Logic Design 106061212 賴傳堯

Lab11

(1) Scrolling Picture

I/O:

Input: clk

Input: rst

Input: rst_n

Input: btn_en

Output: [3:0] vgaRed

Output: [3:0] vgaGreen

Output: [3:0] vgaBlue

Output: hsync

Output: vsync

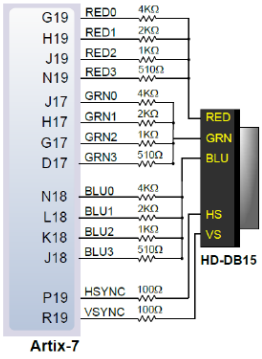
Pin:

W5 - clk

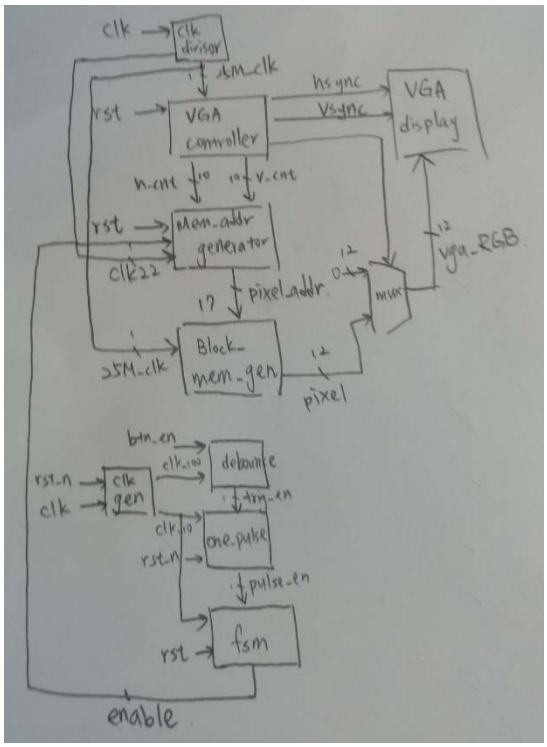
V17- rst_n

U17 - rst

W19 - btn_en



Block diagram:



Discussion:

題解&思路&作法解釋:

這題要替滾動圖片做 enable，滾動圖片部分的 code 直接採用老師給的程式(包含 clk_divisor、mem_adder_gen、blk_mem_gen、vga_controller)。Enable 要用按鈕操控，所以先做 debounce 和 one_pulse(因為後面只用到 pulse_en 的 posedge，所以 one_pulse 其實可以省略，直接用 trig_en 的 posedge)，然後訊號接進 fsm 中，讓 enable 在每按一次按紐時才會從 1 變 0 或 0 變 1。最後把得出的 enable 寫進 mem_addr_generator 中，當 enable=0 時，position 不改變；enable=1 時，才每個時刻加 1，控制圖片向上滾動。

按 enable 之前
(即 rst 的狀態下)

按第一下 enable，
圖片開始向上滾動

按第二下 enable，
圖片停止滾動

再按一下 enable，
圖片恢復滾動

Conclusion:

在做這題時，如果光看老師的 mem_addr_gen 模組，可以看出是 position 在控制圖片的位置，只是這個 position 用來做 pixel_addr 的計算的部分就不是很懂是甚麼意思，也不清楚實際上這個改變是如何去控制螢幕的。

另外，做 enable 的部分就相對容易多，畢竟之前已經做過多次，大致上寫法甚麼的還算熟悉。

(2) VGA-Displayed Calculator

I/O:

Input: clk

Input: rst

Output: reg [3:0] vgaRed

Output: reg [3:0] vgaGreen

Output: reg [3:0] vgaBlue

Output: hsync

Output: vsync

Inout: PS2_DATA

Inout: PS2_CLK

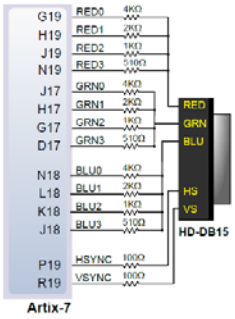
Pin:

