**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**

**1. Specification**

**內容: 寫下你的電路中的inputs, outputs以及其bit widths，名稱必須跟你的verilog code中相同。**

module LAB7\_1\_top(

output [7:0] cathode, // 七段顯示器

output [3:0] anode, // 七段顯示器

input add\_second\_undebounced, // 按一下會加一秒

input add\_minute\_undebounced, // 按一下會加一分鐘

input add\_hour\_\_\_undebounced, // 按一下會加一小時

input switch\_setting, // 切換設定時間

input switch\_hour\_minute\_second, // 切換顯示小時、分鐘或秒

input switch\_12\_24, // 切換顯示12/24小時制

input [3:0] speedup, // demo時可以加速

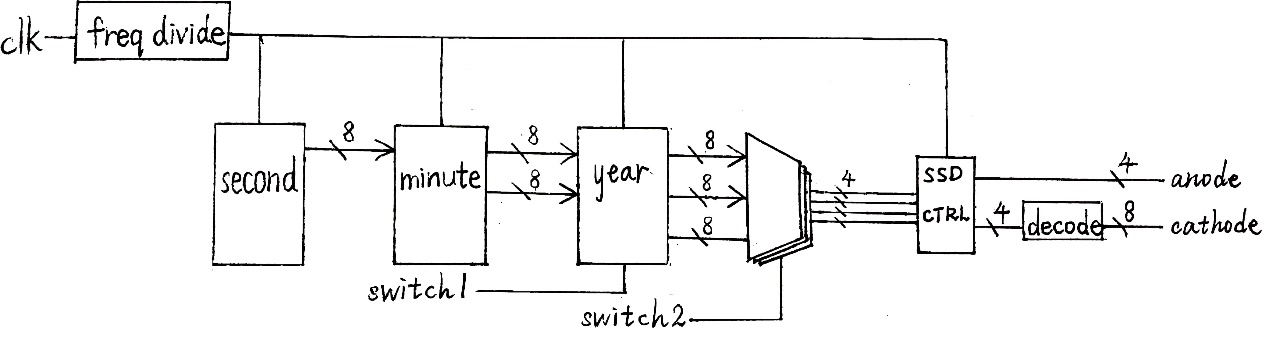
input clk\_100mhz, // FPGA內建的時脈

input rst\_n // 強制重置、初始化

);

**2. Block Diagram**

**內容: 電路中的Block diagram(可以用手畫拍照或電腦繪圖)。**



hour

**3. Finite state machine**

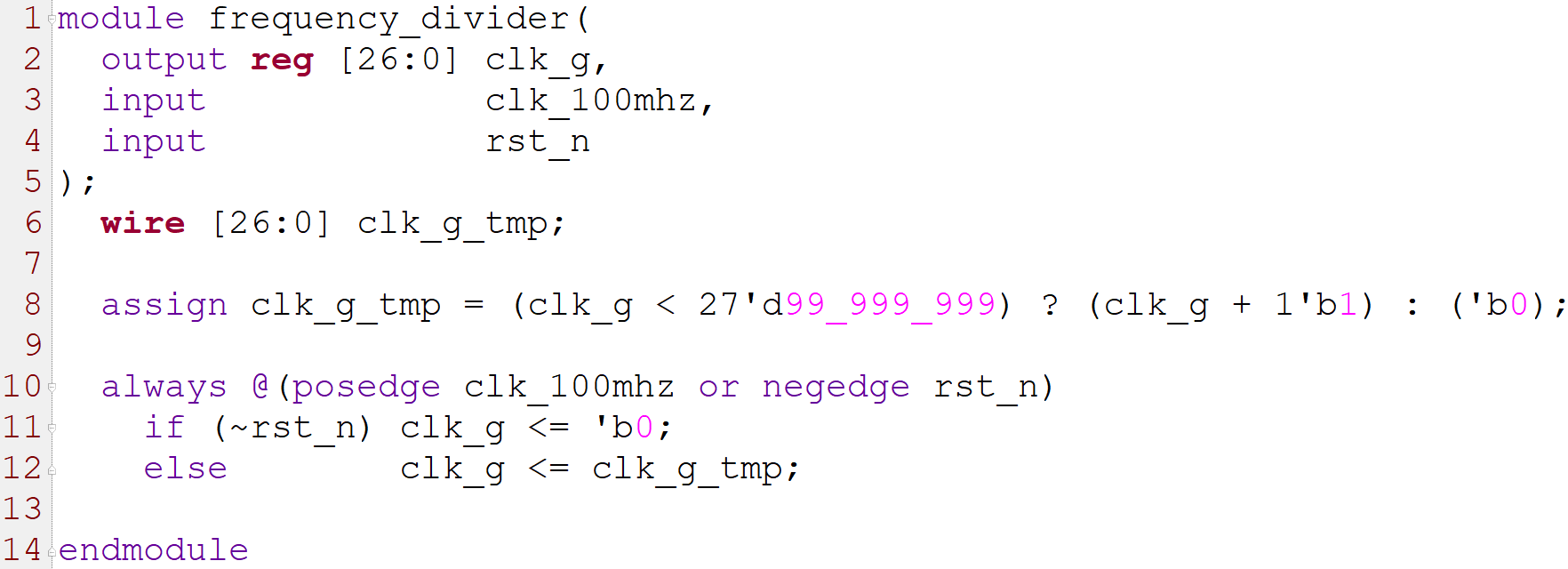
**內容: 電路中的Finite state machine，若無則寫無。**

無

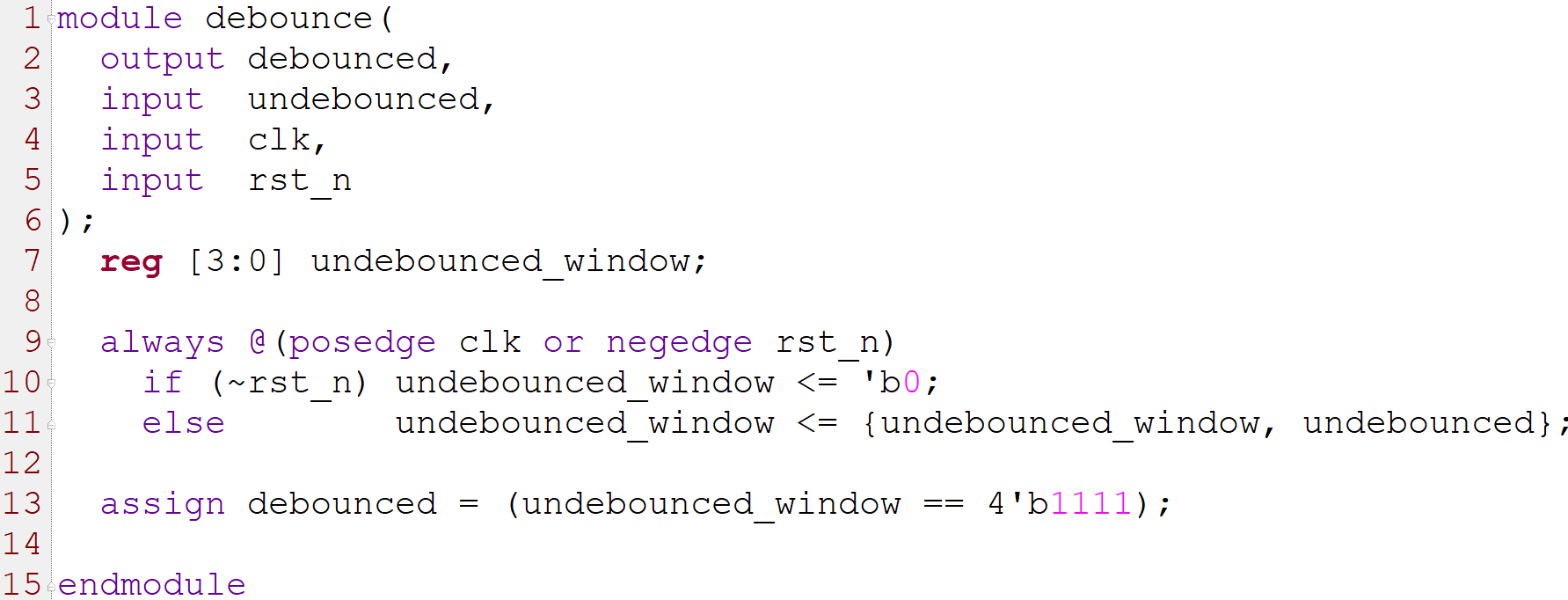
**4. Implement**

**內容: 請列出相關的logic function、詳細用文字解釋電路的運作方法、結果等等，可以貼code解釋或拍FPGA輔助解釋(但不能只貼code跟FPGA結果)。**

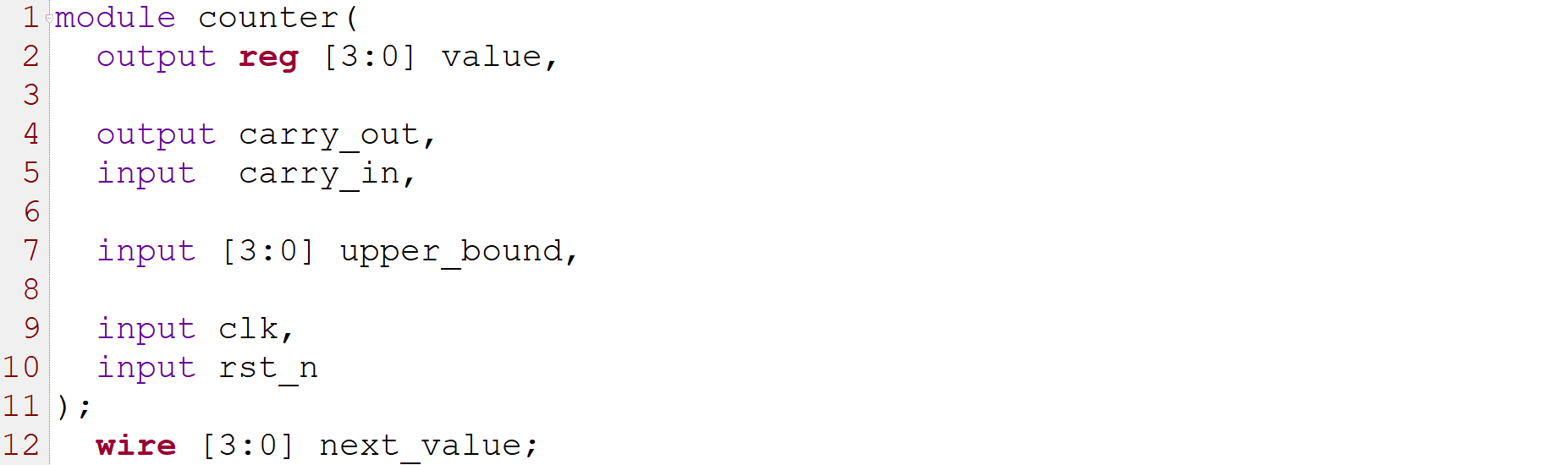
先用一個27位元的計數器構造一個除頻器。可製造出各種不同頻率的時脈，最慢的是一赫茲。如下圖，跟上次Lab的差不多。第8到12行是計數器。直接將除頻器的clk\_g當作output，讓top的其他模組可以參照各種頻率。這個除頻器裡的計數器每數到100M就會重置，所以如果有其他模組有需要使用到一赫茲的時脈的話，可以連接這個除頻器的計數器的最高位數clk\_g[26]。



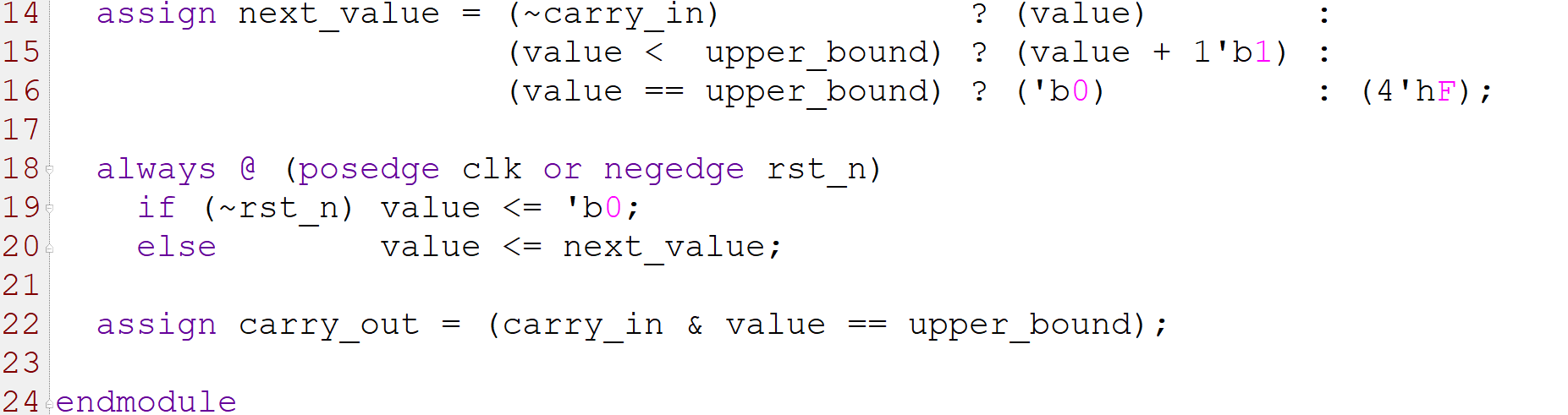
因為按鈕按下的瞬間會有擾動，有時候雖然只按一下，但卻會形成數個震盪。所以可以先把未處理的按鈕訊號連接到一個去除雜訊的模組。下圖是用來排除雜訊的，第9行到第11行是shifter。第13行用來判斷按鈕是否處於按下的狀態超過數個時脈。如果偵測到連續按下，那就代表按鈕已經穩定。



接下來是用來計時的四位元計數器。跟之前LAB的做法很像，在clk的rising edge的時候，利用carry\_in來判斷是否需要加一。如果要加一，再利用upper\_bound來判斷是否要進位。

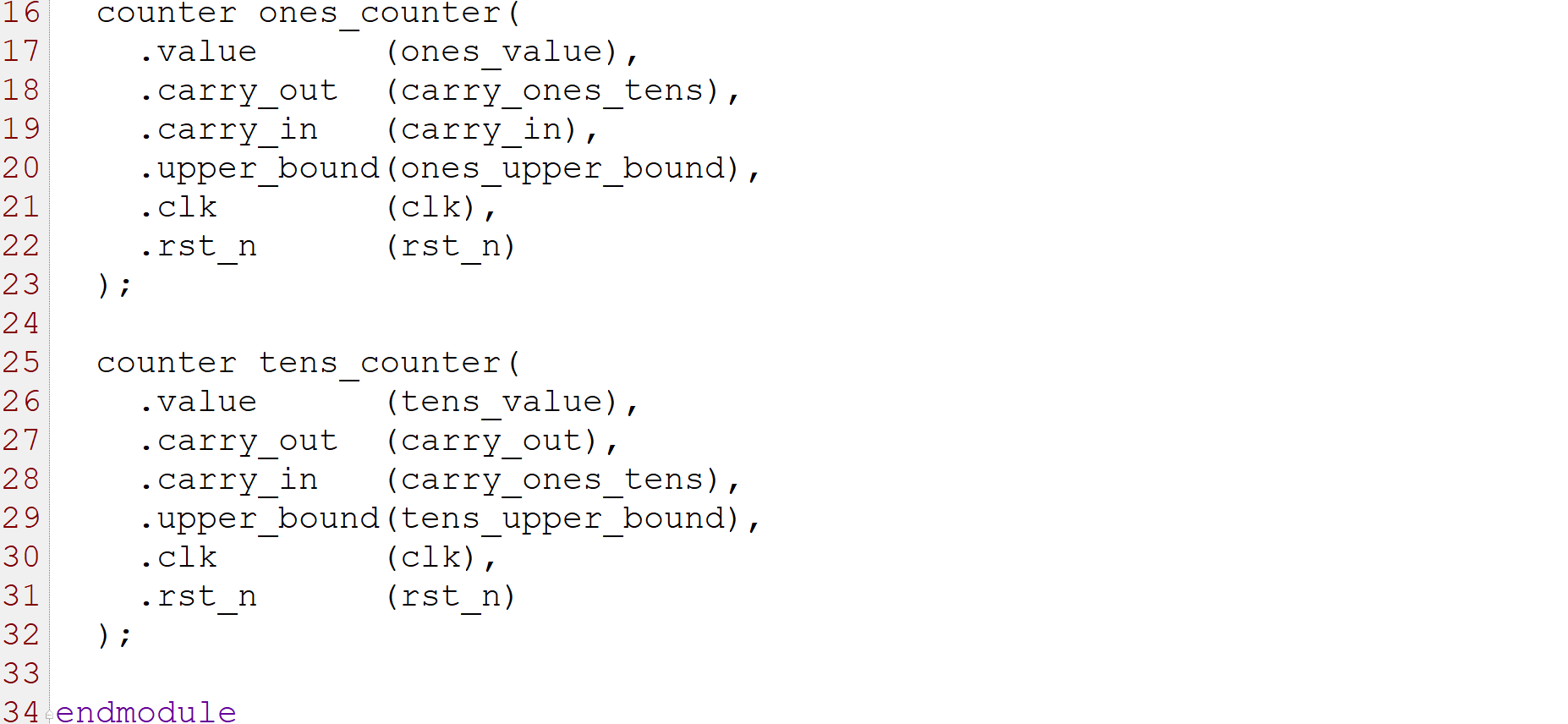
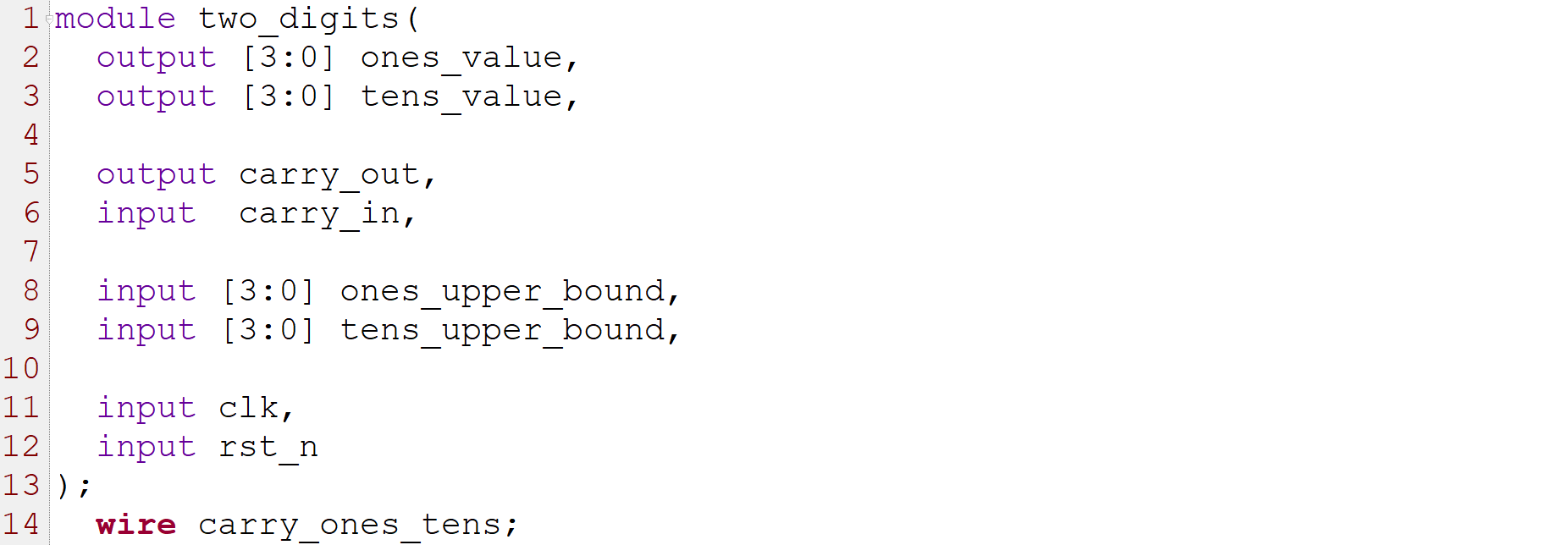


