Lab11 Report

11-1 One-digit BCD addition

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (有點像是重新整理，讓系統回復初始狀態)

en_pb,(控制圖片的滾動與否)

Output:

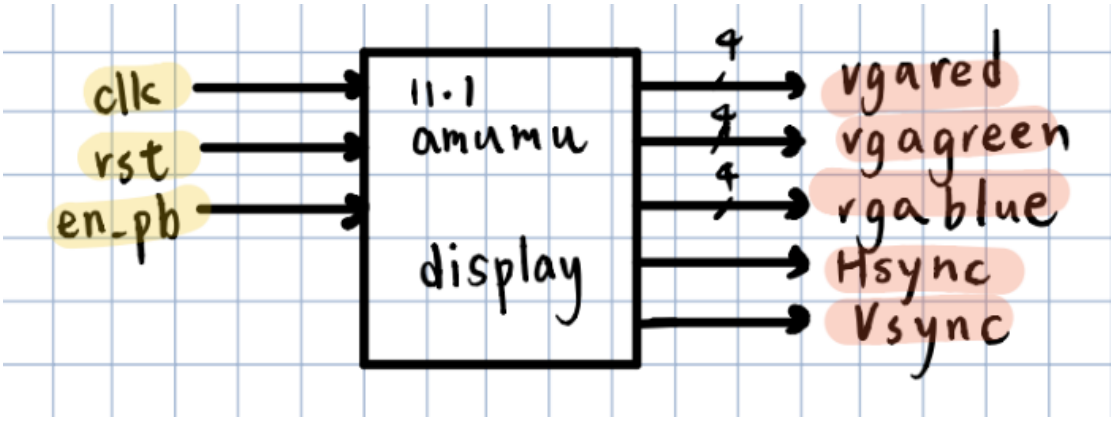
[3:0]vgared, (used to control the red color, 0v(fully off) ~ 0.7v (fully on))

[3:0]vgagreen, (used to control the green color, 0v(fully off) ~ 0.7v (fully on))

[3:0]vgablue,(used to control the blue color, 0v(fully off) ~ 0.7v (fully on))

Hsync,(used for video synchronization in the vertical direction)

Vsync(used for video synchronization in the horizontal direction)