W5 - clk

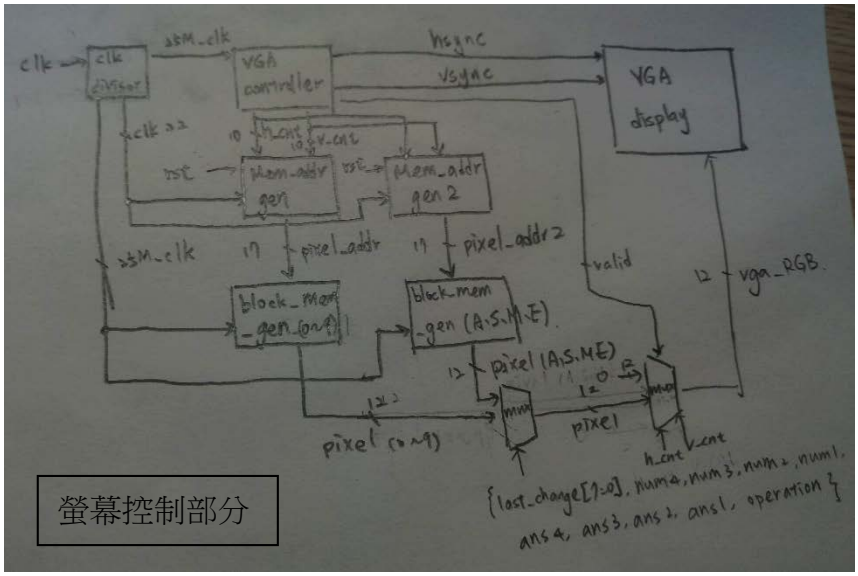
U18- rst

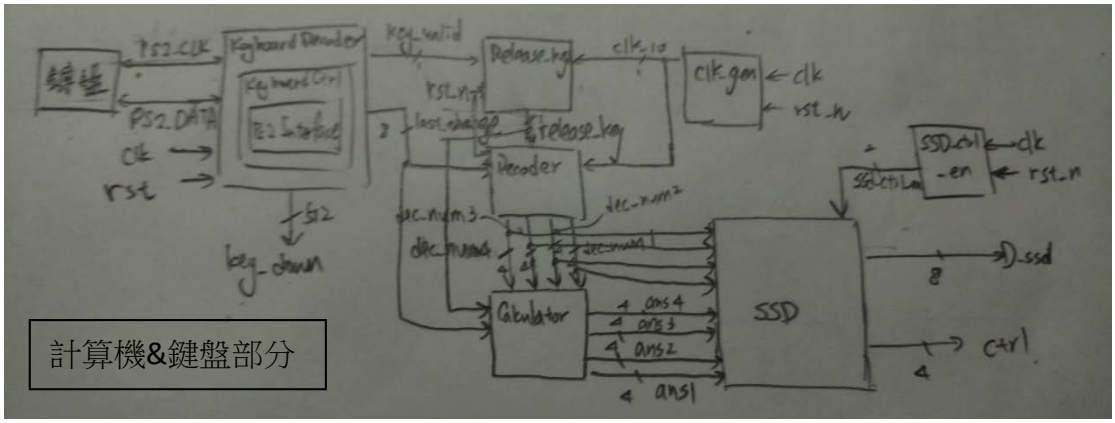
B17 - PS2_DATA

C17 - PS2_CLK



Block diagram:





Discussion:

題解&思路&作法解釋:

這題要把之前的計算機顯示功能從七段顯示器改為螢幕顯示。基本上就是找好運算符號及數字的圖片、一一處理大小、轉檔、再放入程式中。將螢幕中，從橫坐標 192 到 448 分割為五個位置，使得每個寬 64、高 64(介於縱坐標 128 到 192 之間)，其他部分皆顯示黑色。判斷 last_change 是否為 enter 鍵，如果是，則第一個位置顯示等號，第二、三、四、五個位置則分別判斷 ans4、ans3、ans2、ans1 的值為何並顯示對應數字；若否則第一、二、四、五個位置分別判斷 num4、num3、num2、num1 的值為何並顯示對應數字，位置三則判斷 operation 值為何，0 則顯示+、1 則顯示減、2 則顯示*。其中，讀取數字圖片的 mem_addr_gen，是把螢幕在讀到五個顯示圖片位置的時候，重複讀取數字圖，其他部分則讓 pixel_addr=0。而運算符號的圖片由於只有 32*32，所以另外寫了 mem_addr_gen2 來處理，方法基本上相同，只是讀取即顯示位置多加幾條判斷示，使得其位置為 64*64 大小範圍的正中間。

