影像處理演算法實現於平板顯示控制器 F*Con-V.1 之晶片製作

Chip Implementation of Image Processing Algorithms for Flat Panel Display Controller F*Con-V.1

宋志雲　陳永璁　陳志炘　宋易勳
中華大學電機工程學系
Tze-Yun Sung　Yung-Tsung Chen　Chih-Sin Chen
Department of Electrical Engineering
Chung-Hwa University
Tel:886-3-579-0902　Fax: 886-3-578-1150
Email: bobsung@btcx.com.tw, wing777@btcx.com.tw, sin@btcx.com.tw, tom@btcx.com.tw

摘要

本文旨在針對平板顯示器控制器(Flat panel controller)內部之規模縮放器（Scaler）亮度與對比度調整器(Brightness and Contrast Adjuster)、γ修正器(Gamma Correction)與影像顏色濃淡補正器(Dithering)等模組的演算法(Algorithm)深入探討並以硬體描述語言(Hardware Description Language)實現電路,以達即時工作的能力。

1. 規模縮放器（SCALER）

規模縮放器，主要是將輸入之畫面做放大或縮小之動作。但是如何將畫面做放大或縮小之動作且盡量達到不失真的狀態，就是文中要說明的部分。

在硬體結構上中我們並不使用 Deinterlace 的架構也不使用 DSP。在處理小數點之過程中也僅以最簡單之乘法器及利用某些位元來達到4捨5入之運算，並在硬體架構上使用最少的記憶體以節省成本，本文中僅討論規模縮放器的運作方式。

圖 1，是整個平板顯示器控制器的流程圖，所表示的僅是資料之流向，其順序亦與實際的系統架構圖相同。不同的只是少了暫存器、MCU 介面與細部的關連圖。

圖 2，是規模縮放器的架構圖。INP 方塊之功用在於辨別輸入端之訊號，資料經由 INP 模組選擇判斷並轉換為 YUV4：2：2 模式（Video、S-Video、HDTV），若為 RGB 訊號則不轉換。之後 INP 模組便會將視訊訊號與相關控制訊號輸出給規模縮放器（SC 模組）。CSC 則為色彩空間轉換器（Color Space Convertor），其功能在於將經過規模縮放器運算後之視訊資料轉換為 RGB 之視訊規格（YUV 4：2：2 轉為 RGB 規格，若原輸入即為 RGB 訊號則不做任何動作）。CSC 的內部架構僅是將 YUV 轉 RGB 的公式以硬體電路來實現（乘法器、加法器與位移之組合），故在此並不解釋其內部架構。