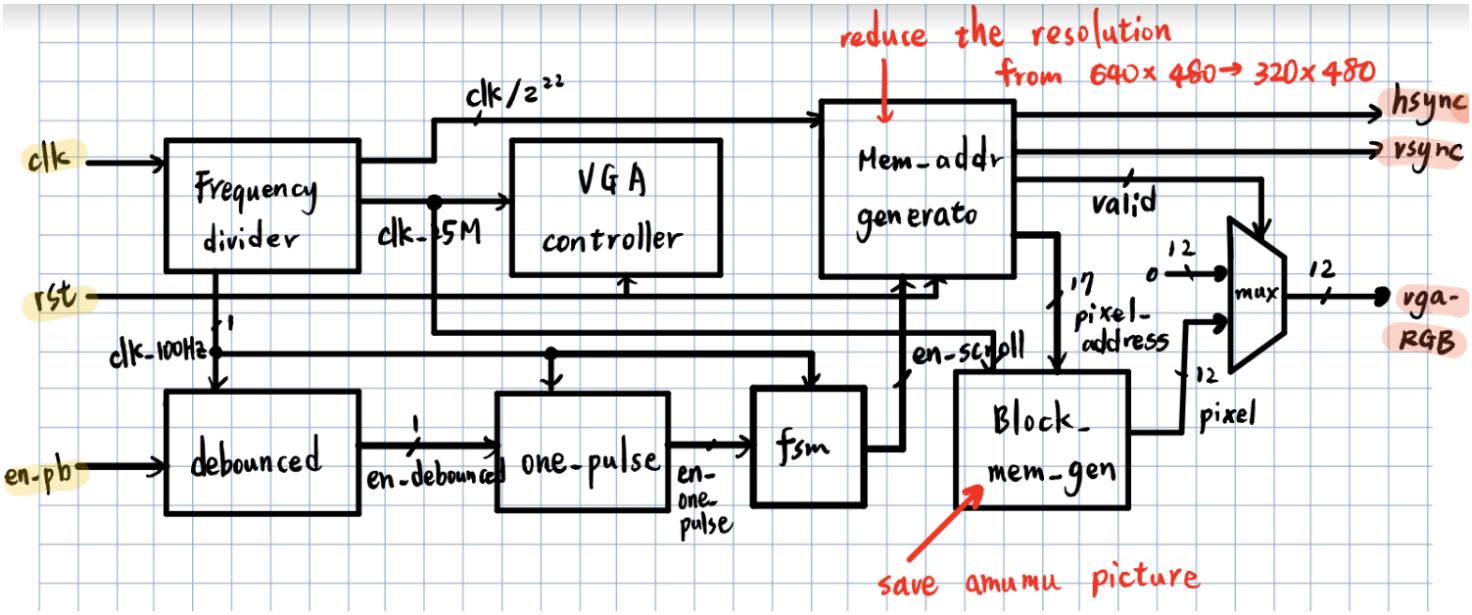
●Design Implementation

1.Outline

這個實驗主要要控制 amumu 圖片的滾動與否，並顯示在 VGA 上面。因為大部分的 code 還有 module 教授都已經給了，所以這個 lab 主要是要加上按鈕 stop/start 功能，還有要理解 code 所代表的意義，其中一部分讓我苦惱許久，不過最後有想出來，會在 Discussion 說明。

因為 FPGA 記憶體容量的問題，只能存放大概 1800hbits 大小的東西，這連一張 640(pixel)*480(pixel)的圖片都不夠，所以要利用助教給的程式把它轉換成 320(pixel)*240(pixel)的大小才行。

2.Logic Diagram



↑ 主要我只加上了 debounced、onepulse、fsm 去控制 mem_addr_generater module 裡面的 postion 來控制圖片的滾動與否。

這邊只介紹之前沒出現過的 module,

i.e. VGA controller 、Mem-addr generater。

▲VGA controller

這個 module 是用來控制 h_cnt、v_cnt、hsync、vsync、vaild。

hsync、vsync 主要用來給螢幕做顯示的，因為螢幕顯示有點像是七段顯示器，不過是每一個 pixel 都要掃過，利用 800*525*60(frame/sec) = 25M(pixel/sec) 速度的 clk 來掃過來達成每秒螢幕更新 60 張的效果。

h_cnt、v_cnt 主要是用來給你做你想要畫面的輸出的，我覺得有點像是 pointer 的觀念，當 h_cnt、v_cnt 指著特定的圖片上面的特定點的時候，vga_rgb 就會回去找那張圖片上面的位置並顯示該畫面，這也是困難的地方所在，在之後 final project 中我們要掃描圖片上的小方塊，並讓他移動，這樣的顯示我覺得並不容易。

Vaild 是讓螢幕知道該在什麼時候顯示，因為螢幕只有 640*480 的大小，換言之在外面其他地方不能亮，所以才會有下方式子產生。

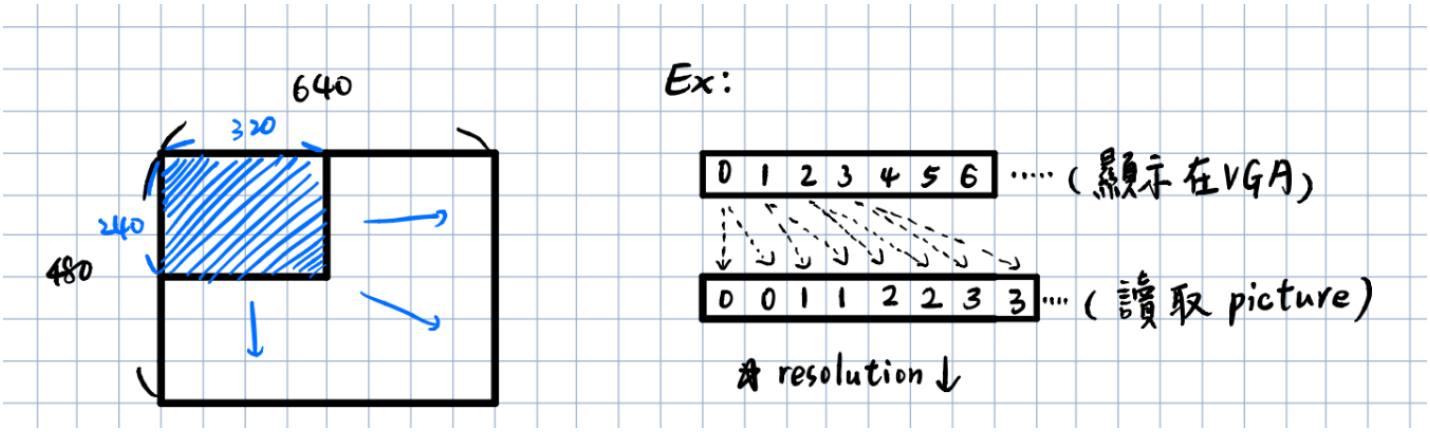
```
assign {vgaRed, vgaGreen, vgaBlue} = (valid==1'b1) ? pixel:12'h0;
```