剩下的計算機部分，則直接取用前面 lab9-2 的模組，刪去 SSD 顯示的部分後直接套用。

實驗過程困難&問題:

在寫這題時，真的是花了快一整天在研究究竟圖片的 addr 和螢幕顯示 v_cnt、h_cnt 之間的關係，只是無論怎麼看、怎麼改、怎麼試都還是一頭霧水。剛開始真的很難理解螢幕數 h_cnt、v_cnt，跟讀取圖片的哪個位置是甚麼關係，也很難想像要如何控制哪裡顯示甚麼圖片，一切都太抽象。後來只好詢問同學，看過對方 mem_addr_gen 的判斷式後居然馬上就懂了。真的很難理解這樣的程式如果沒看過範例，要如何憑空想出寫法，畢竟螢幕背後的控制機制我們所知真的不完全。

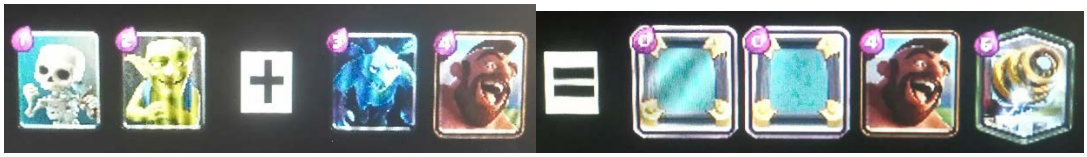
後面在放入圖片時，放第二張圖，就收到 RAM 不足的錯誤，為了這個錯誤又困擾了很久。後來才發現在放入圖片時，要填入的第二個數字我忘了改成 4069(64*64)，才導致系統認為我放了兩張 76800 大小的圖，而 RAM 不足。

Conclusion:

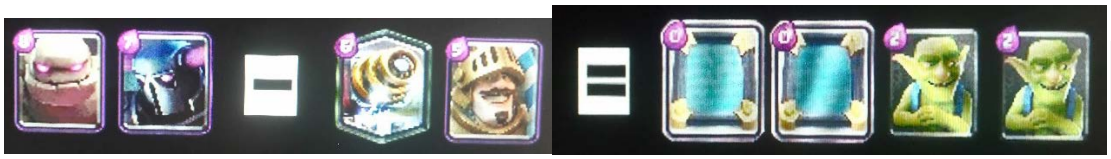
在還沒完全理解的情況下，真的是手足無措，不知道該如何下手；而一旦

弄懂寫法，其實就簡單多了，儘管判斷式還是繁雜，但並不需要太多思考。了解到如何讓螢幕顯示想要的圖片、位置後，會的東西更多，感覺期末能應用的東西又更多了!

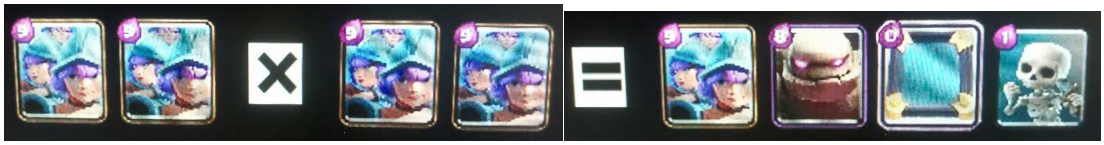
執行結果:



12+34=0046



87-65=0022



99*99=9801

(3) Tetris

I/O:

Input: clk

Input: rst

Output: reg [3:0] vgaRed

Output: reg [3:0] vgaGreen

Output: reg [3:0] vgaBlue

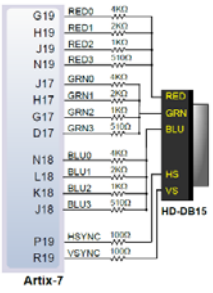
Output: hsync

Output: vsync

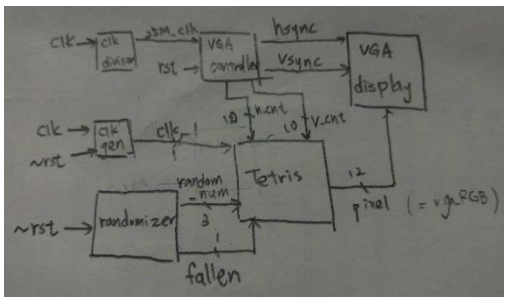
Pin:

W5 - clk

U18- rst



Block Diagram:



Discussion:

題解&思路&作法解釋:

這題並沒有限制一定要外加圖片，所以我就採用直接給定顏色的方式，來顯示俄羅斯方塊。首先因為題目要求顯示範圍寬*高為 10*20 格，所以每格寬為 480/20=24。顯示範圍就在螢幕正中間，當 h_cnt<200 或>440，顯示一整片白色。若是介在這之間的情況下，則判斷模擬亂數產生的 random_num 為和，分別對應到七種方塊。每種方塊的顯示方法，想法是選定某一點，作為置於螢幕水平中心的位置(mid_X)，接著向上、下、左、右用判斷框出方塊的形狀。例如田字形方塊，它的中心很明顯就是四格方塊中間共用的頂點。因此，其圖形設定就是向上後，向左、右各顯示一格，向下亦同。而這個中心的 x 座標就是剛所說的 mid_x，y 座標則是由”fall”這個變數控制。Fall 是一個每秒會增加 24，也就是一格寬的變數也就是說這個中心點的高度=fall 的值，則他每秒就會隨之下降一格。另外，每種圖形都給不同顏色。藉由這樣的方法，就能不放圖片，做出俄羅斯方塊的圖形。

前面提到亂數的部分，是參考黃元豪老師給的講義寫的，用演算的方式模擬出亂數的感覺。

而圖案落到底部消失的部分，則是讓當 fall = (480-24)=456 時，也就是圖形中心離底部只有一格的距離，也就是圖形的下排已經碰到底部的時候，讓 fallen 這個變數=1，傳給 randomizer 模組，產生新的數值，判斷就會更新從最上面烙下另一個新的圖形。

實驗困難與問題:

雖然說解釋起來看似複雜，但其實經過上一題的練習，在寫這題的時候腦袋格外清楚，說實在並不算太花腦力。雖然不算簡單，但比起之前一些繁雜實驗，或這題的 bonus 來說，已經算普通程度的了。

Conclusion:

做完 11-3，代表這學期所有實驗都完成了，真的令人大大鬆了一口氣。這個學期學到很多，上學期邏設所學的根本沒法比較，甚至對 verilog 的理解也完全不同。總之，完成所有實驗真的很令人開心。

(Bonus):

Discussion:

這題我其實思考很久，因為聽同學說，要讓方塊疊起來的話，直接給顏色的寫法會比放圖片要來得簡單的多，所以我也嘗試寫過。

首先第一個問題就是要記住之前出過甚麼方塊，這部分只要設幾個變數，分別記住每次的 random_num，並在疊滿之後刷新(或重新覆蓋掉)就行了。

再來就是疊在底下的方塊必須要在上面方塊落下時，也同時顯示在螢幕上，這部分就跟原本的程式有很大的衝突，解決辦法就是必須先要判斷現在底下有哪些方塊，在 v_cnt、h_cnt 數到那邊時要顯示，剩下的部分，才判斷當前落下的所要顯示的方塊。光是因為這點，就我的想法而言，就會需要大量判斷式，加上因為有”長條狀方塊”的存在，無法保證每次疊加的方塊都是兩格高，判斷就更為複雜了。

Conclusion:

我認為照上面那樣的寫法，經過部分一些可能需要的修正，實際上是可行，只是會非常非常繁雜，衡量之後決定放棄，還是把時間省下來做 final project 比較實際!

Reference:老師 11、12 章講義