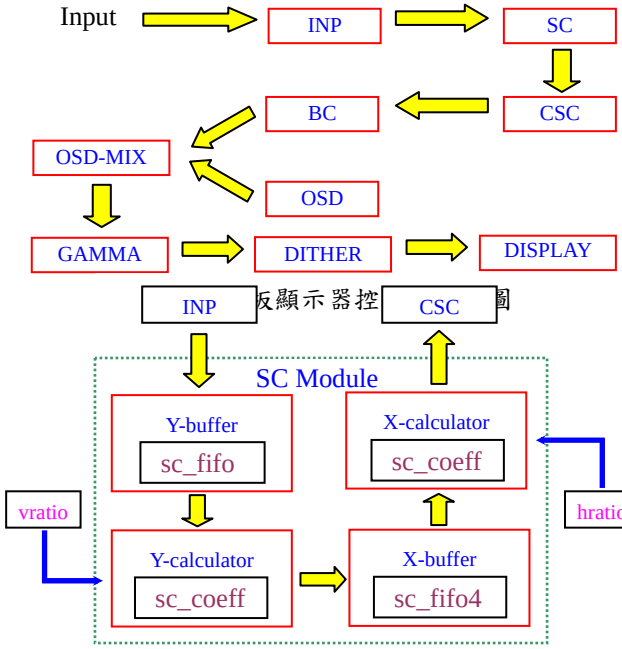


圖 2　規模縮放器架構圖

在架構圖中，共有 Y-buffer、Y-calculator、

X-buffer、X-calculator、sc-fifo、sc-fifo4、sc-coeff、SC-main 等模組。其中 sc_fifo 與 sc_fifo4 模組為提供 Y-buffer 與 X-buffer 使用之記憶體模組，其宣告與動作之方式不在此敘述。如架構圖中所示，進入之畫面訊號均先進入 Y-buffer 中儲存，為節省成本並且不損及性能，只在 Y-buffer 中使用了可以儲存 8 條掃瞄線的記憶體（1024 pixels * 3-bytes * 8 Lines）。

1.2 資料的存取與運算

在創造出新的水平掃瞄線之期間，每一個輸出時脈均會取上下相鄰之 4 條水平掃瞄線上同一個水平位置的 pixels 之資料來做運算，並以此造出新的 pixels，而這一些新的 pixels 的集合就是 1 條新的水平掃瞄線。以上之運算過程就交由 Y-calculator 來做運算，Y-buffer 僅負責取出 4 個上下相鄰之 pixels 資料。取出 pixels 資料的示意圖如圖 3 與圖 4 所示：

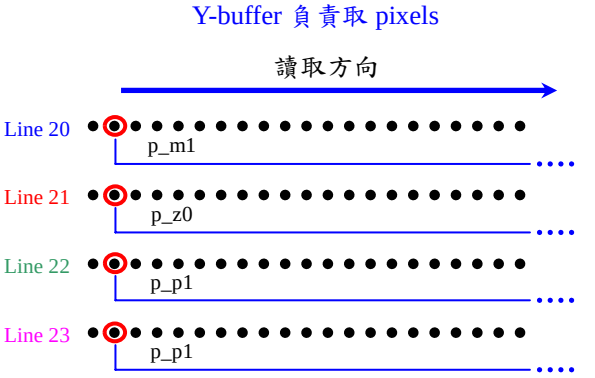


圖 3　Y-buffer 讀取資料示意圖

續圖 3

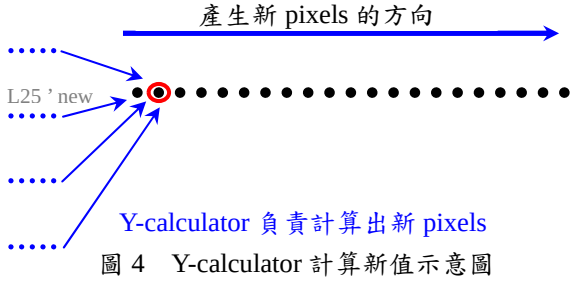


圖 4　Y-calculator 計算新值示意圖

註 1：　L25 'new 表示是已經擴（縮）完線之新水平掃瞄線（暫時性的），但尚未擴（縮）點，故並非最後輸出。

如圖 3 與圖 4，在 L25'new（暫時性的新水平掃瞄線）產生期間，在所設計的硬體電路中，資料的讀取一直維持在 Line 20 ～ Line 23 相同水平位置的 pixels 的資料，並將所得到的資料依序指定為 p_m1（pixel minus 1）、p_z0（pixel zero）、p_p1（pixel plus 1）、p_p2（pixel plus 2），而新 pixel 值所使用的部分程式，如程式片段 1 與程式片段 2，與計算公式如下（註 2）：

```
assign my_m1 = p_m1[23:16] * coeff[34:27]; // C3
assign my_z0 = p_z0[23:16] * coeff[25:18]; // C2
assign my_p1 = p_p1[23:16] * coeff[16: 9]; // C1
assign my_p2 = p_p2[23:16] * coeff[ 7: 0]; // C0
```