第14到16行是下一個數值的判定。第18到20行是D Flip-flop，用來更新數值。第22行判定是否需要通知上一位進位。

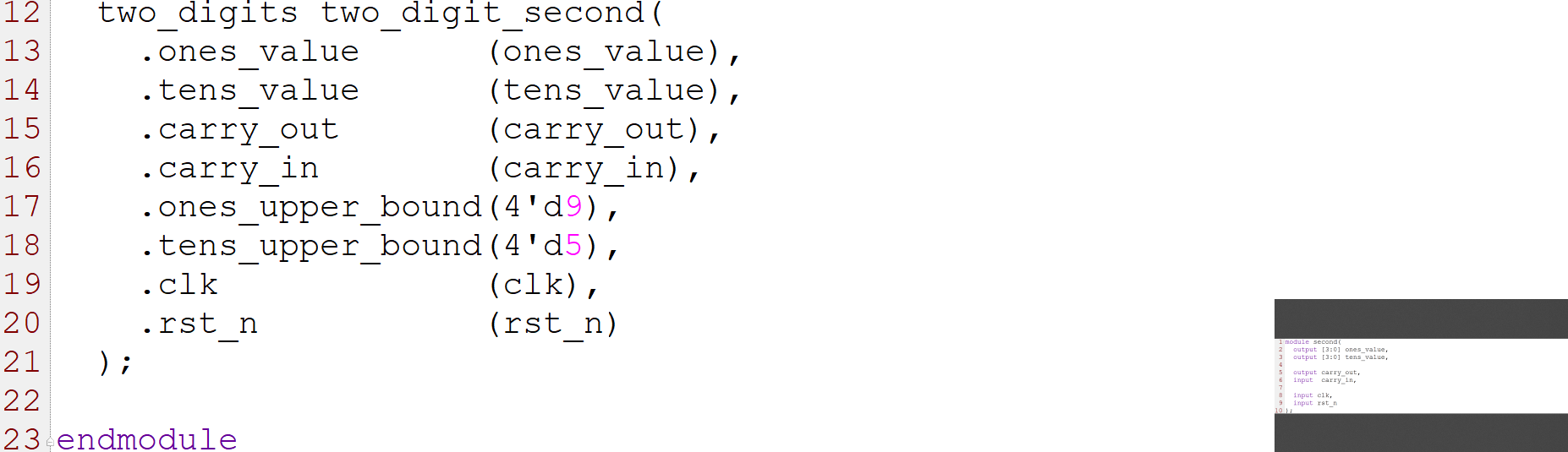


這樣的一個四位元計數器可以BCD表示一個十進位數字，如果要表示小時、分鐘、秒，就必須要有六個四位元計數器。

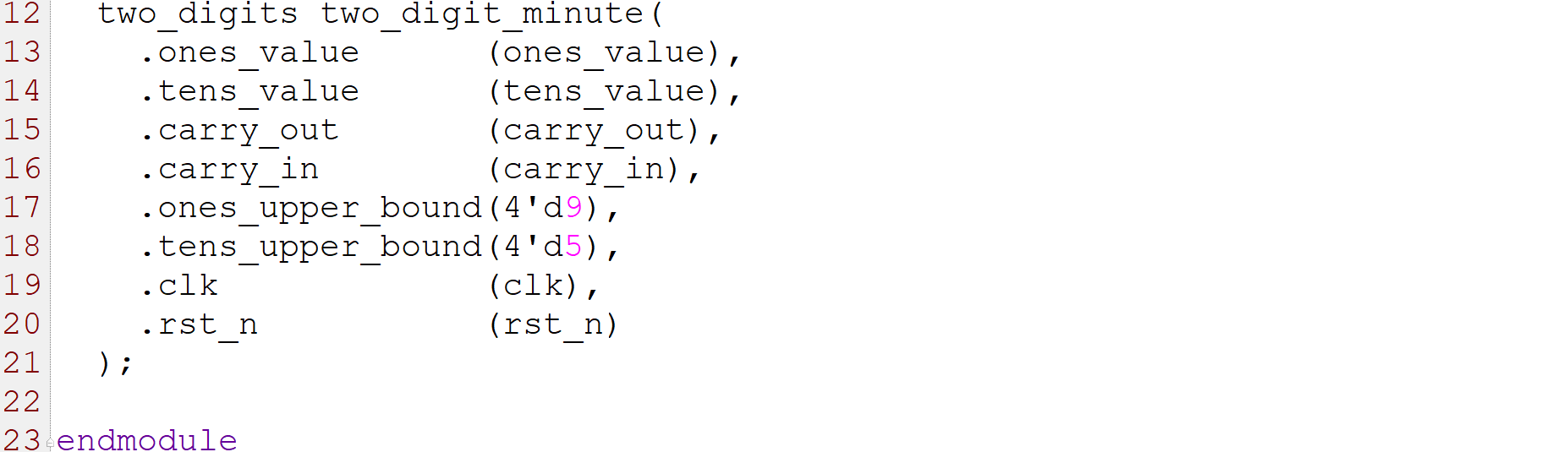
可以把兩個四位元計數器包裝在同一個模組裡面，因為小時、分鐘、秒鐘都各由兩個十位數組成。把個位數的carry\_out連接到十位數的carry\_in，這樣個位數要進位時，十位數就會加一。



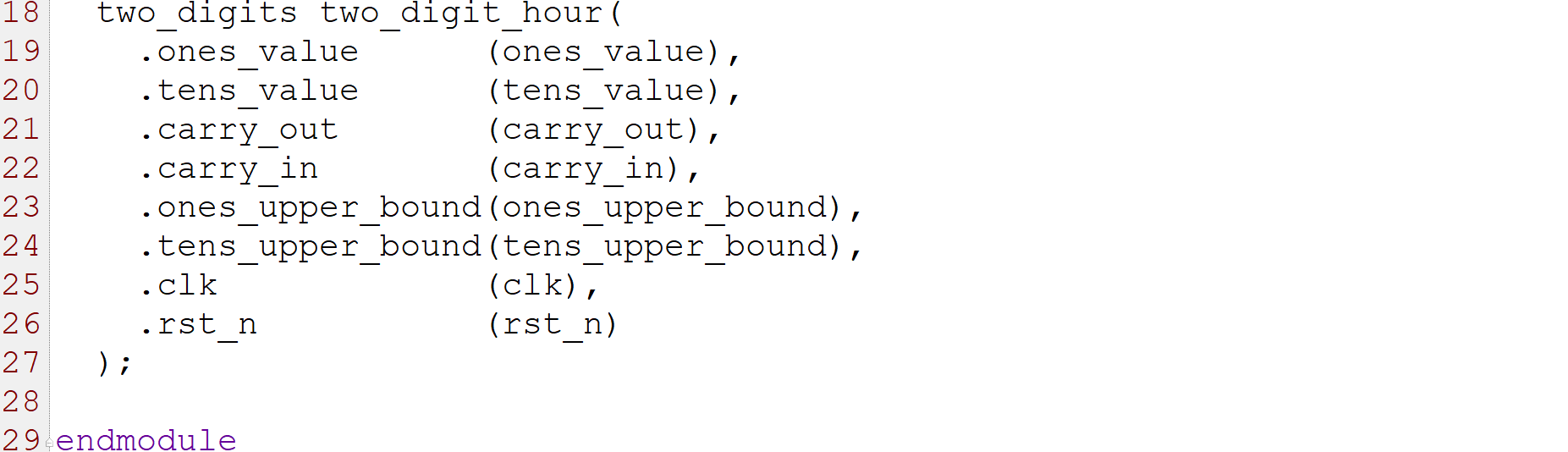
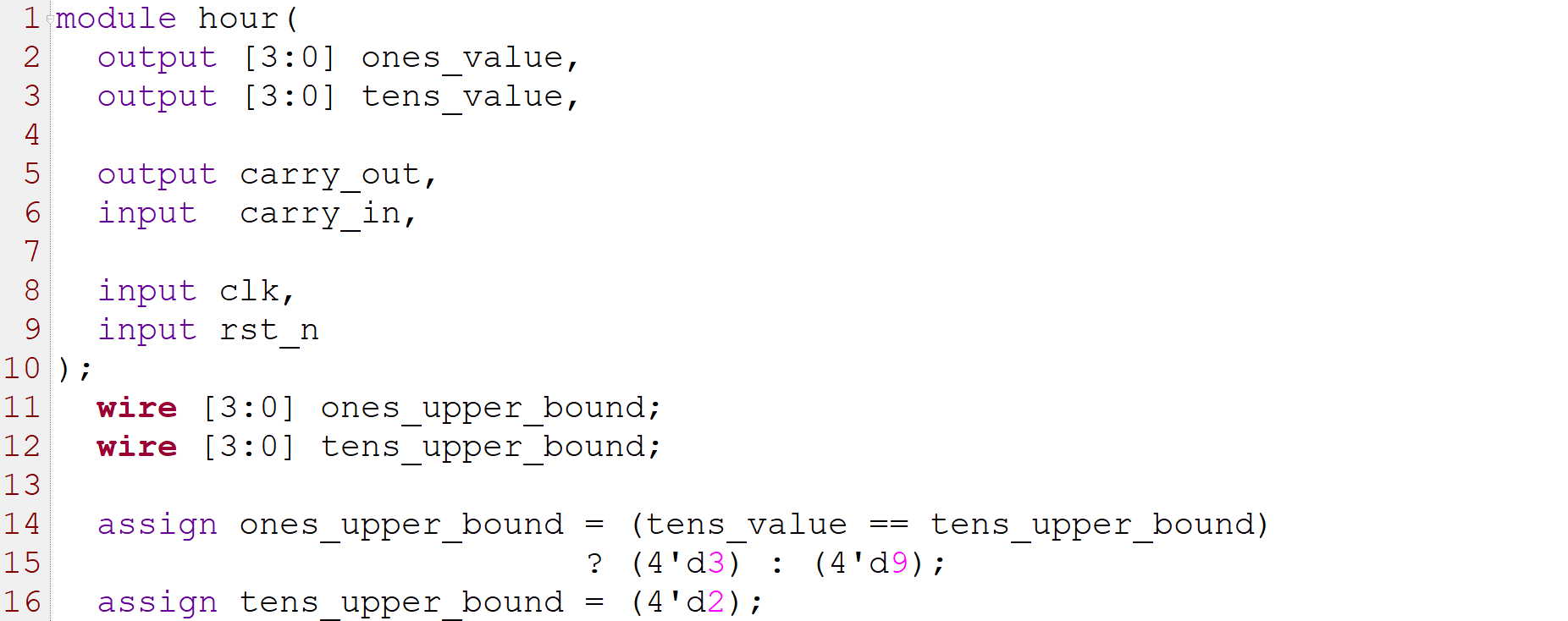
小時、分鐘、秒鐘都各自包裝成模組。這樣要使用時，外界只需要向內部連接時脈和進位訊號就好，避免在一個模組裡同時有太多接線。秒鐘的個位數上界是9，十位數是5。



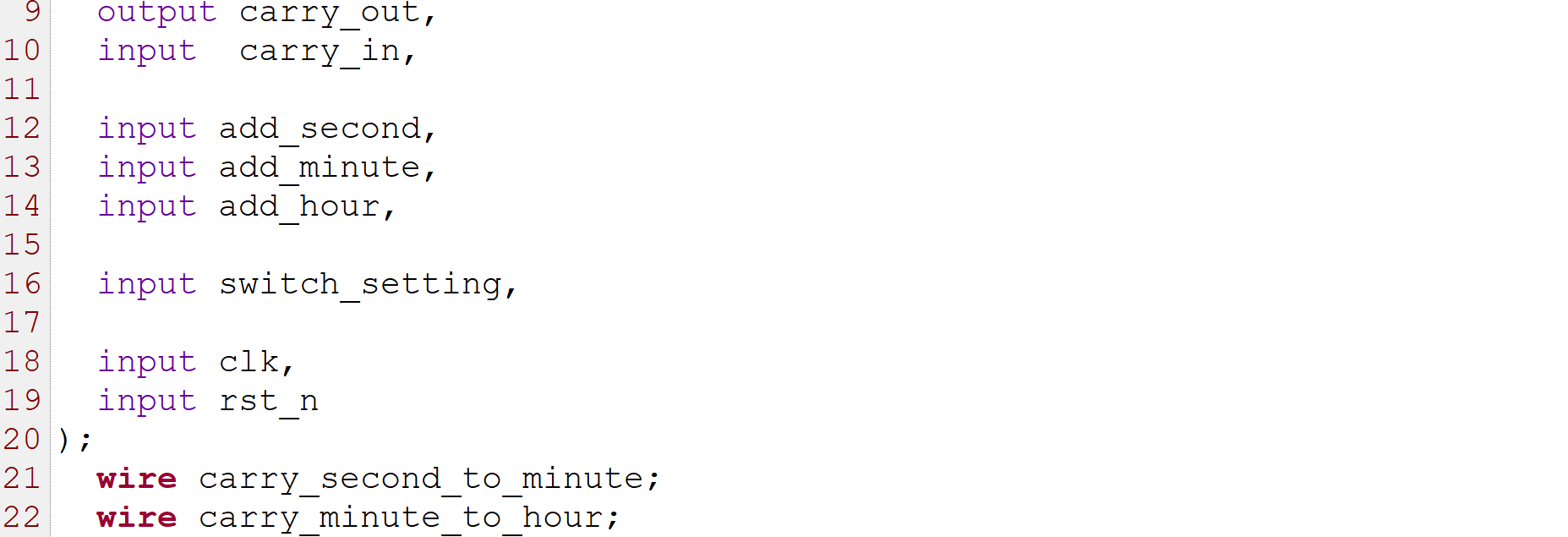
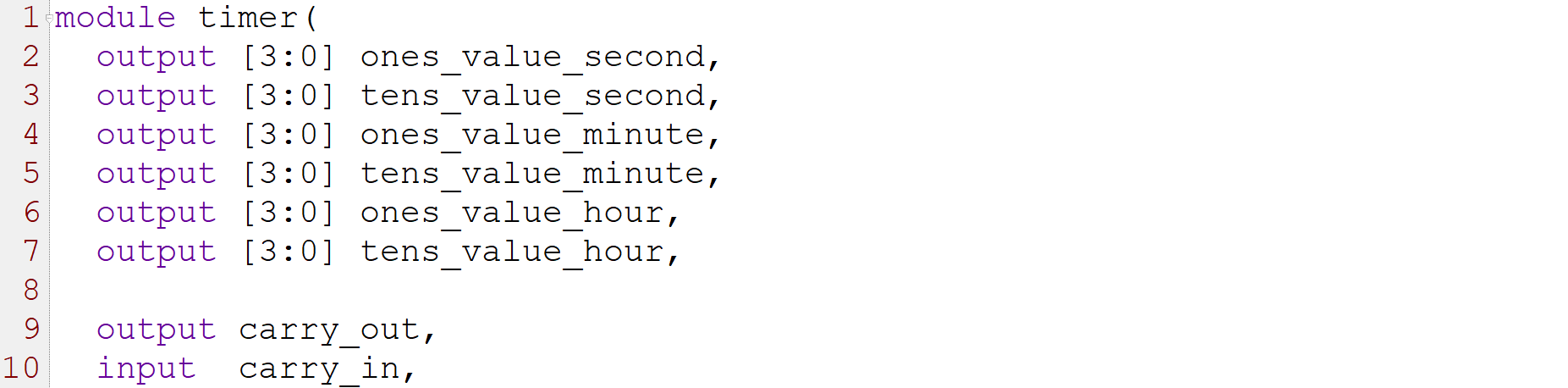
分鐘和秒鐘一樣，個位數上界是9，十位數是5。



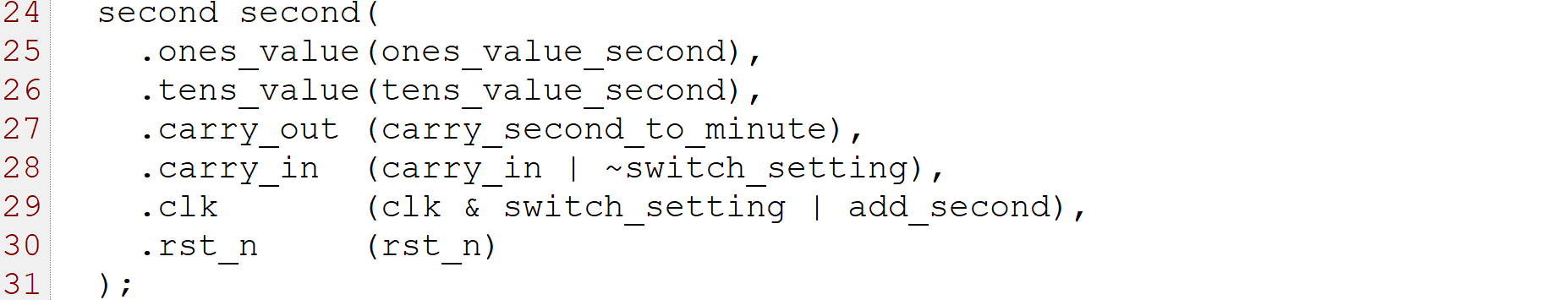
小時的個位數判斷較複雜。第14到15行是小時的各位數上界判定，如果小時的十位數是0或1，那麼個位數上界就是9，如果十位數是2，個位數上界為3。十位數上界則固定為2。

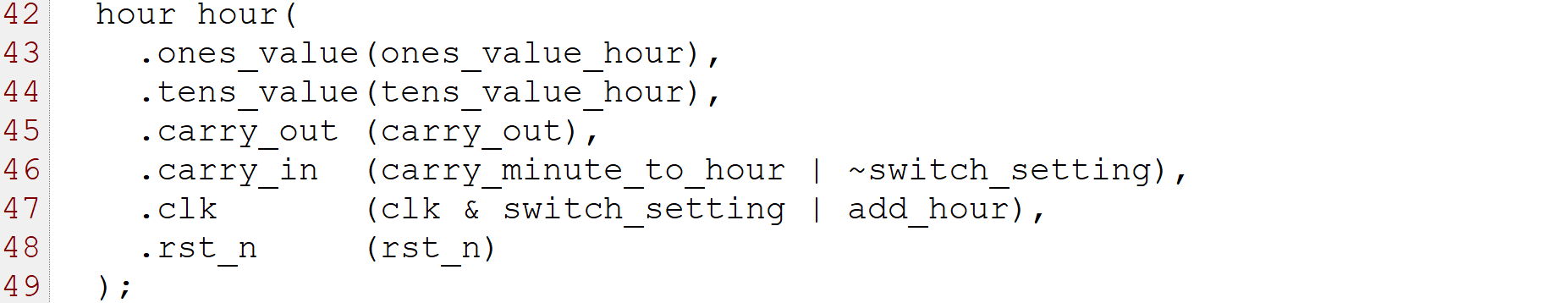
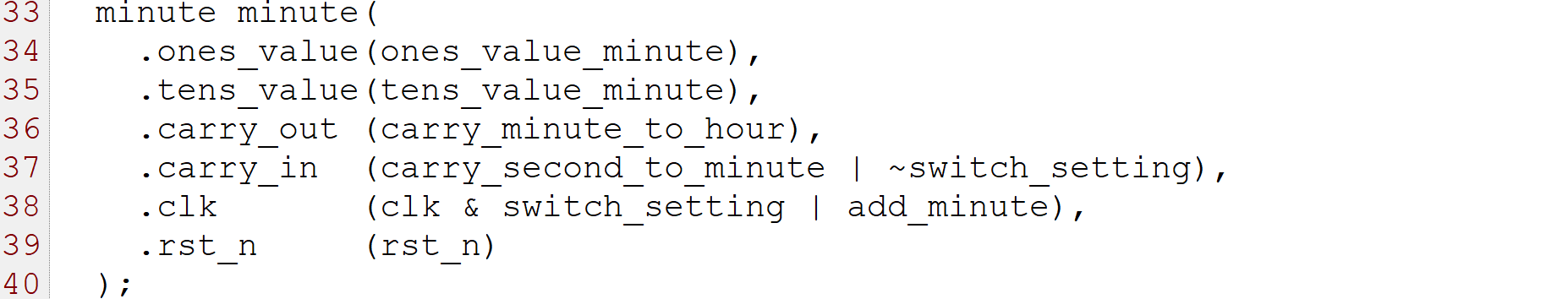


小時、分鐘、秒鐘的模組都已經建構完畢。接下來就可以把上述三個模組包裝在一起。這個模組的功能主要有二：1.將carry的接線接好。2.判定設定時間的clk接線。



收到switch\_setting的訊號時，要設定時間，為了讓設定時間的過程更順暢，每個clk都要連同switch\_setting一起考慮，阻止一赫茲的時脈與小時、分鐘、秒鐘直接連接，並使得小時、分鐘、秒鐘只有在對應的按鈕被按下時才加一。進位的carry\_in也要同switch\_setting一起考慮，在設定模式中，暫時阻止小時和分鐘以及分鐘和秒鐘之間進位，且使進位訊號恆為1’b1，這樣按鈕被按下時，對應的小時、分鐘、秒鐘才會加一。

