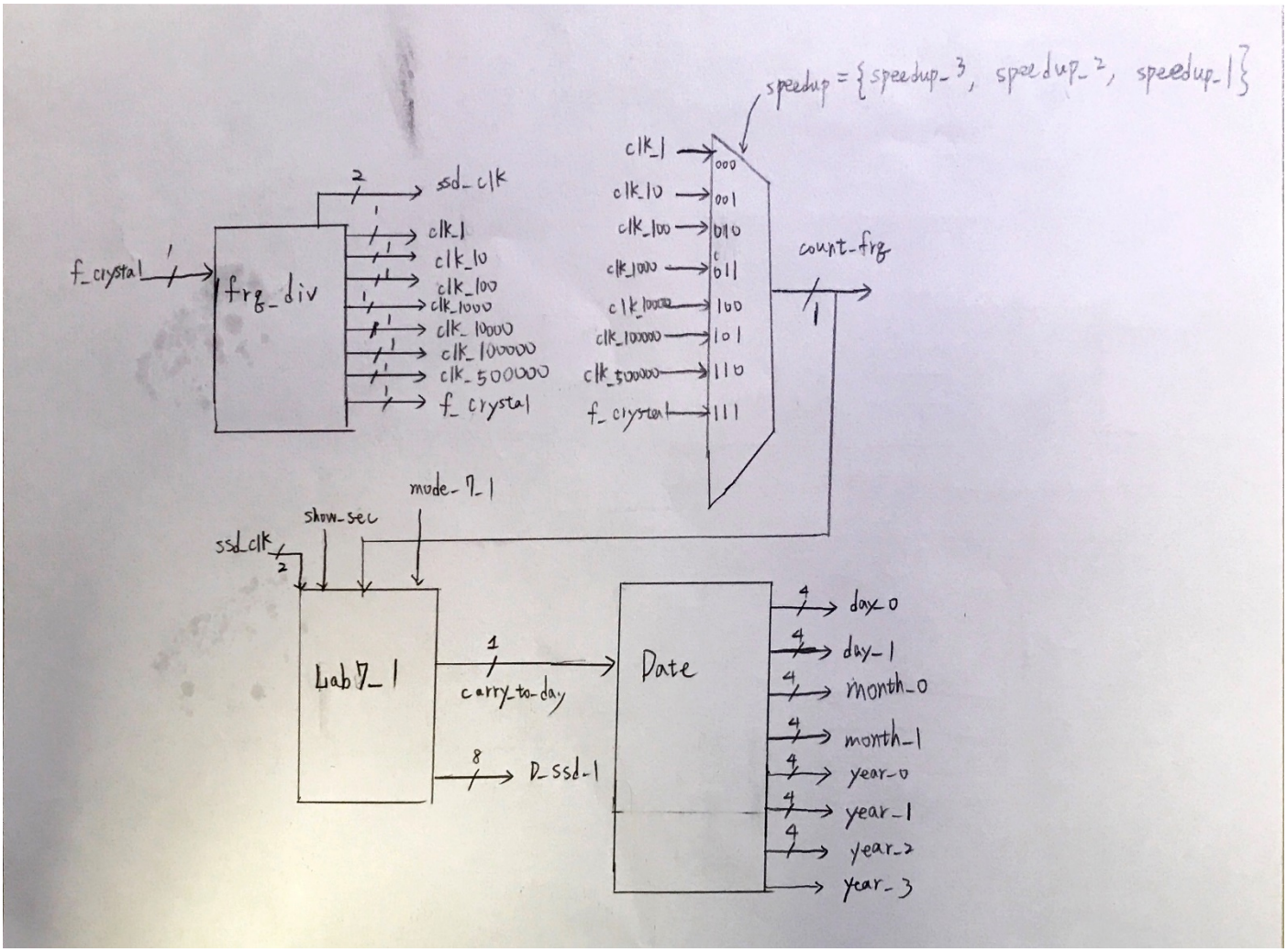
3 (Bonus) Add the time display support of both AM/PM and 24-hour, and the leap year support. (The year will start from 2000 to 2200 and use 4 SSDs to display.)