▲ Mem-addr generater

這個 module 主要是讓 640*480 轉變成 320*240 的解析度，還有利用變數 position 來控制螢幕移動與否。因為是老師給的 module，所以我一開始並不瞭解他的原理，想了很久之後才想通，大概說明一下我的想法:

```
assign pixel_addr = ((h_cnt>>1)+320*(v_cnt>>1)+ position*320 )% 76800;  //640*480 --> 320*240
```

這行可以讓解析度變差的原因是，在二進位中 shift 右邊一個 bit 代表著除以二，而 h_cnt、v_cnt 分別代表著橫軸與縱軸，所以我們把兩個都除以二，最後結果會讓整個面積變成只剩下四分之一，達到所要的效果。(如下圖)



另外要*320 則是因為 pixel_addr 只有一維，所以要利用*320 來記錄二維的數字。Position 能有讓螢幕滑動的效果是因為，他跟(v_cnt>>1)的權重一樣，所以當我們將 position 加大的時候，他掃描的地方就會比原本預期的地方還要下面，造成 amumu 往上的視覺效果。

↓以下是 I/O 接腳

I/O	vgaRed[3]	vgaRed[2]	vgaRed[1]	vgaRed[0]
Pin	N19	J19	H19	G19

I/O	vgaGreen[3]	vgaGreen[2]	vgaGreen[1]	vgaGreen[0]
Pin	D17	G17	H17	J17

I/O	vgaBlue[3]	vgaBlue[2]	vgaBlue[1]	vgaBlue[0]
Pin	J18	K18	L18	N18

I/O	clk	en	hsync	rst	vsync
Pin	W5	U18	P19	U17	R19

●Discussion

其實這一題並不難，主要是要了解每一個 module 的功能，還有熟悉 ip code 的運用方法，經過這個 lab 我才知道有這麼多 lab code 可以運用。不過我覺得這個題目的敘述很奇怪，讓我思考了很久:

1.3 Pressing odd times **en** button to start / resume scrolling. Pressing even times **en** button to pause scrolling. Counter for **en** press is reset to zero when **reset** is pressed.

當初我一直在想到底是為什麼要押偶數次讓他停止，而壓奇數次讓他繼續滑，結果原來題目想要表達的意思是按一下停止、下一次再按再繼續滑 XD。不知道是我理解能力太爛還是這個題目敘述很奇怪，如果題目是要我原本那樣的理解的話，我好像寫不太出來……

11-2. Calculator on VGA

●Design Specification

Input:

clk,(接上板子的原本的震盪頻率 W5)

rst, (控制整個功能的開關，有點像是重新整理，回到起始值)

Output:

[3:0]vgared, (used to control the red color, 0v(fully off) ~ 0.7v (fully on))

[3:0]vgagreen, (used to control the green color, 0v(fully off) ~ 0.7v (fully on))

[3:0]vgablue,(used to control the blue color, 0v(fully off) ~ 0.7v (fully on))

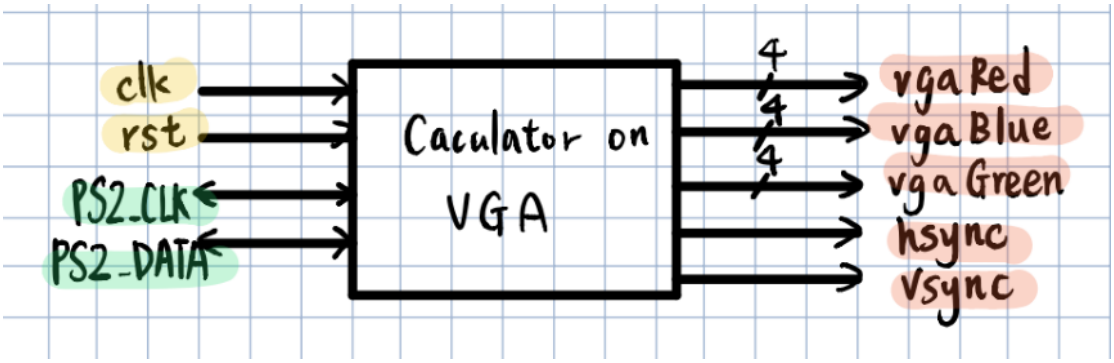
Hsync,(used for video synchronization in the vertical direction)