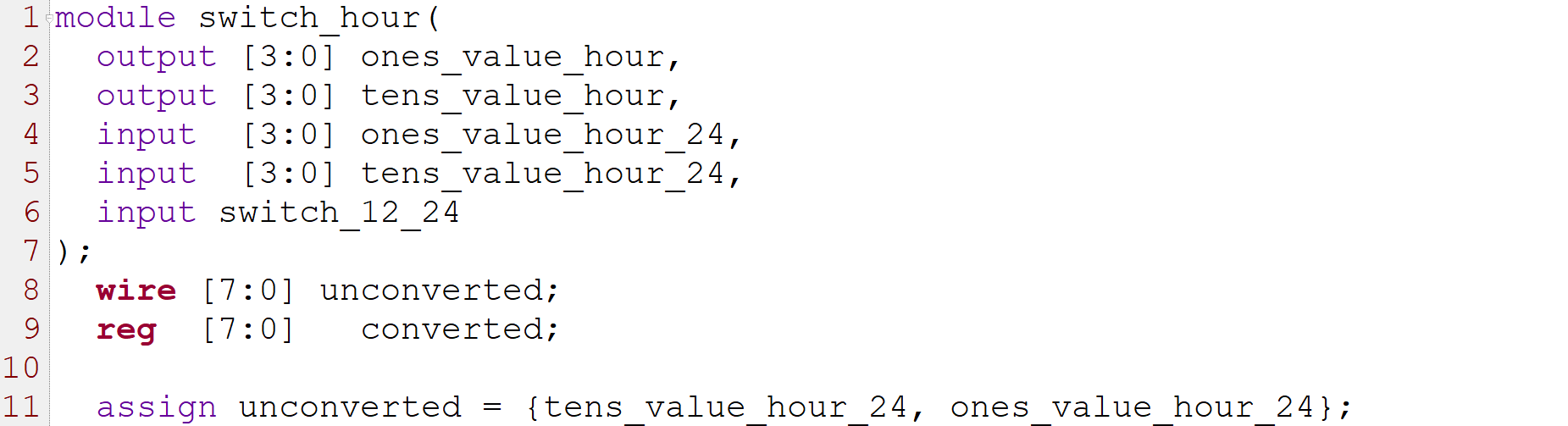




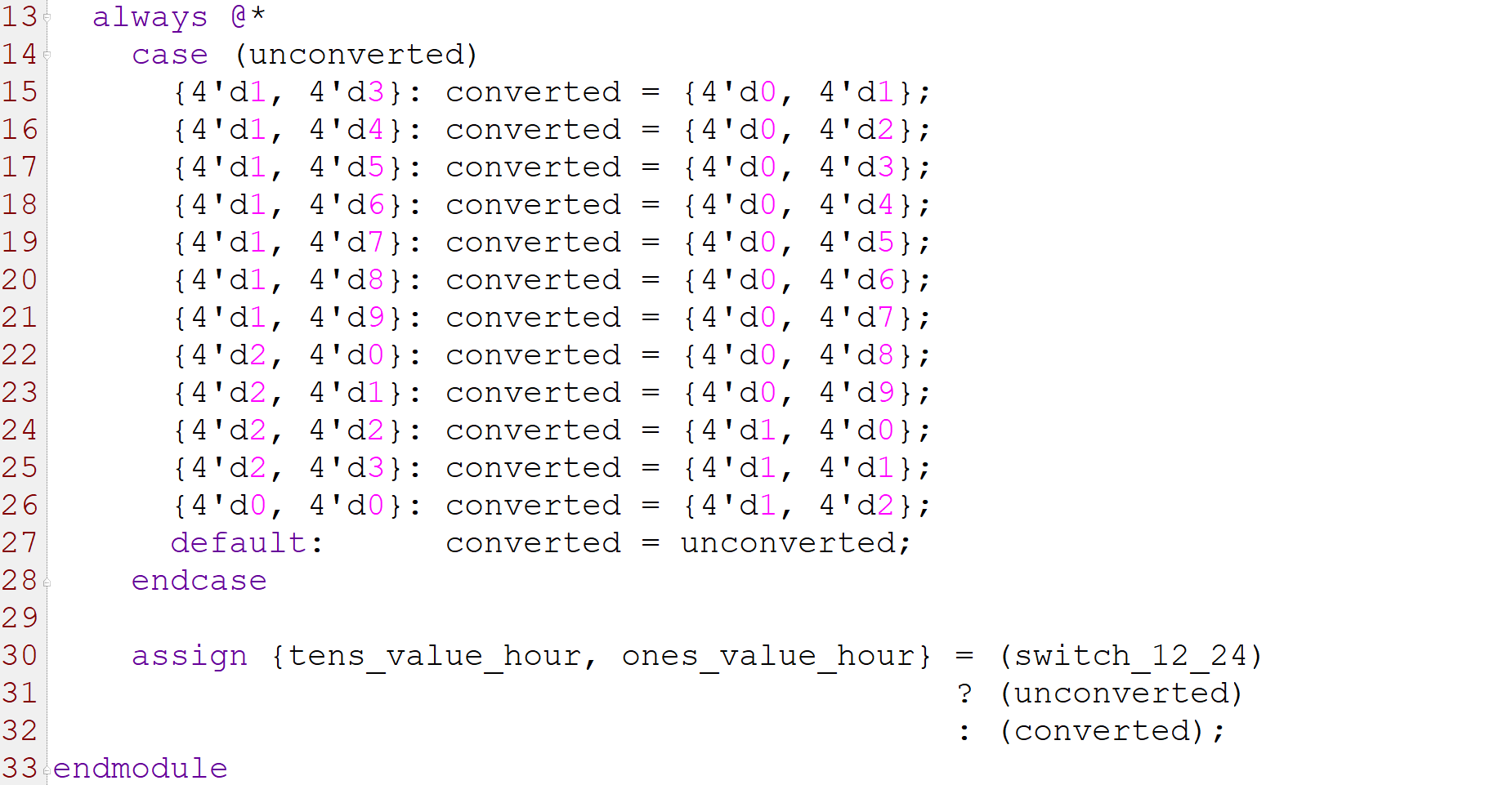
把上述的判定訊號與小時、分鐘、秒鐘的模組連接好，這樣timer就基本完成了。

至此，我們已處理完計時的部分。只剩下顯示的部分還沒建構。

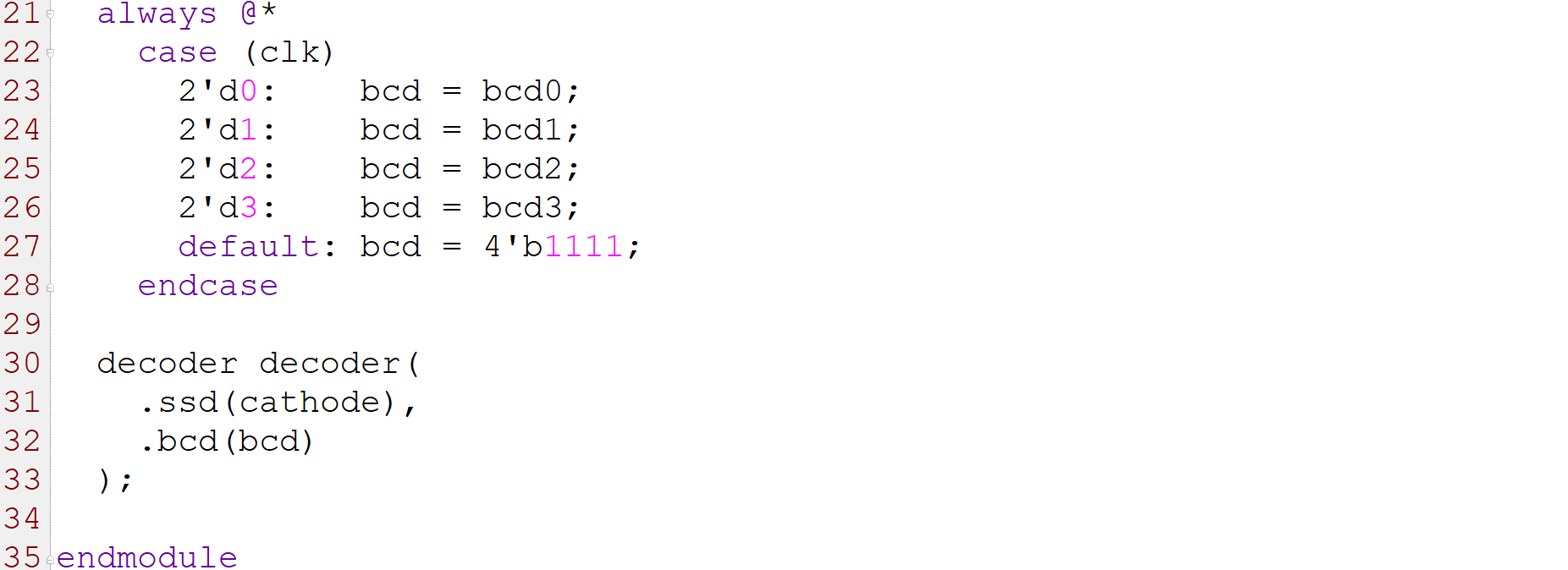
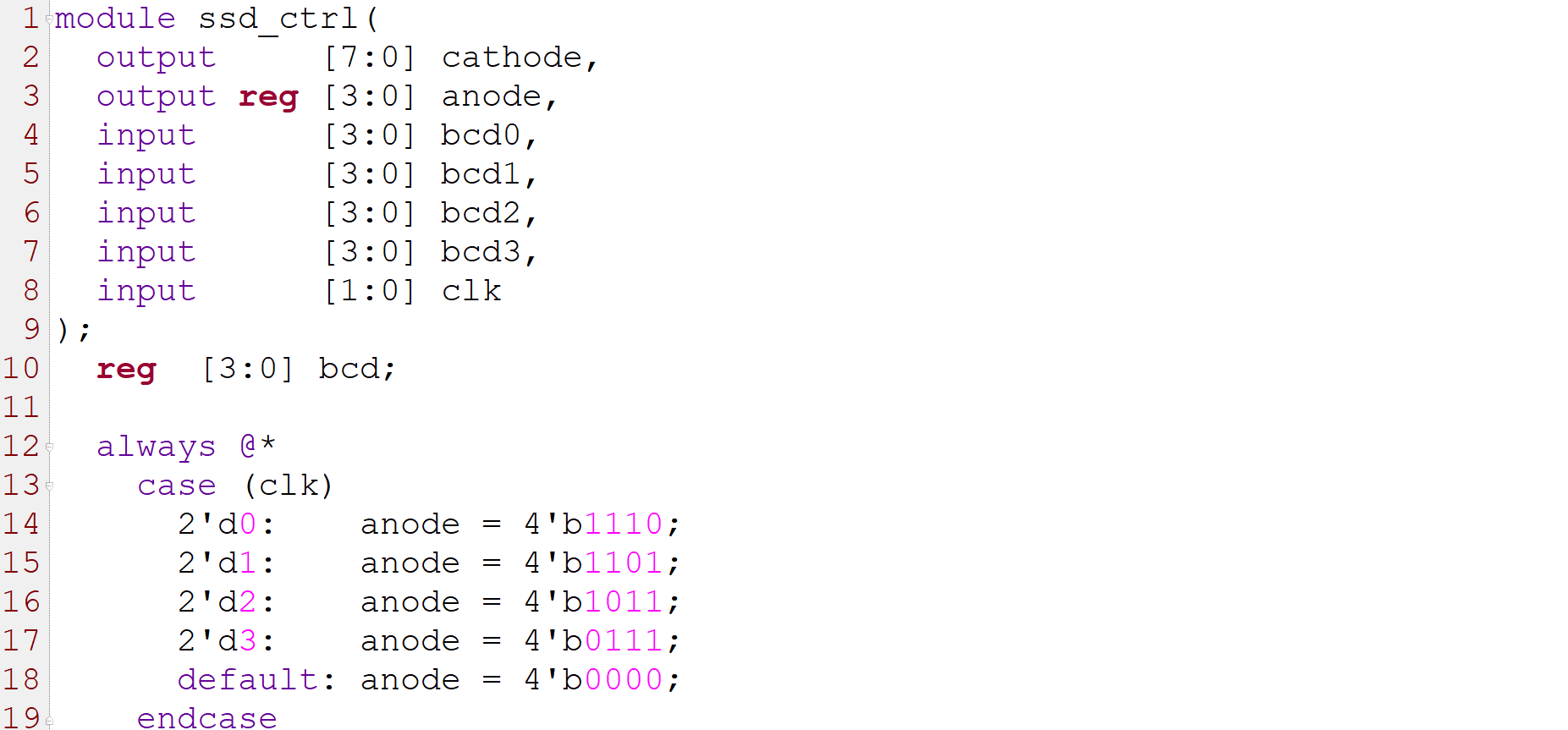
顯示小時的模式有兩個，一個是二十四小時制（不用轉換），一個是十二小時（超過十二的話要減十二）。



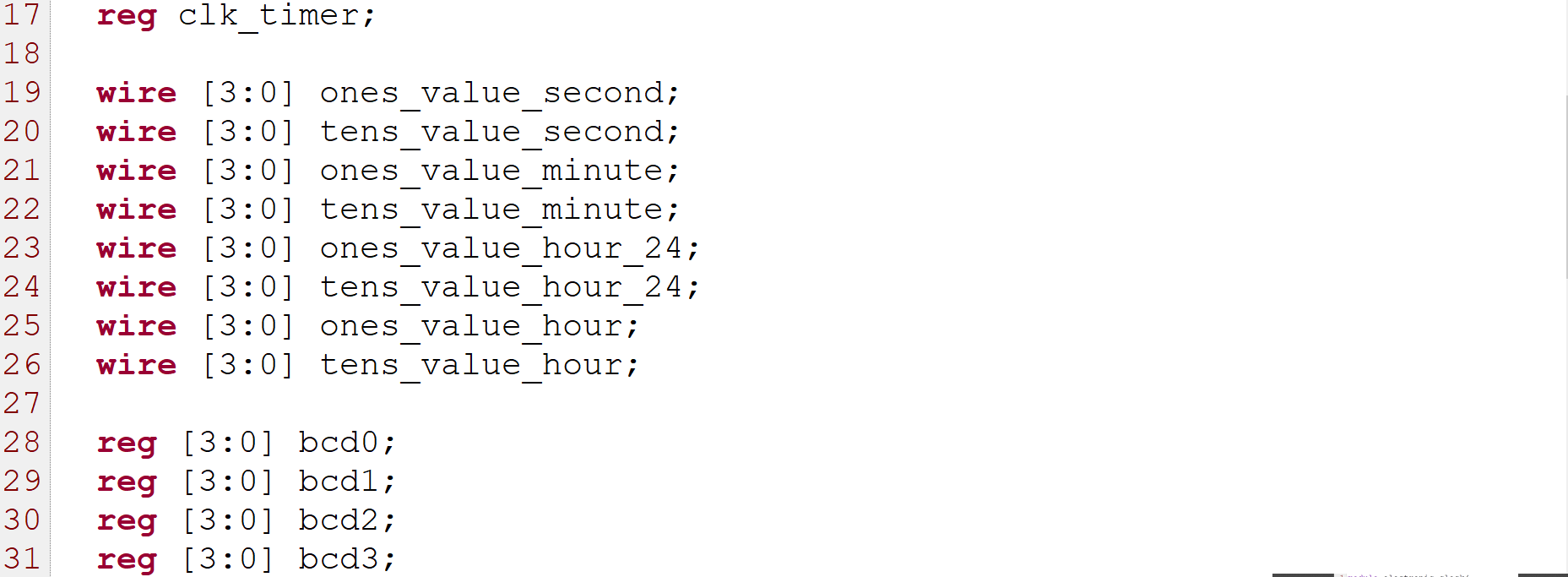
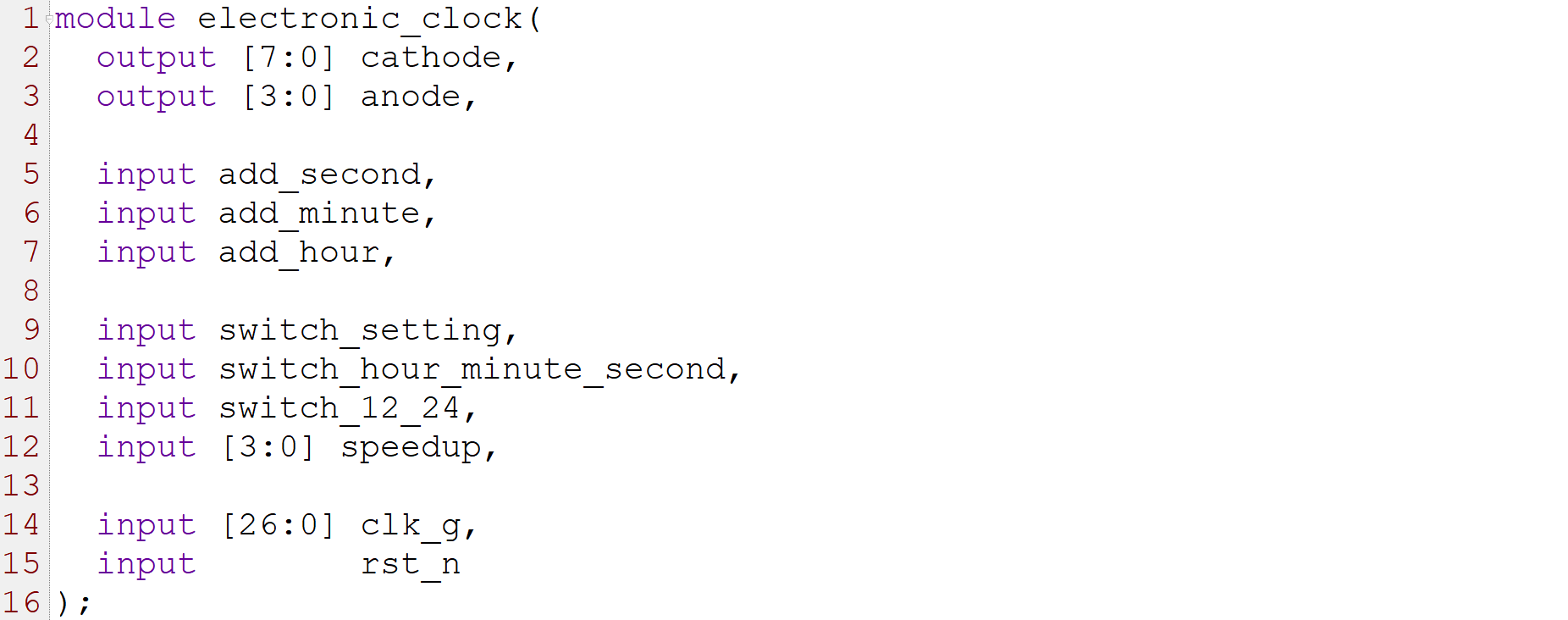
case不多，直接列出來，用類似解碼器的方法即可。第30到32行判斷當下是否要轉換。



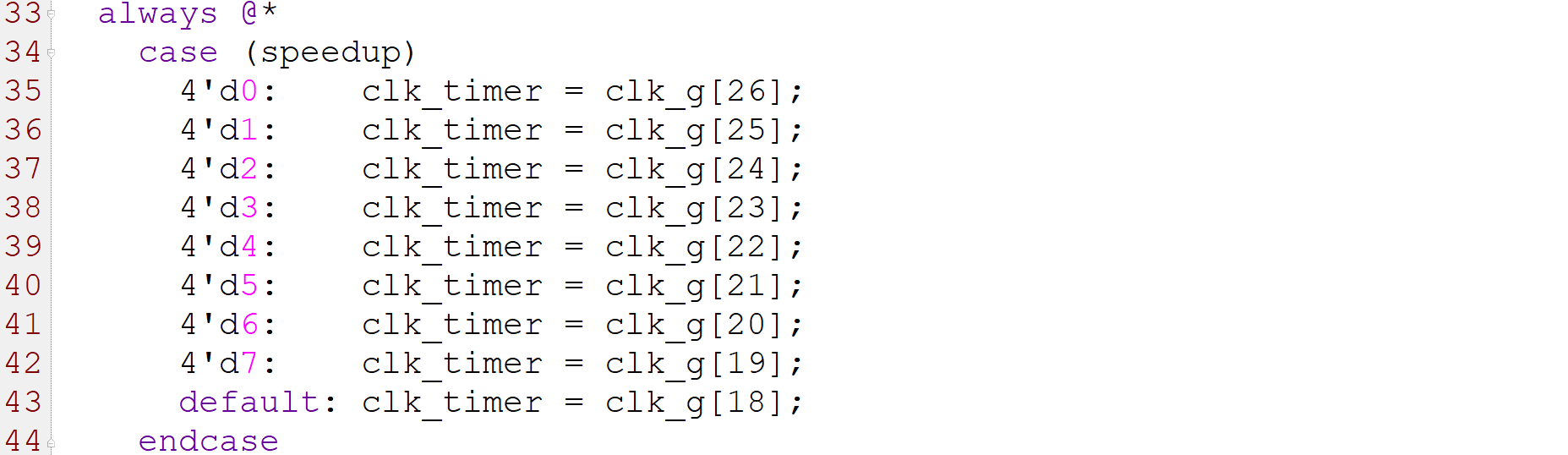
至此，已經有了所有時間的BCD訊號。接下來要把這些BCD訊號轉成七段顯示器的樣式，且利用視覺暫留來使七段顯示器看起來可以同時顯示四個不同的數字。其中第30到33行的decoder的內容跟上次的LAB一模一樣。