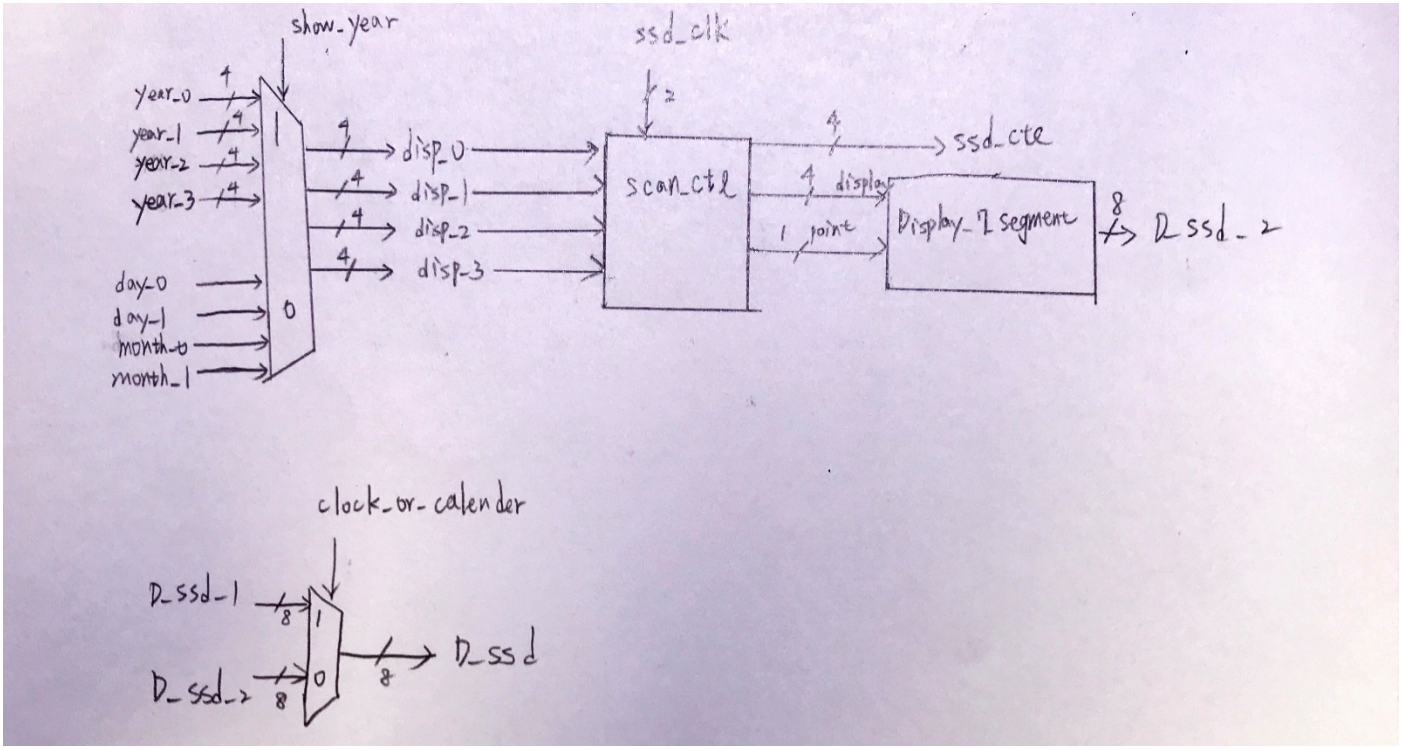
Design Specification:

Input: f_cyrstal, rst, show_year, speedup_1, speedup_2, speedup_3, show_sec, mode7_1, clock_or_calender.

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

Block Diagram:





Design Implementation:

I/O pin assignment:

ssd_ctl[3]—W4

ssd_ctl[2]—V4

ssd_ctl[1]—U4

ssd_ctl[0]—U2

D_ssd[7]—W7

D_ssd[6] — W6

D_ssd[5] — U8

D_ssd[4] — V8

D_ssd[3] — U5

D_ssd[2] — V5

D_ssd[1] — U7

D_ssd[0] — V7

f_crystal — W5

rst — R2

show_year — T1

speedup_1 — V17

speedup_2 — V16

speedup_3 — W16

show_sec — U1

mode7_1 — W2

clock_or_calender — W17

In this experiment, it's basically as same as the Lab7_2. The only difference is year has 4-digit display and leap year display.

For dealing with leap year, as there are 4 digits to represent (year_3, year_2, year_1, year_0). I first

convert these 4 digits into a decimal number by calculating year_3×1000 + year_2×100 + year_1×10 + year_0 and assign it another 11bits variable, year. Then according to the rule of leap year (每 4 年閏且每 100 年不閏每 400 年又閏， i.e. (year % 4 ==0 && year % 100 != 0) || year % 400 == 0). If it's leap year, February would have 29 days otherwise it would have 28 days. Hence, I only need to set the LSB digit of days to support leap year function.

Discussion:

As long as Lab7_2 is completely correctly, there should be no difficulty in completing Lab7_3. But I still forgot the rule of leap year, I need to google it before writing Verilog.

Conclusion for Lab7:

In this Lab, I have learned how to implement clock and calendar. Both of them are very common in our daily life. They look very easy before doing this Lab. During this Lab, bugs always occurred in boundary places, such as January 31st to February 1st. One of the important parts is how to pass carry to next level correctly (where I mention in Lab7_2). As soon as this issue is solved, other parts of function is basically relatively easy to complete.