Vsync(used for video synchronization in the horizontal direction)

Inout:

PS2_CLK,(讓 PS2, KEYBOARD 彼此溝通用)

PS2_DATA(讓 PS2, KEYBOARD 彼此溝通用)



●Design Implementation

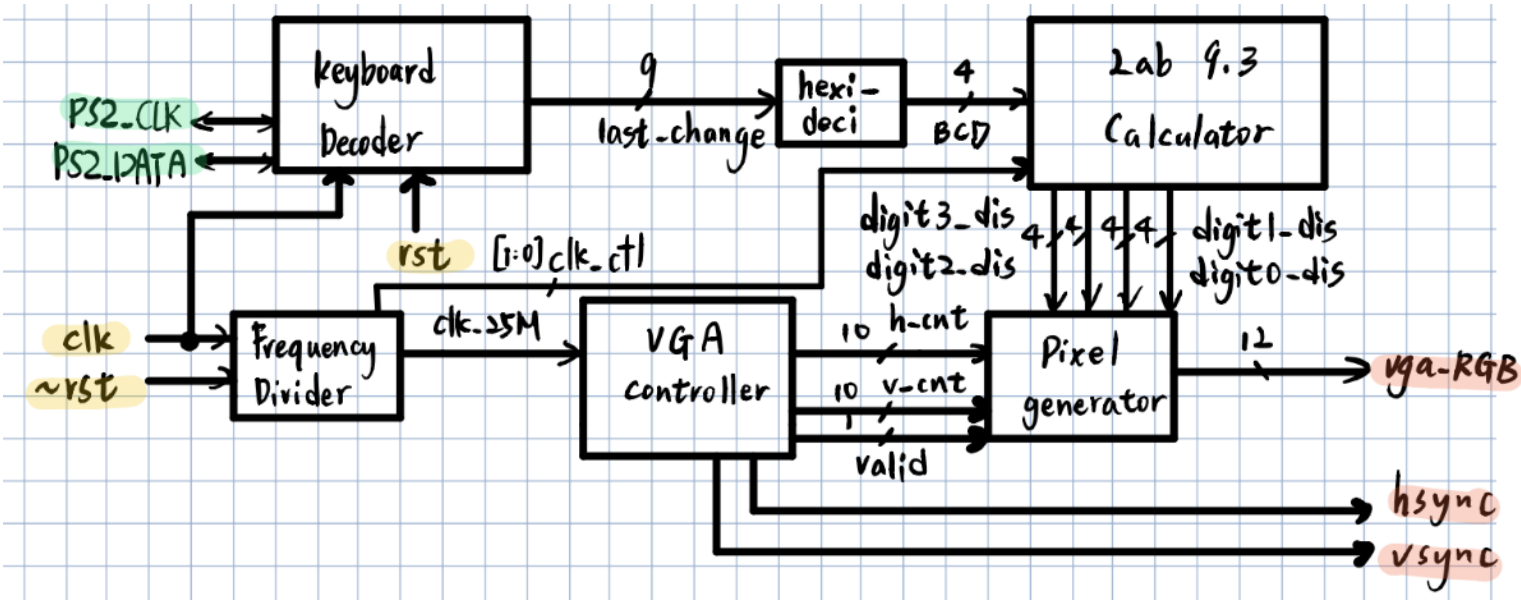
1.Outline

這個題目主要是要把 9.3 的 2 位數加、減、乘法器改顯示再螢幕上，這題看起來好像要改的東西不多，不過實際上我卻花了非常多時間。

有兩種辦法，一個是直接寫死，在螢幕上刻出數字；另外一種是像是阿姆姆那樣，找一張圖片，然後利用掃描的方式把圖片上的數字顯示在螢幕上。

這邊我採取的是寫死的方式，雖然直覺，但是會要許多判斷式，讓整個 module 非常大，比較麻煩。

2.Logic Diagram



↑ 上圖是整個系統的邏輯圖，因為計算機的部分在之前介紹過了，所以這邊略過，只介紹我新增加的部分。

▲Pixel_generator

這邊我設計每一個數字的寬度是 30(pixel)，長度是 60(pixel)，然後模型就是七段顯示器，數字體為白色，其他部分為黑色。因為這不是讀圖片，所以沒有壓縮的問題，所以直接把 h_cnt 還有 v_cnt 的最大值用 640、480 來運用。