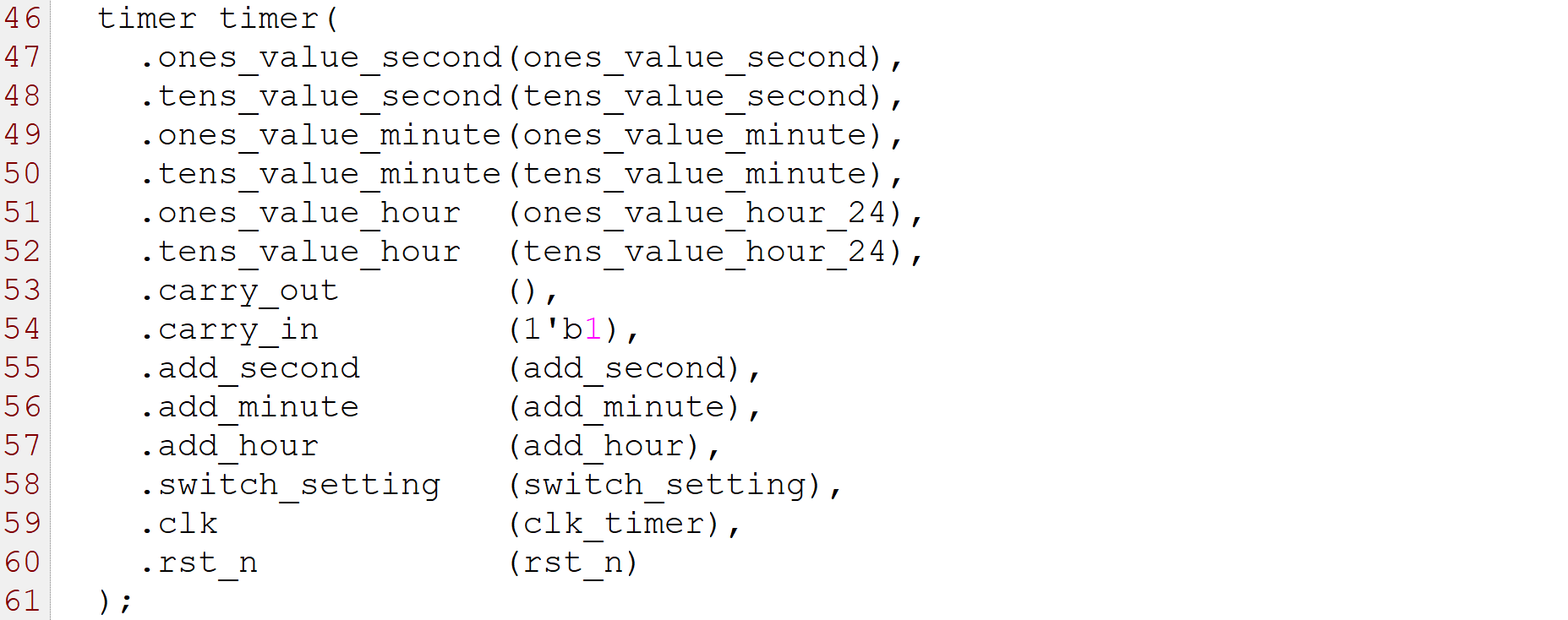
上述的三個模組（timers、witch\_hour、ssd\_ctrl）可以包裝在一個模組裡面，避免同一層級內的接線太多。



為了demo方便，我設置了一些加速的選項。平時連接到timer的時脈是一赫茲的，如果要加速的話，只要使用MUX改將頻率更快的時脈傳送到timer即可加速。

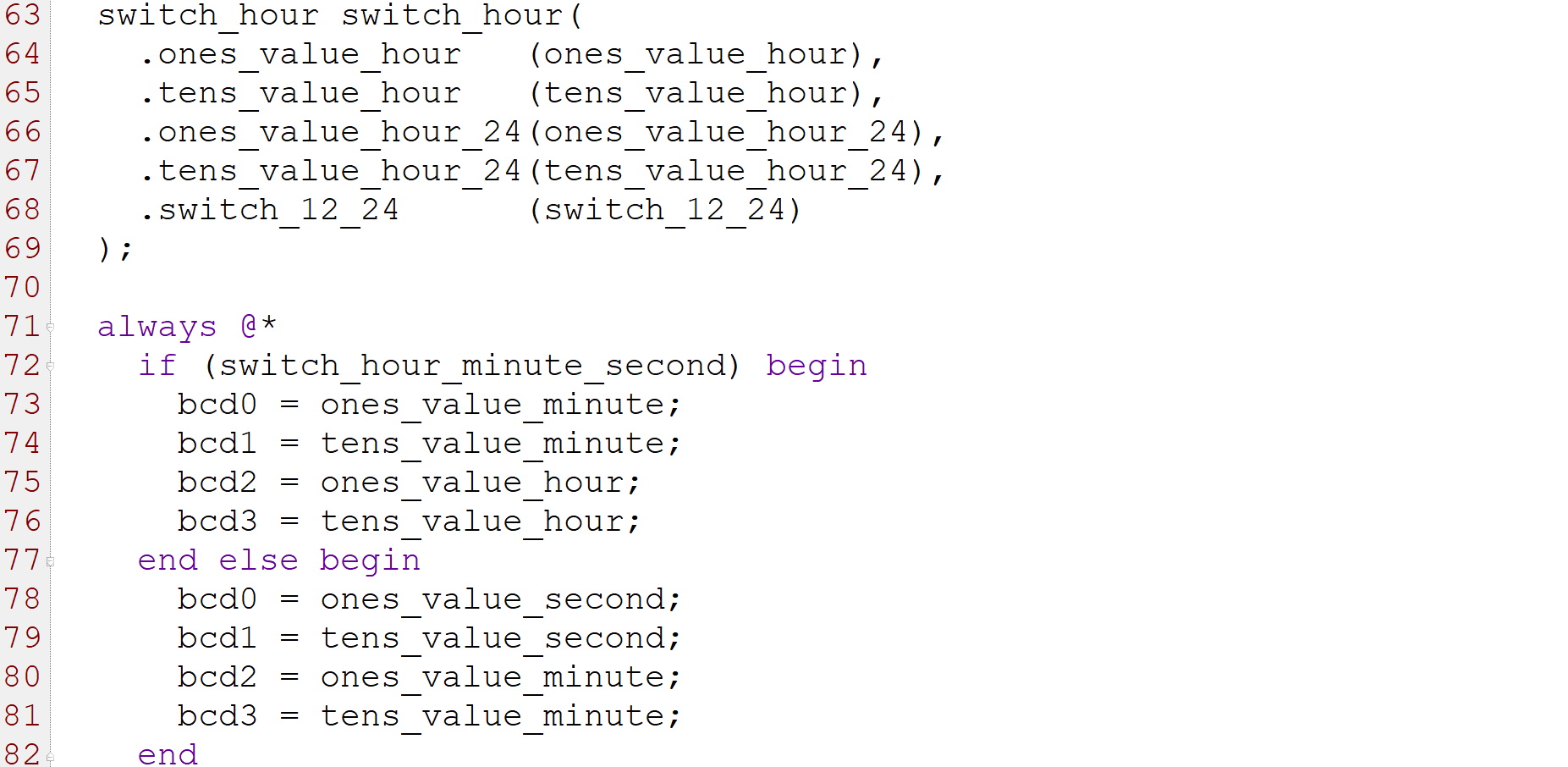


timer的接腳有些是不會用到的，例如carry\_out。而carry\_in恆為1’b1，這樣只要有clk\_timer或要add\_second、add\_minute或add\_hour，tiemr裡面的計數器都會判斷是否需要加一或進位。



其中，timer的hour的數值是24小時制，必須要先通過switch\_hour判斷當下的switch\_12\_24，決定是否需要減十二，不需要的話才直接將數值傳送到ssd\_crtl。

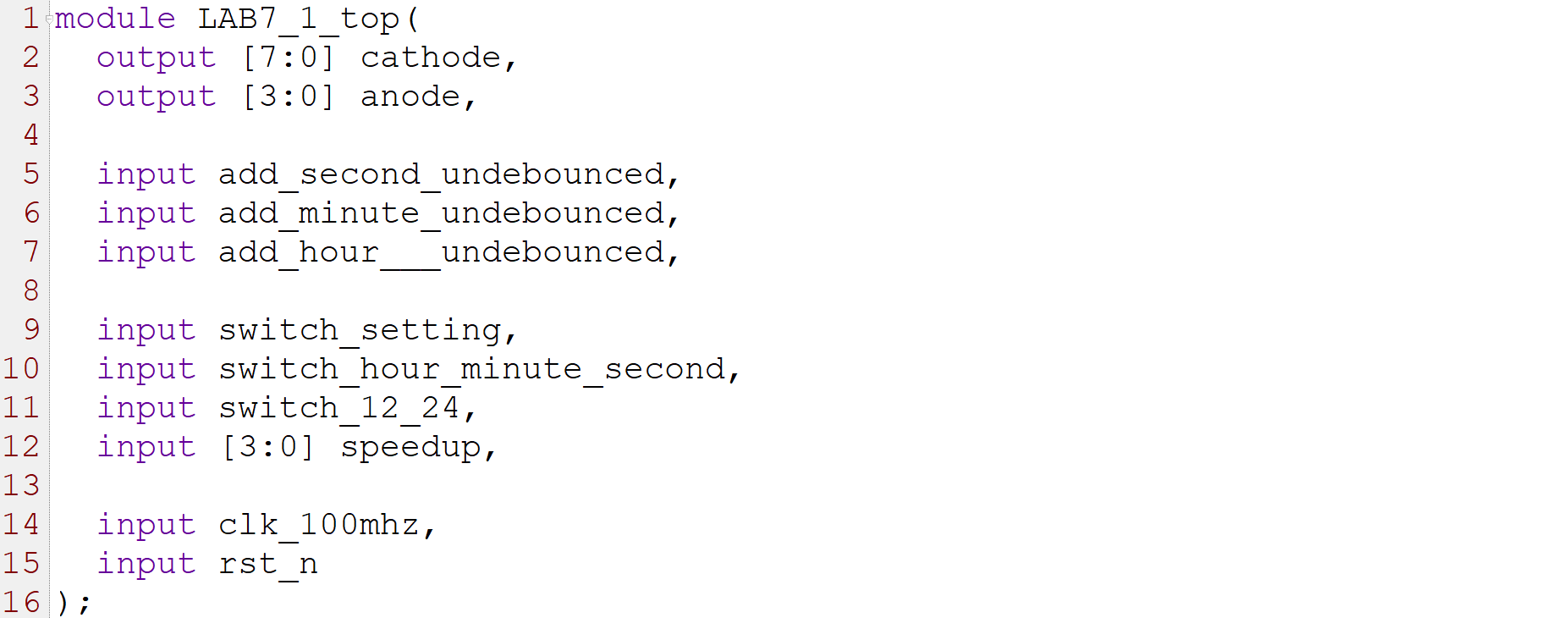
判斷是否需要轉換成十二小時制之後，還要檢查switch\_hour\_minute\_second，以決定要顯示小時跟分鐘，還是顯示分鐘跟秒。



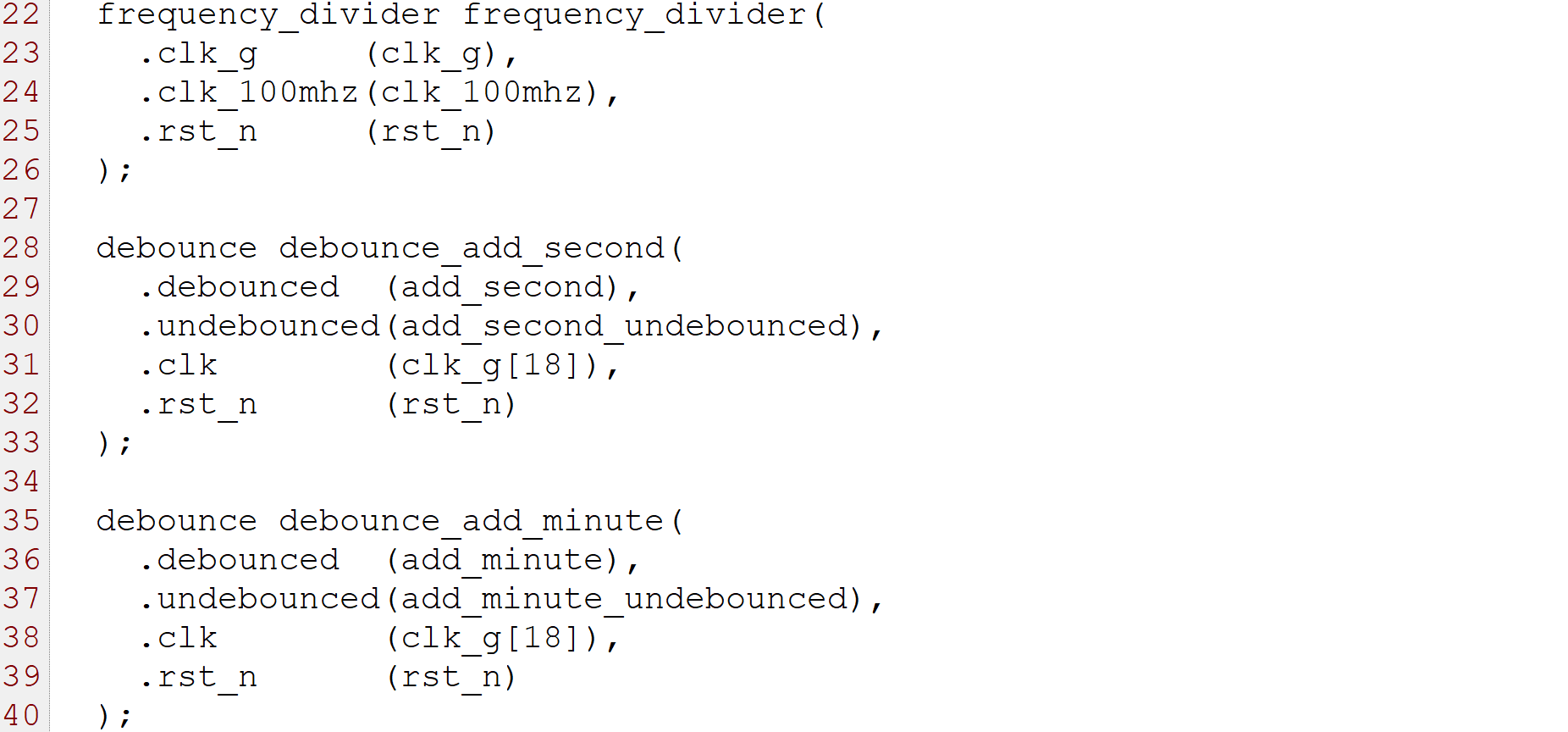
最後再把訊號傳送到ssd\_ctrl即可。



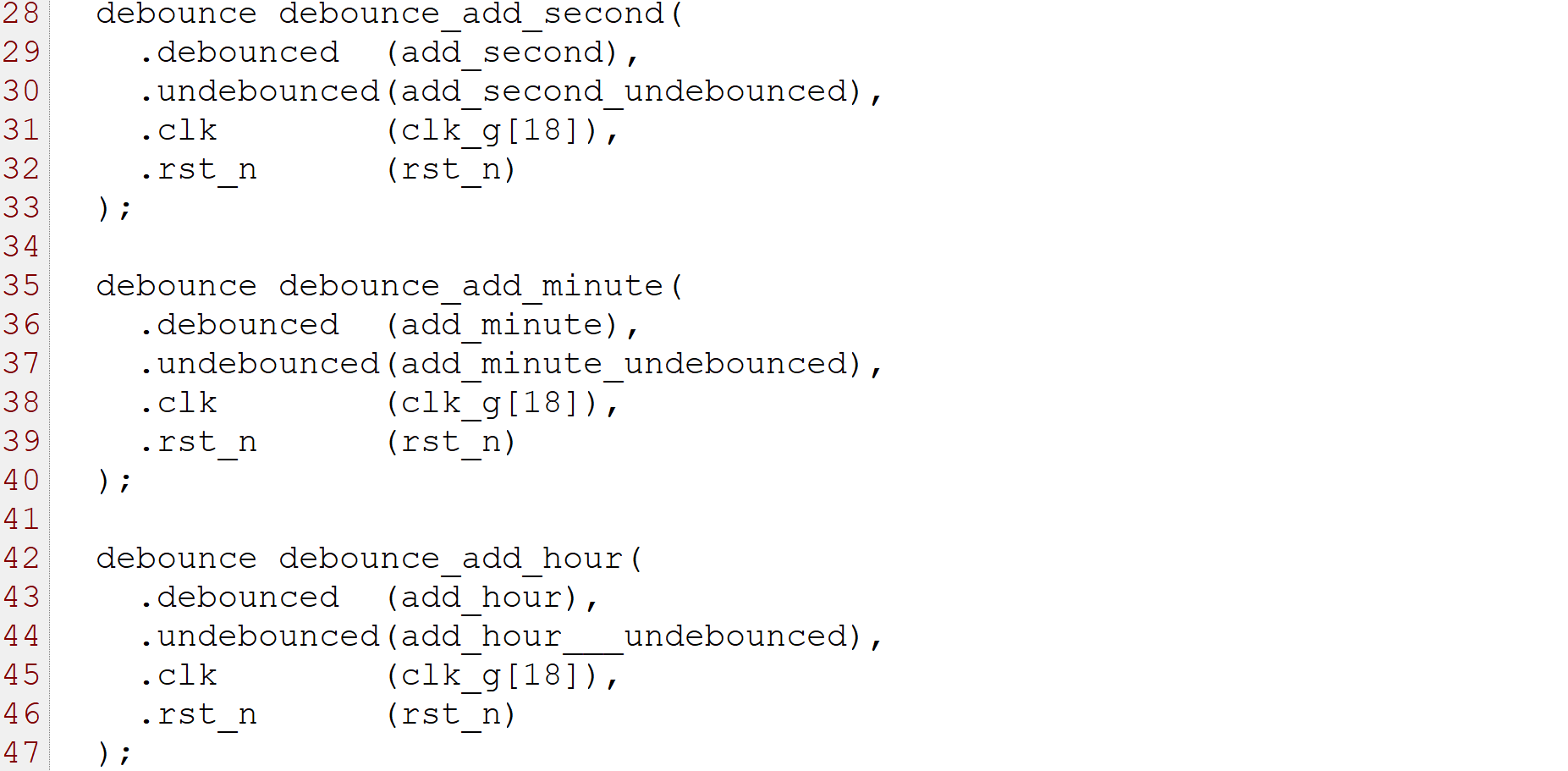
至此，完成第一題的所有模組都已經建構好了，只要再建構出top即可。



由於該建構的都幾乎建構完畢了，因此這題的top很單純。先除頻。



除出來的頻率用來debounce除了clk\_100mhz之外的所有input訊號。



把debounce過的訊號連接到剛剛建構的electronic\_clock，這樣就大功告成了。



**5. Conclusion**

**內容: 可以寫下你的這個lab的想法、遇到的問題、解決方法、心得等等，請自由發揮。**

有了上次LAB的經驗，我這題按部就班的一個一個模組慢慢寫。果然，雖然一開始的進度緩慢，但好處是，最後完全不用debug，第一次generate bitstream測試就成功！比較困難的是下一題，我原本試著沿用本題辛辛苦苦寫好的模組，但卻因為月份和日期上下界的複雜度超乎我所預料，所以只好全部重寫。

**2 For the date functions in clock (no leap year), we have the following functions:**

**o Day (Jan/March/May/July/Aug/Oct/Dec: 1-31,**

**Feb: 28, Apr/June/Sept/Nov: 30),**

**o Month (1-12),**

**o 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).**

**1. Specification**

**內容: 寫下你的電路中的inputs, outputs以及其bit widths，名稱必須跟你的verilog code中相同。**

module LAB7\_1\_top(

output [7:0] cathode, // 七段顯示器

output [3:0] anode, // 七段顯示器

input undebounced\_add\_day, // 按一下加一天

input undebounced\_add\_month, // 按一下加一個月

input undebounced\_add\_year, // 按一下加一年

input undebounced\_switch\_setting, // 切換設定模式

input undebounced\_switch\_day\_month\_year, // 切換顯示年月日

input [3:0] undebounced\_speedup, // demo時加速用

input clk\_100mhz, // 內建的時脈

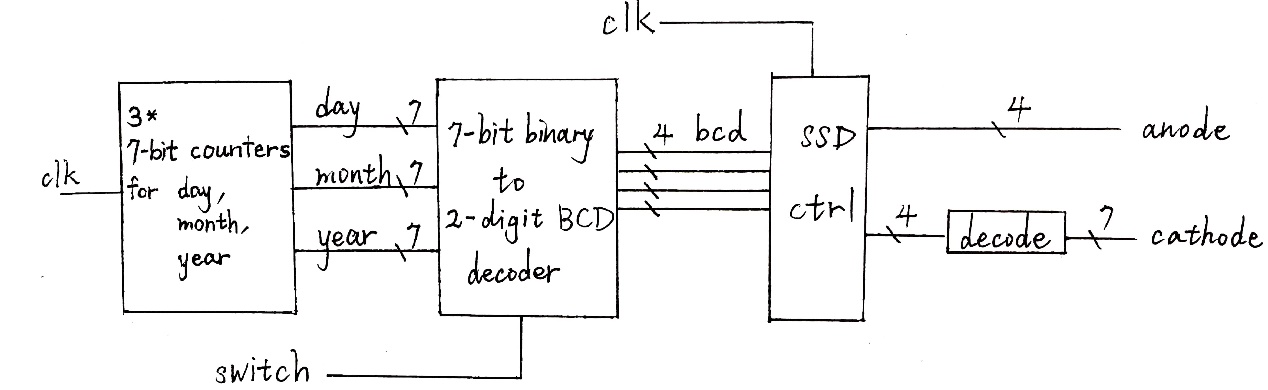
input rst\_n // 重置、初始化

);

**2. Block Diagram**

**內容: 電路中的Block diagram(可以用手畫拍照或電腦繪圖)。**

結構跟上一題一樣，只差在進位判斷和上界判斷的邏輯。但因為月份跟日期的上界判斷較複雜，所以本題捨棄以往一個位數用一個計數器的方式，而改成一個時間單位就使用一個計數器。也就是說，年月日總共只需要用三個計數器，每一個計數器含七個位元。



由計數器產生出來的七位元二進位數字再各自連接到一個七位元二進位轉二位數BCD的解碼器，最後才連接到七段顯示器的控制模組。

**3. Finite state machine**

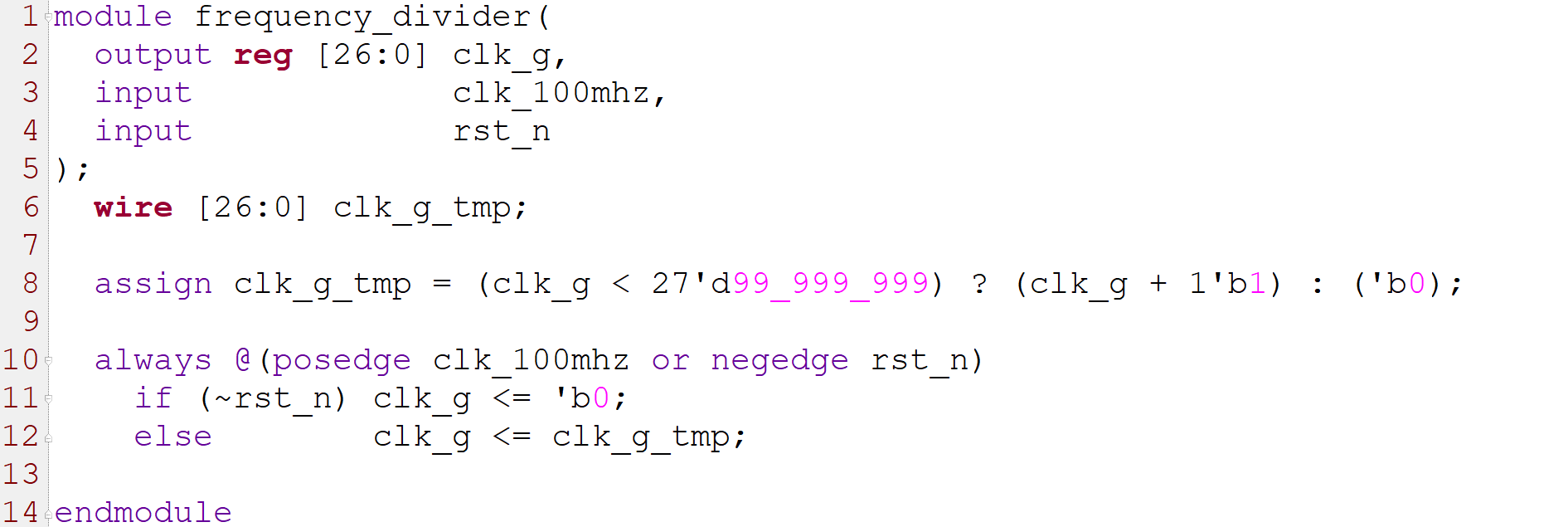
**內容: 電路中的Finite state machine，若無則寫無。**

無

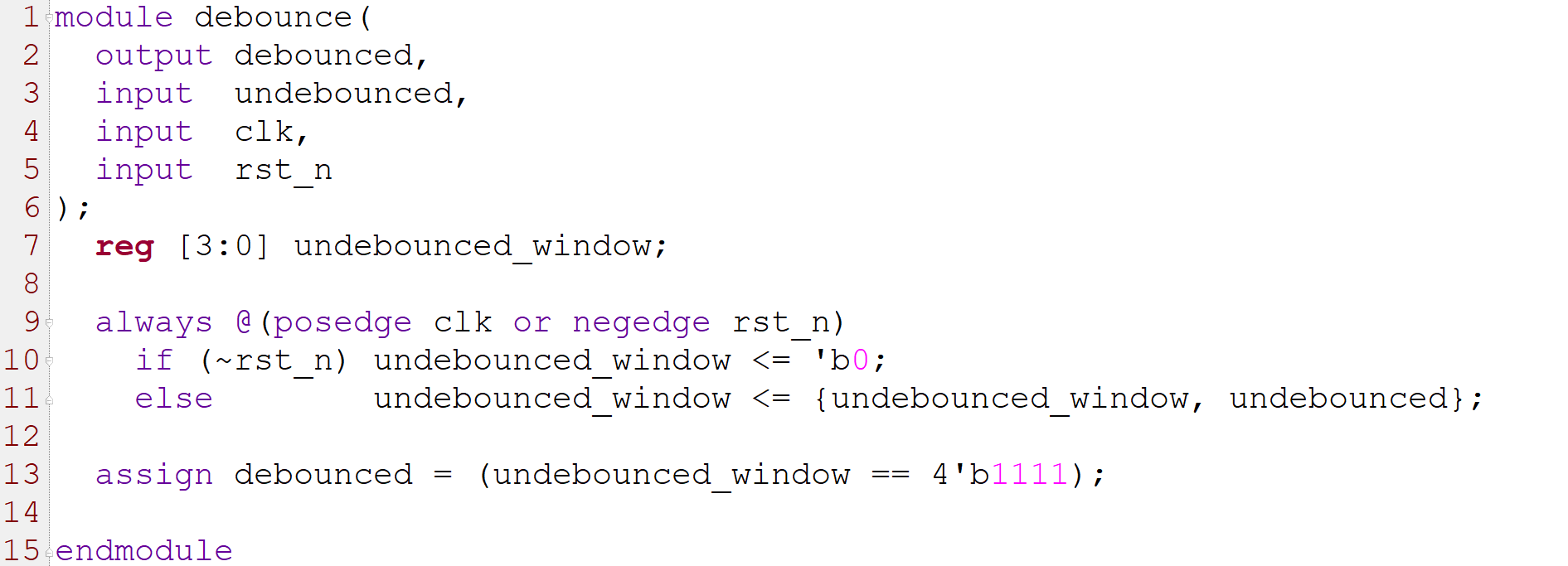
**4. Implement**

**內容: 請列出相關的logic function、詳細用文字解釋電路的運作方法、結果等等，可以貼code解釋或拍FPGA輔助解釋(但不能只貼code跟FPGA結果)。**

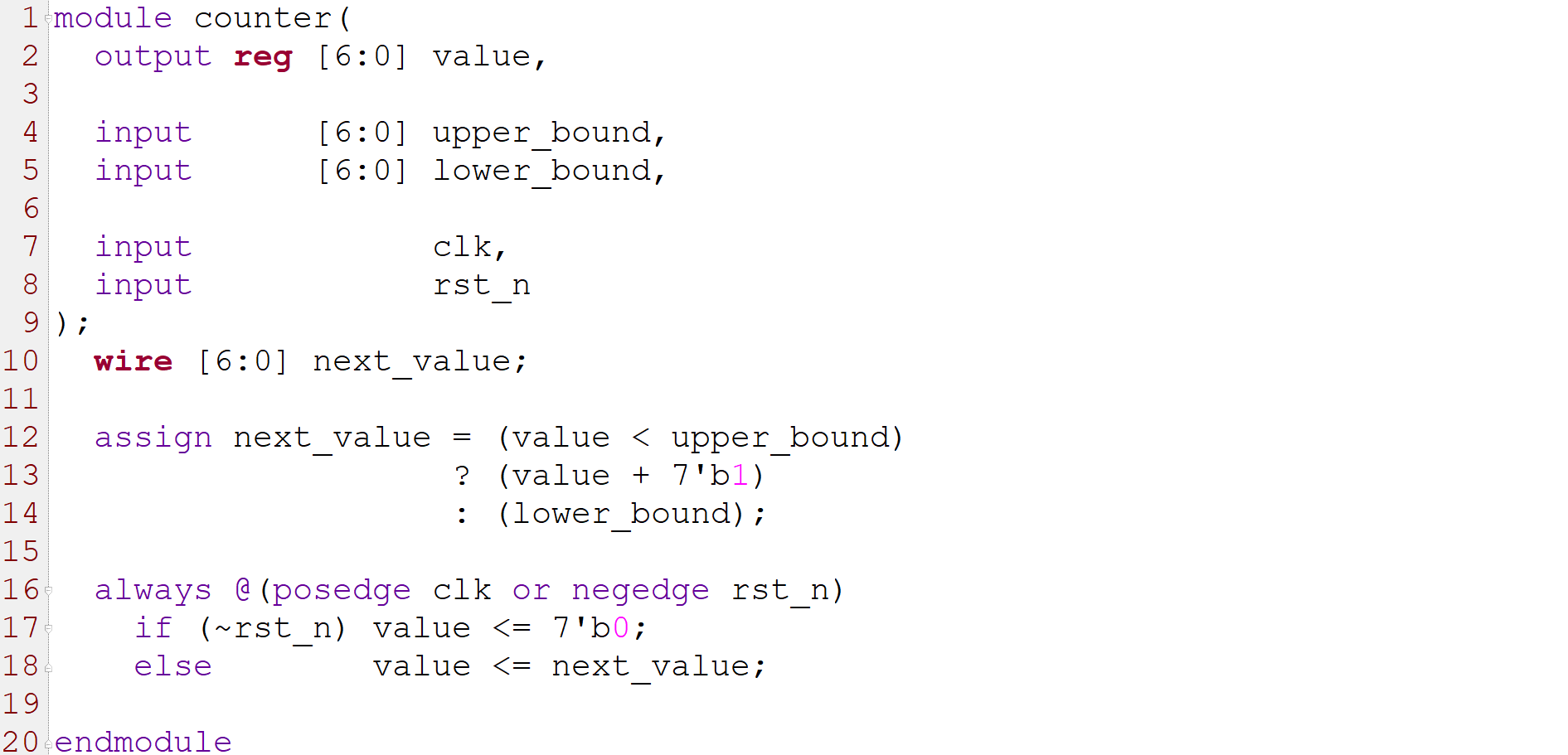
直接把除頻器的計數器當作output，提供其他模組各種頻率的時脈。



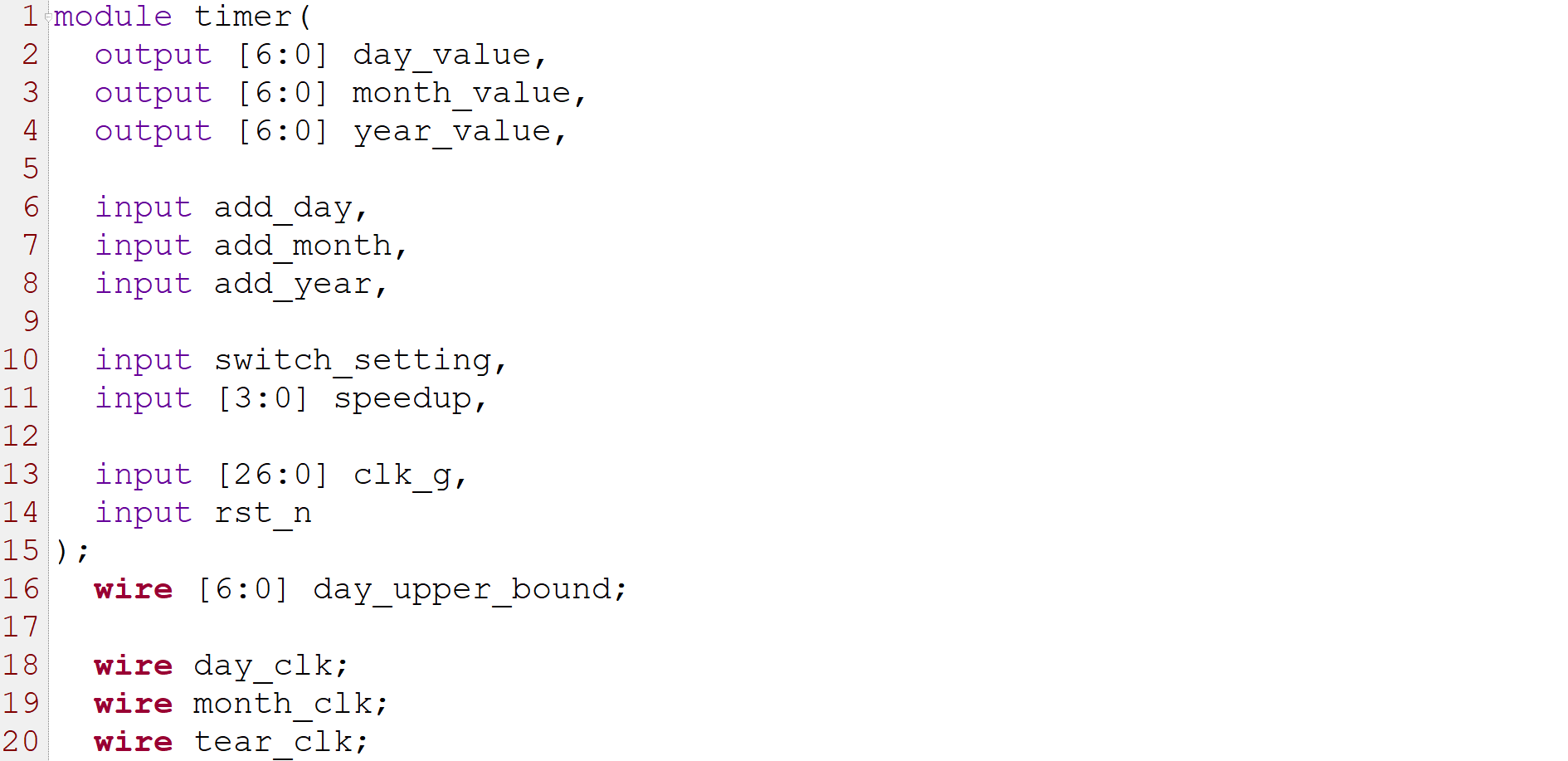
除頻器和debounce的模組都直接沿用上一題的就好，不須修改。

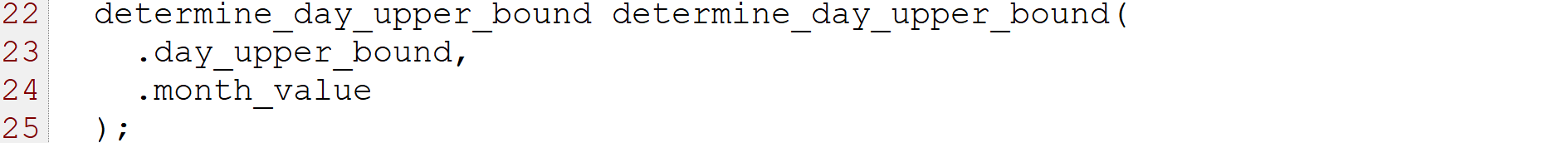


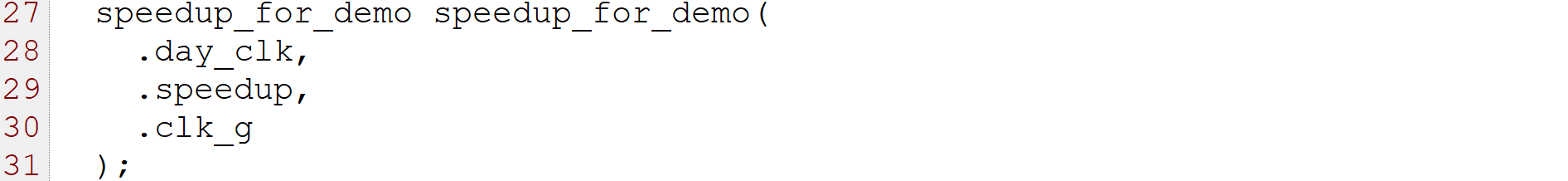
由於年月日的上下界判斷較複雜，如果要把每個時間單位拆開成兩個位數的話，會讓邏輯變得複雜。如果改成直接使用三個七位元的計數器分別記錄年月日的話，就不用各別考慮個位數的上下界了。



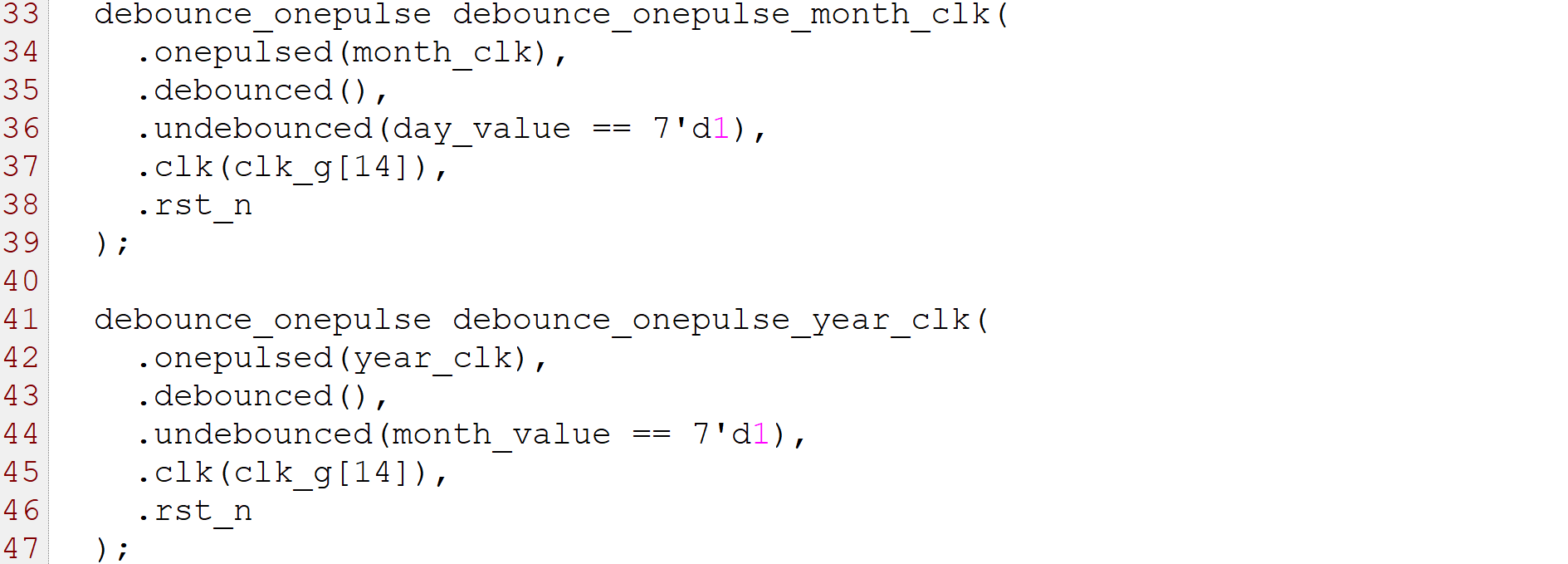
結構跟上一題的四位元計數器一樣，只是新增了下界作為輸入。稍微修改第12到14行的判斷，當數值已經到達上界時，下一個時脈不是歸零，而是變成下界。