這邊舉一小部分的 code 當作例子:

```
always @(clk)
 begin
 case(en)
 2'b00:
 begin
case (digit3)
 4'd15:
begin
   if(h_cnt < 100 & h_cnt > 70 & v_cnt < 210 & v_cnt > 205  )
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
   else if(h_cnt < 100 & h_cnt > 95 & v_cnt < 240 & v_cnt > 210  )
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
   else if(h_cnt < 100 & h_cnt > 95 & v_cnt < 270 & v_cnt > 240  )
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
   else if(h_cnt < 100 & h_cnt > 70 & v_cnt < 270 & v_cnt > 265  )
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
   else if(h_cnt < 75 & h_cnt > 70 & v_cnt < 270 & v_cnt > 240  )
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
   else if(h_cnt < 75 & h_cnt > 70 & v_cnt < 240 & v_cnt > 210  )
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
   else if(h_cnt < 100 & h_cnt > 70 & v_cnt < 243 & v_cnt >  238 )
        {vgaRed, vgaGreen, vgaBlue} = 12'hfff;
```

這邊是第一位數字，我把每位數字的所有可能會出現的數字還有符號都打進去了，上面的每一行 if 條件是所代表的是都是一段七段顯示器，然後我利用變數 a 來代表每個數字的間距，這樣子我之後如果想要改變間距的時候，只要改變我 parameter 的數值即可，這個 module 寫完總共有 1136 endmodule 1136 行……

↓以下是 I/O 接腳

I/O	vgaRed[3]	vgaRed[2]	vgaRed[1]	vgaRed[0]
Pin	N19	J19	H19	G19

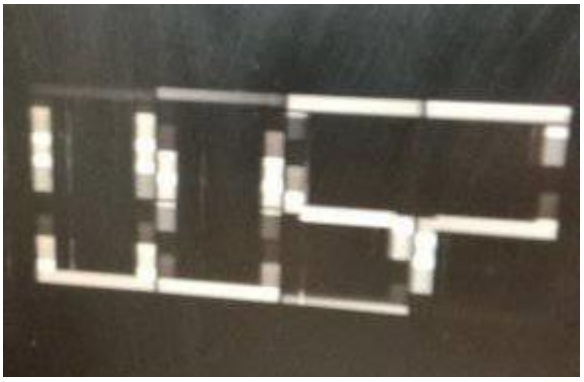
I/O	vgaGreen[3]	vgaGreen[2]	vgaGreen[1]	vgaGreen[0]
Pin	D17	G17	H17	J17

I/O	vgaBlue[3]	vgaBlue[2]	vgaBlue[1]	vgaBlue[0]
Pin	J18	K18	L18	N18

I/O	clk	PS2_CLK	PS2_DATA	hsync	rst	vsync
Pin	W5	C17	B17	P19	U17	R19

●Discussion

這題花我很多時間的原因是我一開始的想法有點錯誤，我原本想要利用七段顯示器的方式寫在上面，也就是說還會有一個[1:0]clk_ctl 來控制整個每個數子的亮暗，雖然最後是做出來了，但是看起來結果會像是這樣:



會不斷一直閃，雖然是看的到，甚至好像還有一點科技感(?但是跟我想像中的不一樣，我原本會想要這樣寫的原因是式子中的

```
else
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
```

我想說每一個 case 中最後的這個 else 會讓其他沒有被宣告的地方都變成黑色，這樣子不是其他地方就不能是白色嗎? 所以我想說那我就利用七段顯示器的概念寫，但可能因為 vga 的 delay 也蠻嚴重的，所以不管我怎麼調 clk_Ctl 的數值，都無法解決這個問題。

最後我才知道，原來完全不用 clk_ctl，因為訊號會蓋過去，就是說，我完全不用考慮那個 else 的黑色，因為在下一個 condition 的時候如果其他地方要亮的話，就會傳出新的訊號了，所以完全不會有這樣的問題。

最後我覺得還是用掃圖的方式比較好，因為這樣的方式非常繁雜，而且好像結果也不太好看(?

●Conclusion

這次的實驗真的是非常困難，我原本在拚 11.3 的加分題，我其實都有前兩題都有做完，但是之後的疊起來我在當天 demo 一直打，到最後卻還是失敗，真的太可惜了，結果筆電就沒有電(忘記帶充電器)，連前面的的不分給分都沒有 demo 到……真的是非常可惜。

我覺得這學期的課程對我來說真的蠻有意義的，這種動手打 code 的我雖然不是特別有天分，不過我卻蠻有興趣的，這讓我這學期決定要做相關的專題了，我和朋友找的是黃朝宗教授，聽說他的課程跟專題都非常的充實，希望我能夠活著出來。