把負責年月日的三個七位元計數器包裝在同一個模組裡。

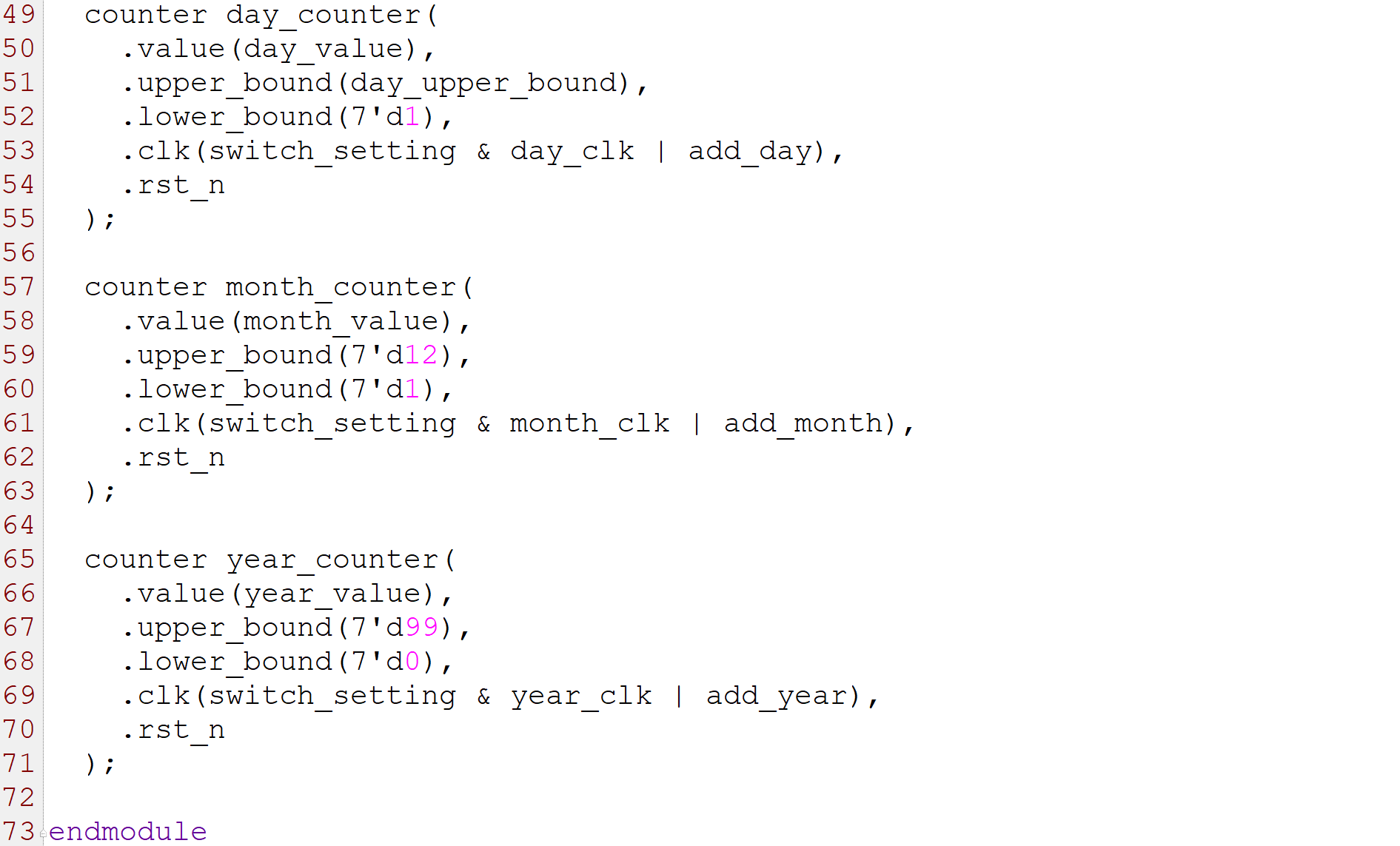
由於有大月跟小月，所以日期的上界要根據所在月份做調整。

而為了demo方便，仿照上一題的模式，不直接把一赫茲的時脈連接到年月日的clk。建構一個頻率選擇的模組，把要加速的倍率連接到計時的模組。

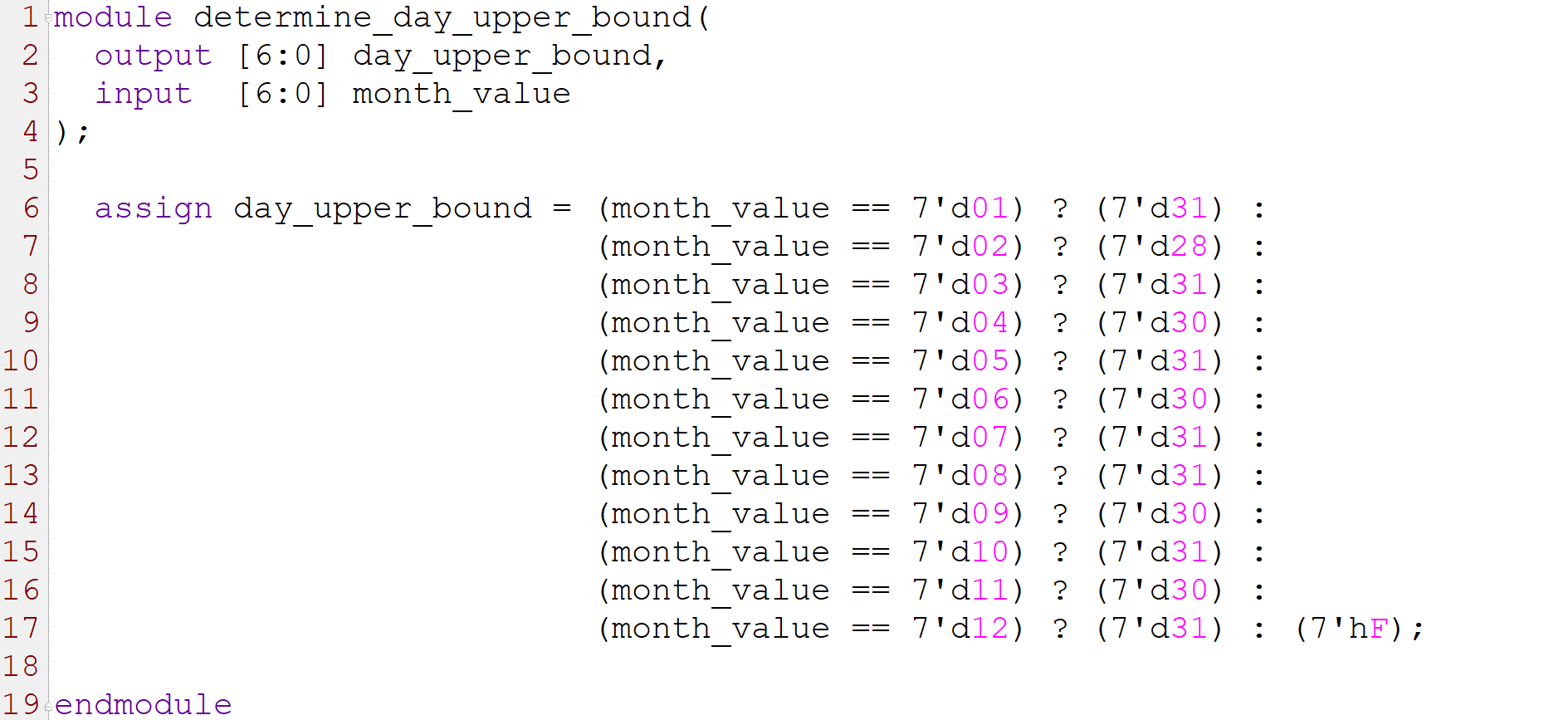
由於負責進位判斷沒有內建在剛剛的年月日的七位元計數器內，所以要另外寫。可以直接把年跟月的clk改成連接到一個比較器，如果前一位數的數值為7’d1，那就代表前一位數進位了。我將比較器的結果連接到debounce模組之後才再接到年跟月的clk，主要是考量到比較器的輸出結果在輸入有變化時會有不穩定的數個波動，有點類似按按鈕時的彈簧震動，會使輸出快速的在1’b0跟1’b1之間切換。因此，如果不經過debounce模組就直接把比較器的結果輸出到月跟日的clk的話，常常會遇到明明前一位進位只需加一，但是卻因為比較器的輸出有震盪而增加了三、四次。



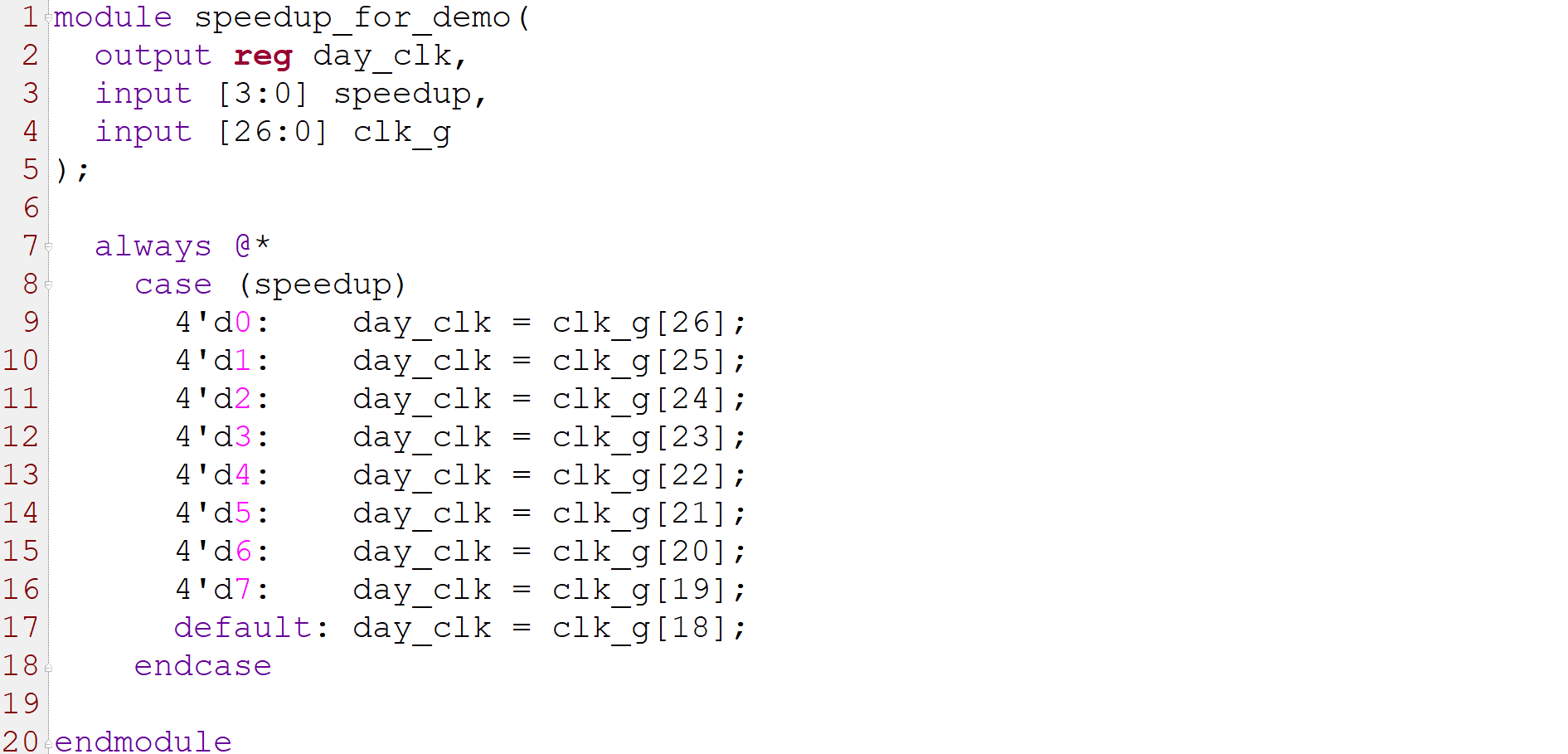
最後，把上述判斷出來的上界、clk、和下界指定給各個時間單位即可。



上述模組其中determine\_day\_upper\_bound直接根據月份來判斷日期的上界。

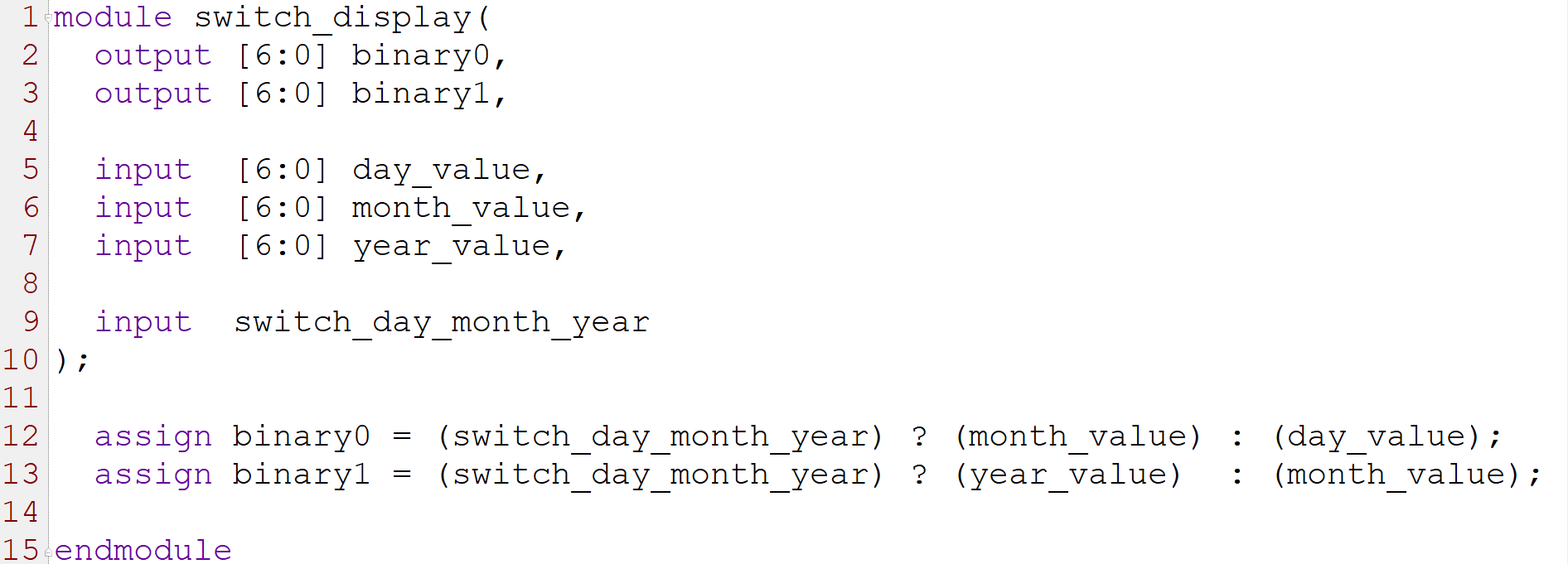


用來demo時加速的模組也是用MUX組成。

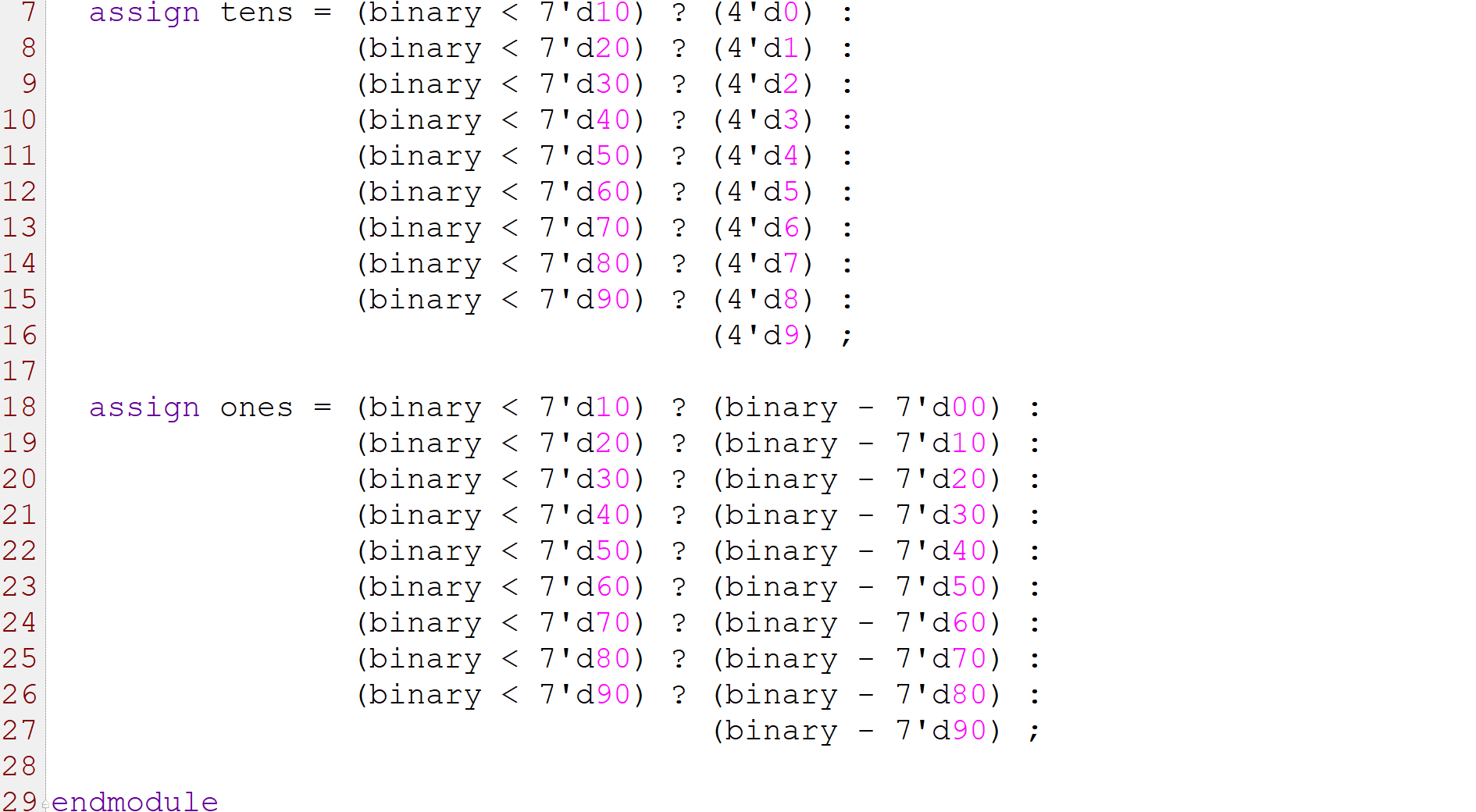


至此，timer已經建構完成，接下來要建構用控制顯示器的一些模組。

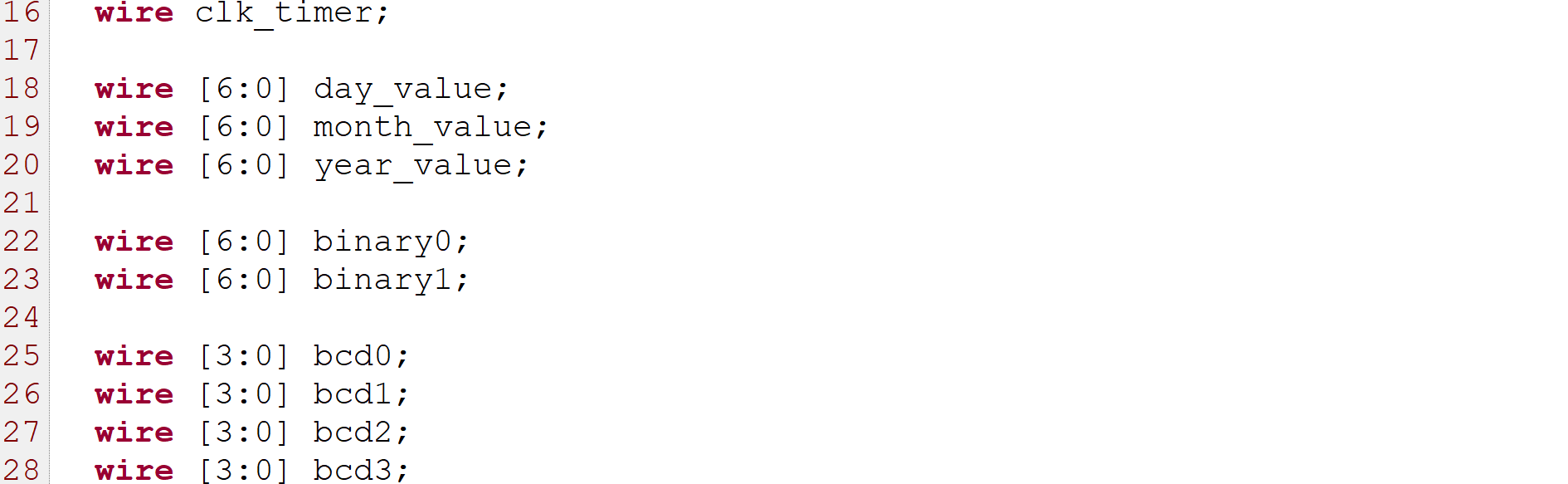
由於FPGA上的七段顯示器只有四個位數，所以需要使用一個switch來切換顯示年月或是月日。依照switch\_year\_month\_day來判斷要將年、月、日三者中的哪兩個訊號傳送到顯示器。



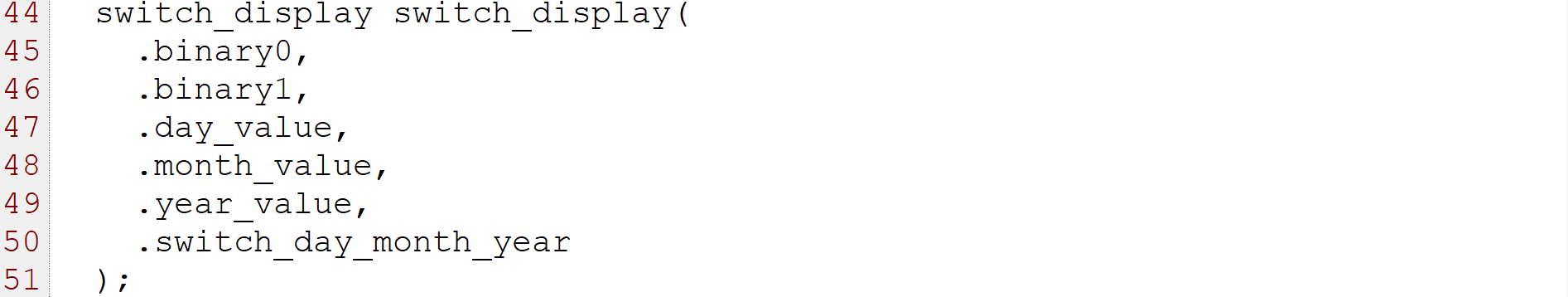
並使用七位元二進位轉二位數BCD的解碼器來把時間訊號轉換成BCD。

這裡有很多方法可以套用。我簡單依照十位數把所有情況分成十種case。當然，這裡也可以直接把一百種情況列出來，只是我覺得那樣會有點太冗長。

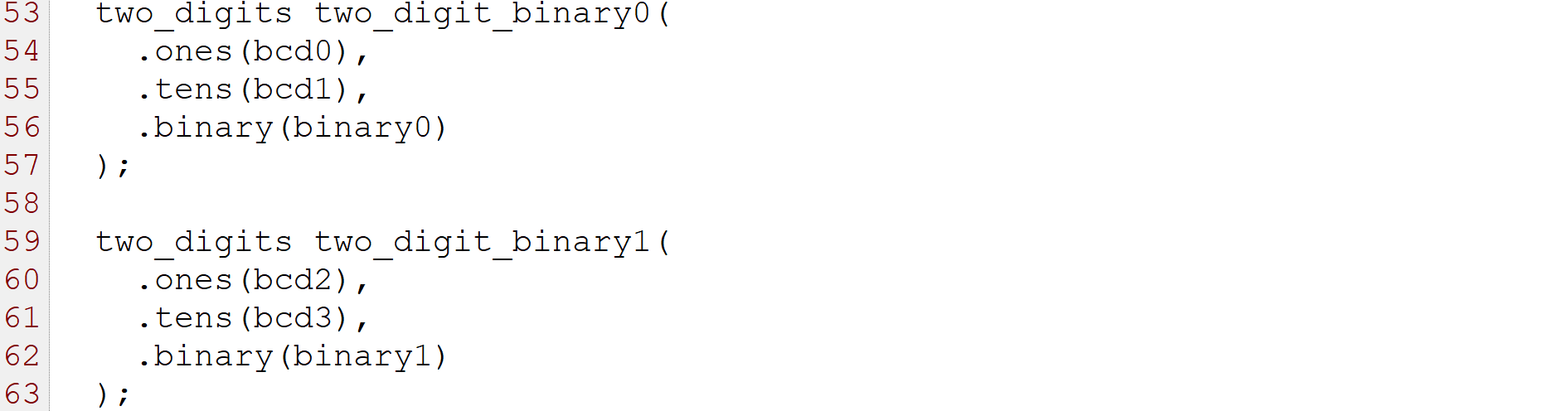
把以上三個模組（timer、switch\_display、two\_digits）包裝在一起，就完成了本題的計時功能與顯示功能。



連接方法很直觀。為了能控制要顯示年月還是月日模式，timer的時間訊號先連接到switch\_display判斷要保留年月日三者中的哪兩個訊號。

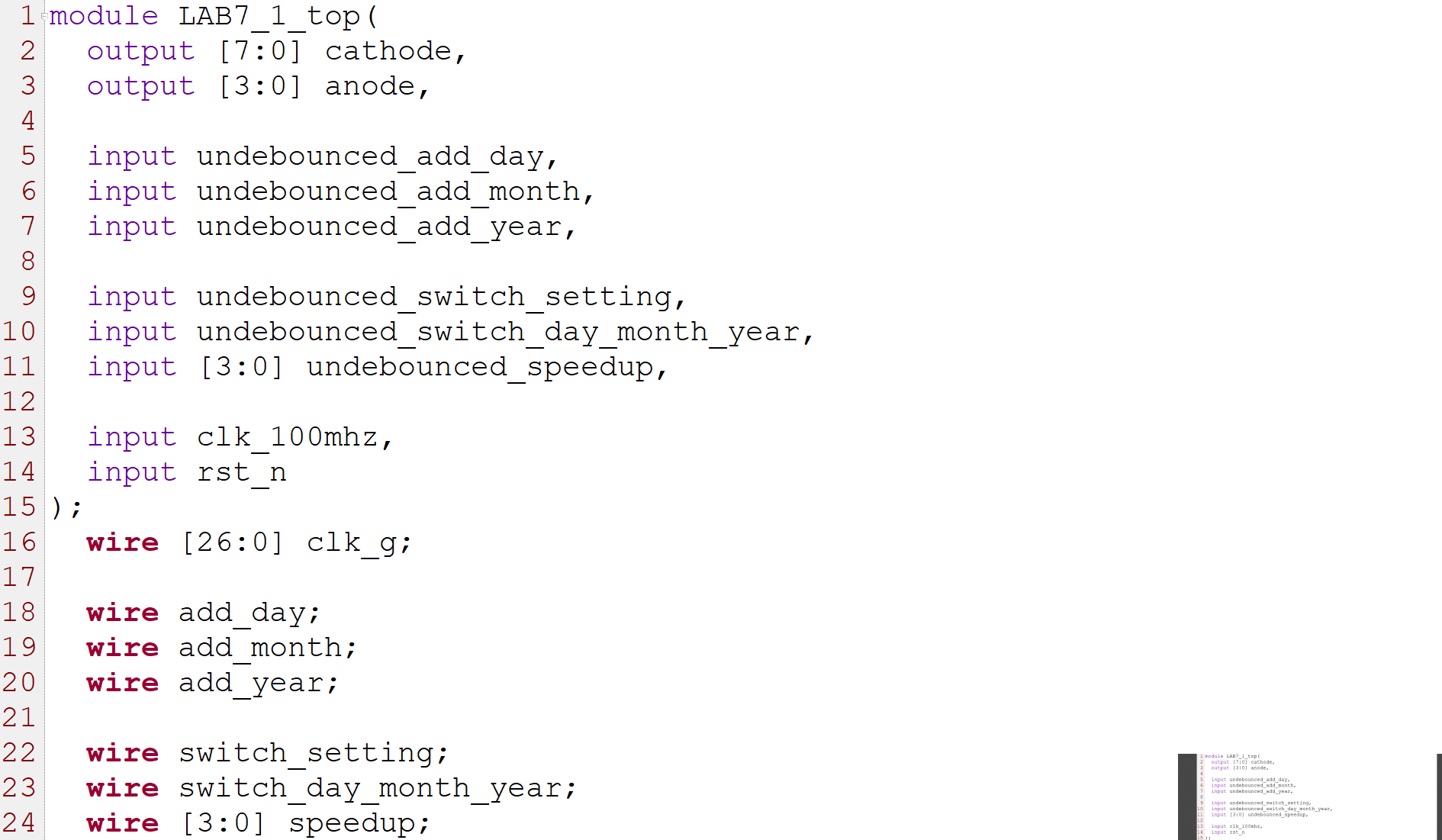
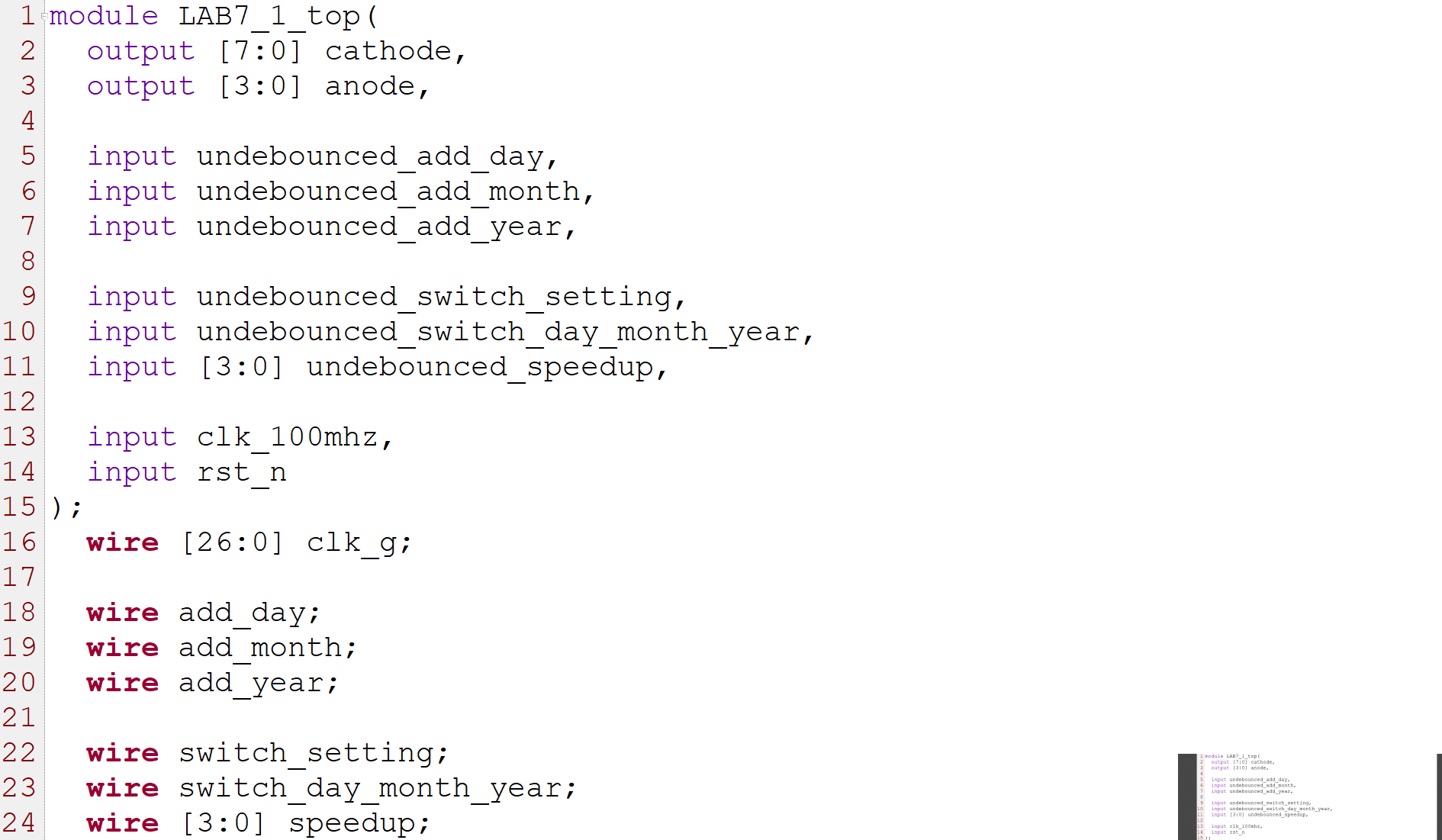
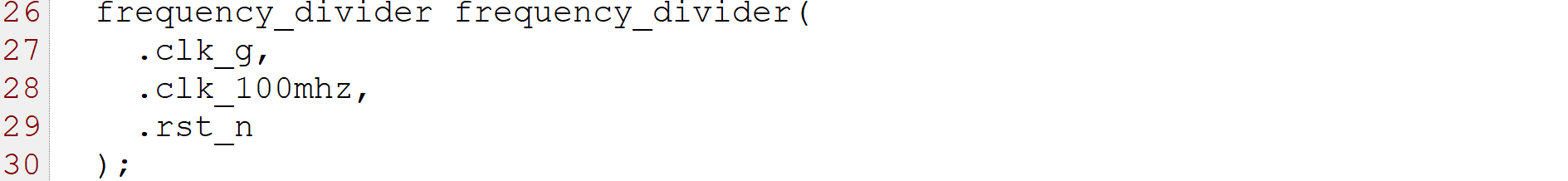


再把保留的訊號經過解碼器轉成總共四個BCD之後，連接到控制七段顯示器的模組。其中七段顯示器的模組（ssd\_ctrl）跟上一題一模一樣，直接沿用即可。

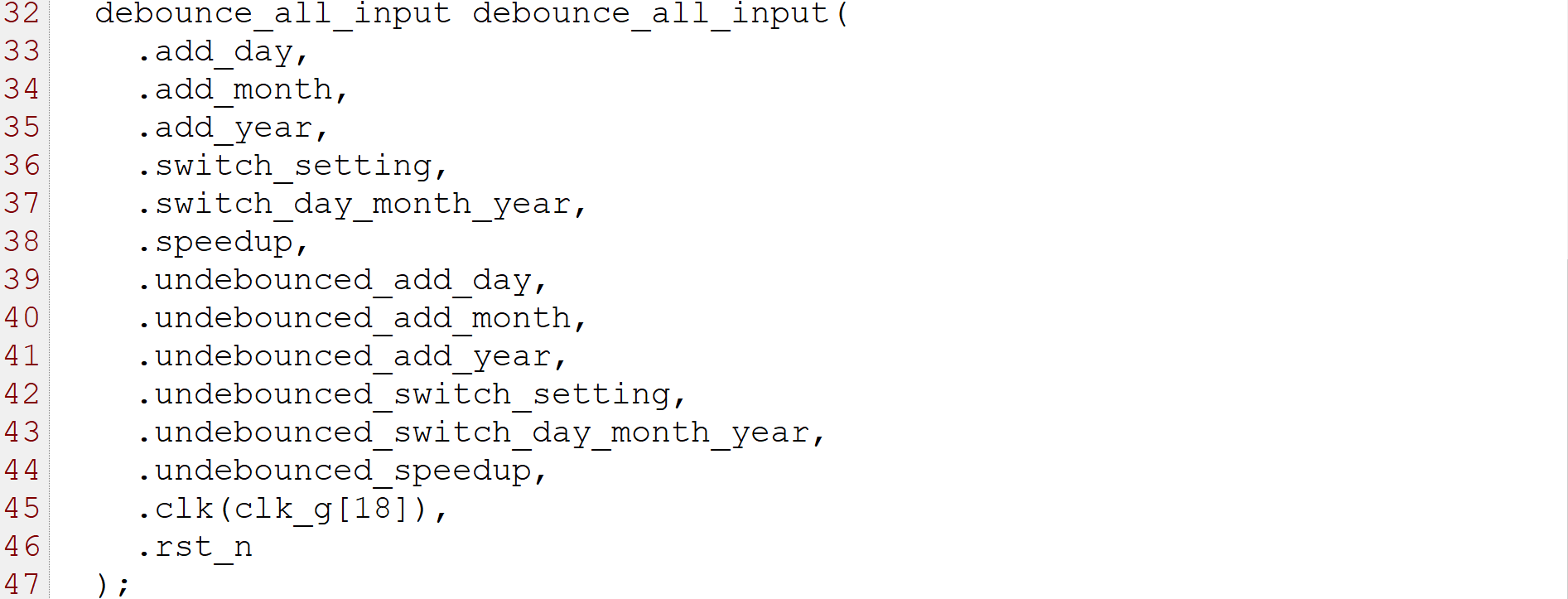


至此，我們已經把所有計時和顯示的功能包裝好在一個模組（electronic\_clock）裡了。下一步就是將top建構出來。

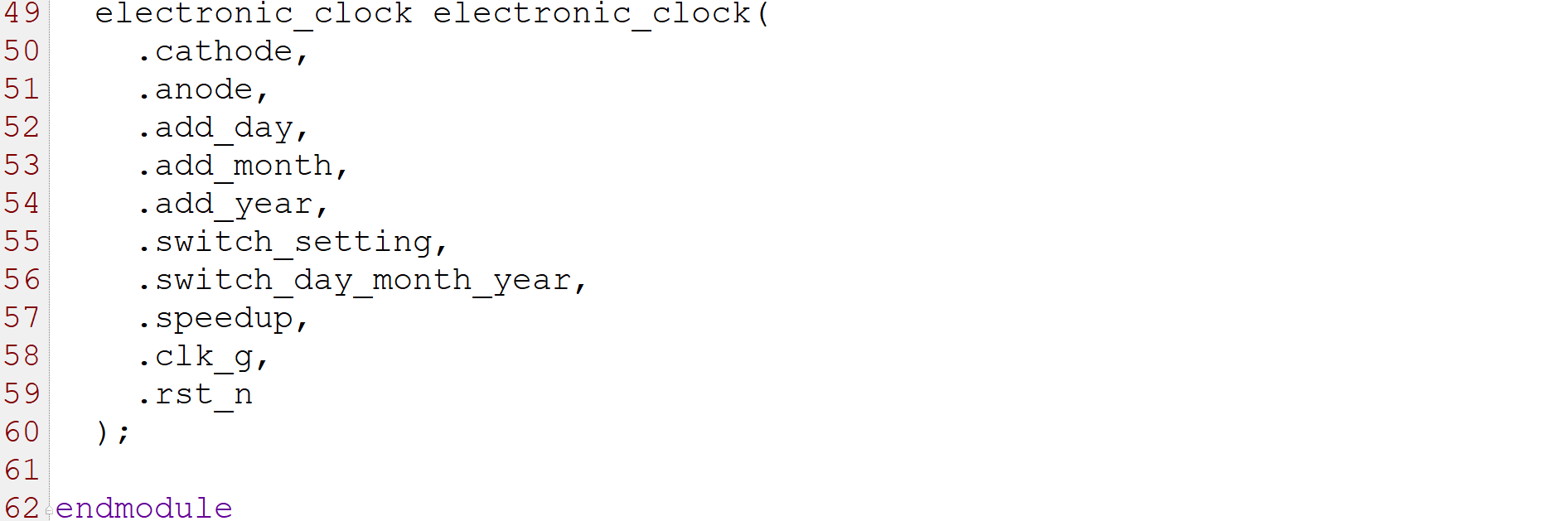
跟上一題一樣，由於我們已經把該建構的功能都建構出來了，我們接下來只需要將一些FPGA上面的input連接到electronic\_clock上即可。

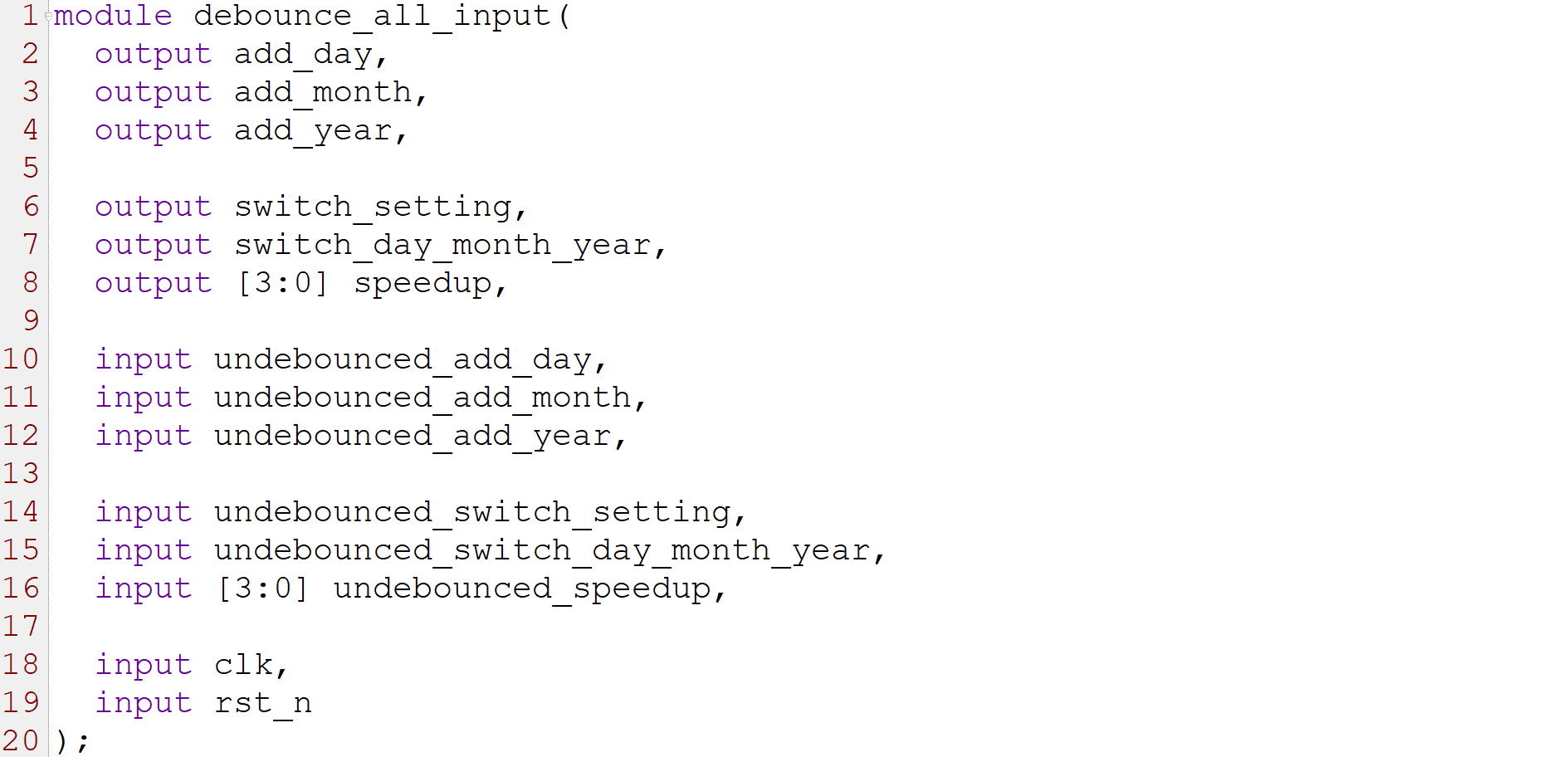
FPGA內建的時脈clk\_100mhz不用debounce，直接連接到除頻器，再將除頻器的輸出與electronic\_clock接好。

先把所有可以用手操作的input連接到debounce模組。

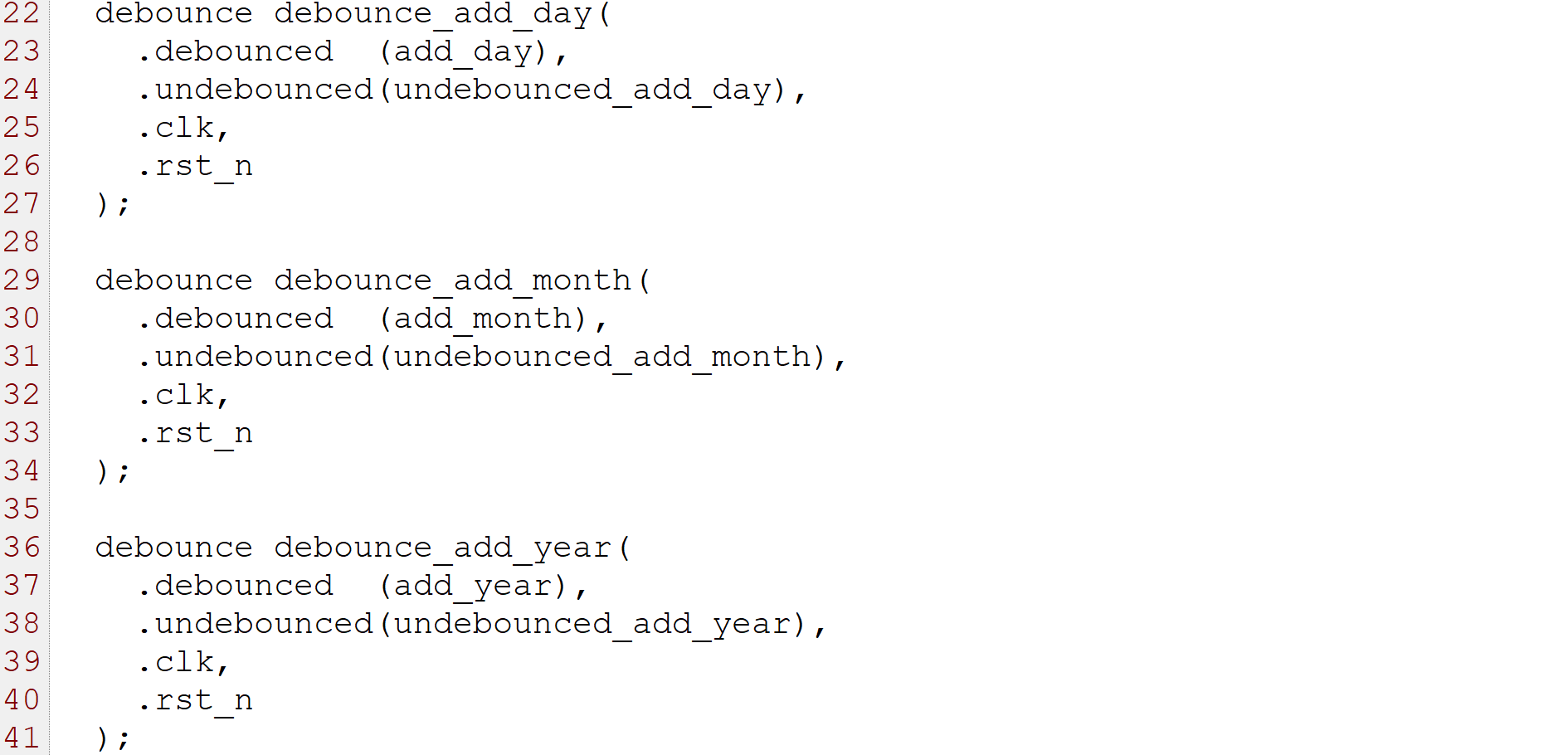


debounce完畢的訊號再與electronic\_clock對應的port連接。

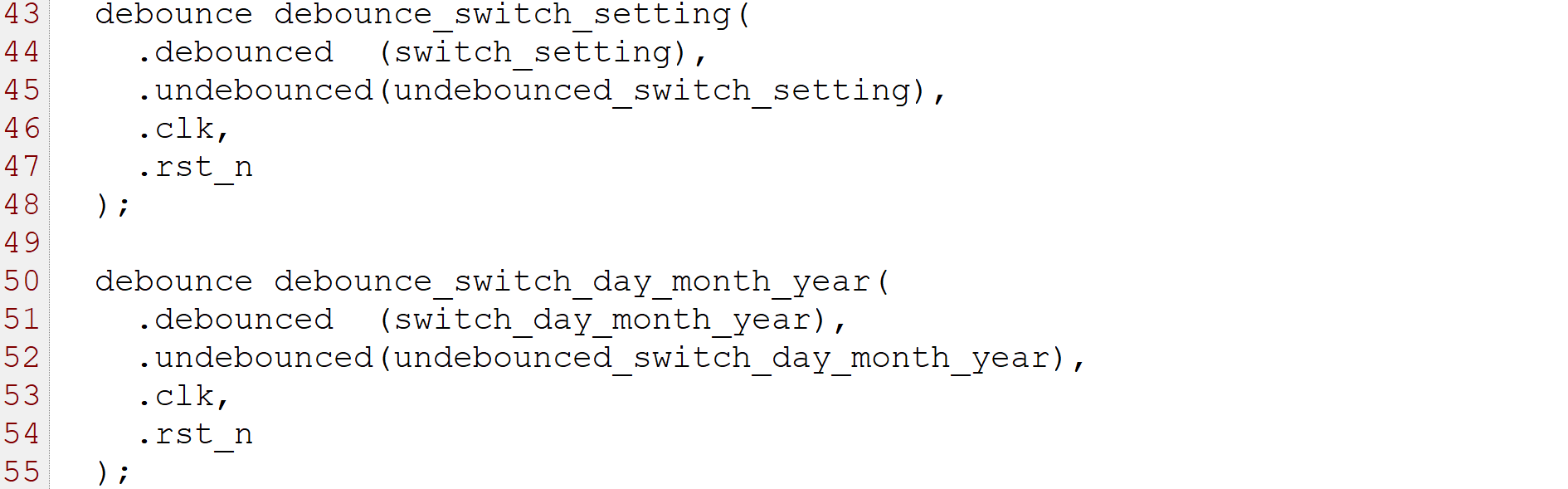


其中，debounce\_all\_input就只是把所有訊號debounce後再輸出而已。

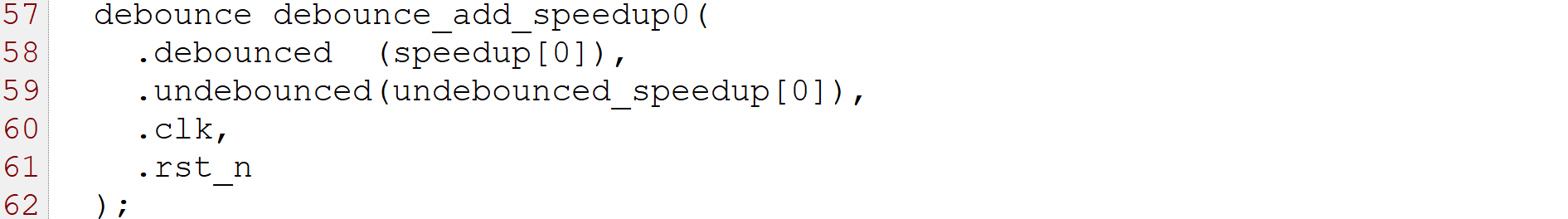
用來增加天數的按鈕。年和月的只要法照並把day改成year和month即可。



用來控制顯示和設定模式的兩個switch。



四個控制demo速度的switch，其他三個只要把0改成1、2、3即可。



**5. Conclusion**

**內容: 可以寫下你的這個lab的想法、遇到的問題、解決方法、心得等等，請自由發揮。**

這題我折騰了半天，到了隔天早上我才debug完畢。用來計時年月日couunter的那部份我原本想直接沿用上一題的模組，但是無論我怎麼修改判斷個位數的下界的邏輯，都會莫名其妙不進位或是一直卡在零然後不上數。有鑑於上一次LAB的經驗，我直接毅然決然直接重頭開始再建構一次，並採用了新的方法，不再需要冗長而複雜的上下界判斷。

此外，這一題讓我學到一個很重要的觀念：這是硬體描述語言，不能單憑之前學過的軟體的觀念來寫，還必須考慮到實際上的硬體層面的問題。例如：原本在我的模組timer裡面，年和月份carry\_in沒有debounce就直接連接了，這就造成了很嚴重的問題，那就是比較器需要一點點的時間才能反映出當下真正的數值。

換句話說，如果要把非天生就是時脈的訊號，例如非clk\_100mhz、非除頻器製造的clk\_g[26:0]等等，連接到某個Flip-flop的c當作時脈的話，就建議要考慮是否有需要debounce或甚至debounce + onepulse。這讓我驚覺原來我學期剛開始的那幾個倒計時的LAB會一直做不出來的原因，例如，當時還沒有意識到以下形式的硬體描述語言背後的真正含意：

reg q;

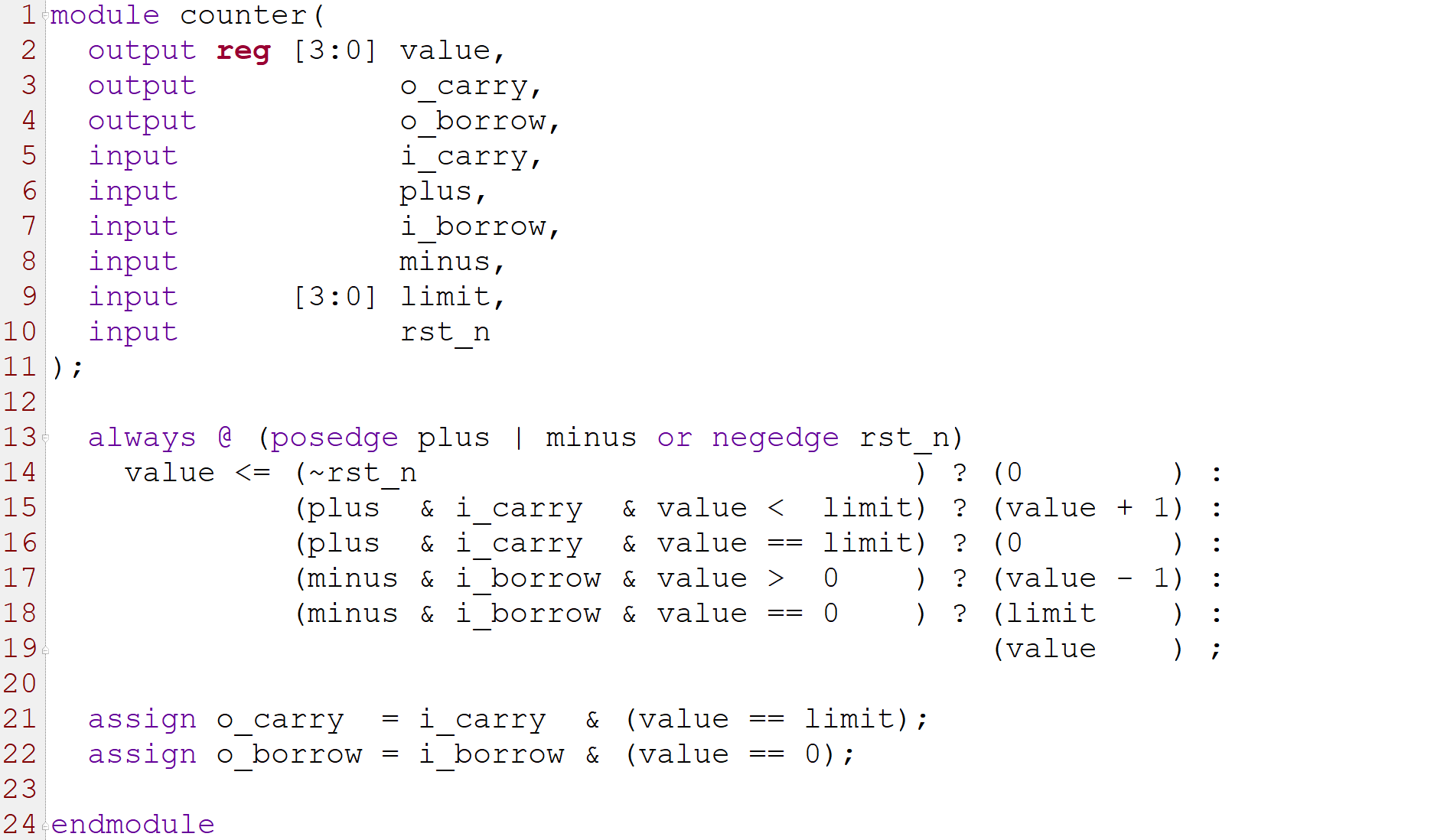
wire a, b, c, d, …;

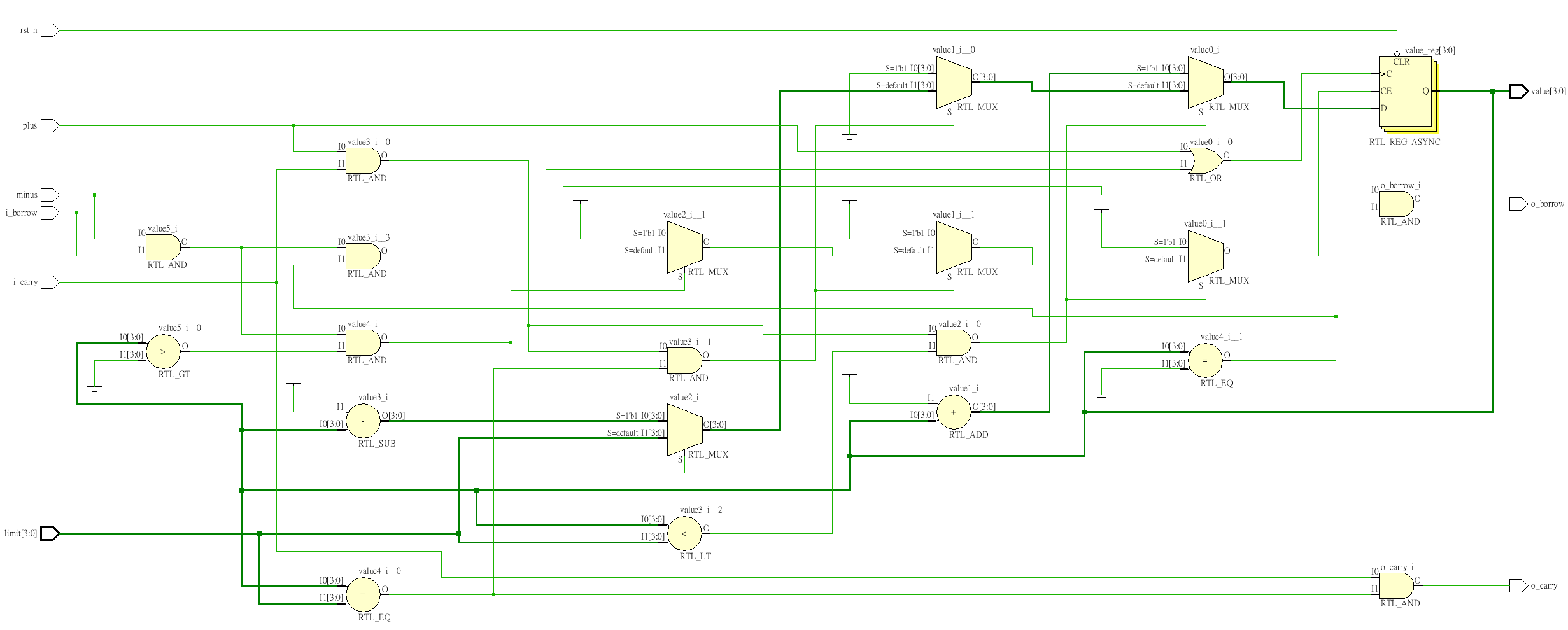
…

always @(posedge a | b | c | …)

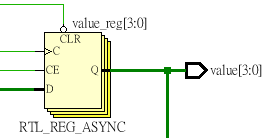
q <= (*o/p of some CL w/ a, b, c, … as inputs*);

更直接一點的例子（上數兼下數計數器，來源：LAB6的第二個失敗版本）：



我原本所期待的是，每個plus和minus的rising edge都會使得value的數值增加一或減少一。雖然依照軟體語言的解釋方法，這段（第13到19行）描述完全沒有問題。然而，由於plus和minus同時出現在D Flip-flop的C和D前面，很有可能在D還沒有反應完畢當下plus或minus改變所造成的不同輸出之前，因為C已經先有rising edge了，這個Flip-flop就把還沒經過combinational logic運算完畢的「舊的」D值存入value。從下圖RTL analysis的schematic可以看出來：

請特別留意上圖右上角的Flip-flop。



由於D前面是一大串combinational logic的運算結果，有可能在plus或minus改變後、D更新完畢當下應有的值之前，這個D Flip-flop就把舊的D值存入value了。要改善這種情況其實很簡單，只要把原本直接連接到C的plus與minus先經過debounce之後才接到C即可。如此一來，D前面的一大串CL就能有數個clk的時間可以確保提供已經運算完畢的新D值給Flip-flop。

由上述觀察可以發現，debounce或onepulse模組不僅僅可以用來穩定FPGA外來的非clk的input，還可以用在有需要拿非clk的signal用來當作Flip-flop的C的時候使用，用來穩定CL輸出（如第一個本題的比較器與carry\_in的例子）、或是用來延遲C收到rising edge的時間（如第二個上數兼下數計數器的例子）。

此外，我去實驗室demo時，現場的助教檢查了一下我的verilog。助教說，D Flip-flop的CLK盡量只連接FPGA內建的時脈（clk\_100mhz）或是經過除頻器處裡過後的時脈（clk\_g）。CLK盡量不要直接與除了時脈之外的input連接，比較理想的替代做法應該是，把所有input訊號先傳到FSM，再由FSM傳訊號給Flip-flop。