程式片段 1

```
assign ry = myr + { 8'b0, myr[16:8] } + 8'h80;
```

程式片段 2

$$\text{Pixel 值} = \frac{C3 \times p\_m1}{256} + \frac{C2 \times p\_z0}{256} + \frac{C1 \times p\_p1}{256} + \frac{C0 \times p\_p2}{256} + 0.5$$

運算完後取整數值...相當於 4 捨 5 入

註 2：單一 pixel 的 RGB 3 原色分開計算，若原訊號為 YUV4：2：2 格式，因 YUV 格式轉換為 RGB 格式為線性方程式，故可如 RGB 方式處理。待規模縮放器處理完後再由 CSC 做轉換格式之動作。

程式片段 2 中之 myr 就是程式片段 1 中 4 項之總和，因 p_m1 ～ p_p2 原為色階值（0 ～ 255），在乘上 0 ～ 255 之係數後必須再除以 256 所得之值才為以權重計算之新 pixels ，故 ry 最後會去掉最後 8 個低位元（亦即除以 256 之意）。程式片段 2 中，加上 myr_[16:8]之意思為補償計算後之新值，因為係數最大為 255，但最後卻需除以 256，故會產生誤差，故補償之。程式片段 2 中，加上 8' h80（=128），就是加上 0.5 之意，若原值小數部分大於 0.5 就會因此而進位。

故在硬體電路中，我們並不需要加入 DSP 就能達到與 DSP 運算後差異極小之結果，因而達到降低成本之目的。

1.3 擴（縮）線之方法

在 Y-buffer 中，使用水平掃瞄線資料組停留或跳躍的方式來達到擴線或縮線之動作，其中，停留或跳躍的方式依據平均分散原則。例如原輸入為 800 條水平線掃瞄，若要擴展為 1024 條水平掃瞄

線，平均每 3.57 條水平掃瞄線就要停留 1 次水平掃瞄線位址，以多計算出 1 條新水平掃瞄線。由於水平掃瞄線的數目均為整數條，因此以另一種演算法，使得擴線的方式能夠精準的在當使用到最後一組（2 條）水平掃瞄線資料時，能夠恰好創造出所需的水平掃瞄線條數。

最後 3 條新水平掃瞄線所使用之輸入水平掃瞄線資料組中之資料數依序為 4 條、3 條、2 條。如倒數第 3 條新水平掃瞄線使用原輸入之{Last 4、Last 3、Last 2、Last 1}；倒數第 2 條新水平掃瞄線使用原輸入之{Last 3、Last 2、Last 1、Last 1}；倒數最後 1 條新水平線使用原輸入之{Last 2、Last 1、Last 1、Last 1}，其目的在使最後 3 條新水掃瞄平線之比重漸漸偏向原輸入之最後 1 條水平線掃瞄訊號。其中 Last n 表示最後之第 n 條水平掃瞄線資料。

1.4 擴線（水平掃瞄線資料組停留）

在 800 條水平線擴展為 1024 條水平線的例子中，我們使用特殊之演算法得到以下之結果：

inc_p ＝ 1、1、1、0、1、1、1、0、1、1、1、1、0、1、1、1、0、1…

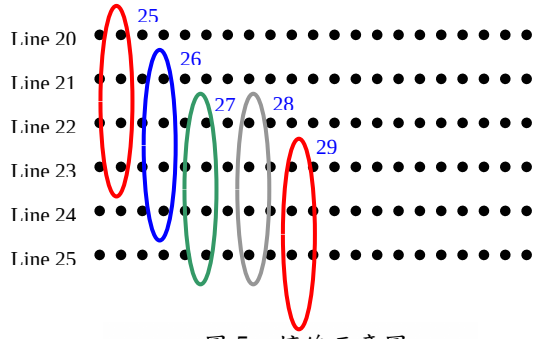


圖 5　擴線示意圖

以上的數字表示下 1 條新水平掃瞄線所要使用的資料組（每 4 條水平線掃瞄資料為 1 組）往下位移多少條水平掃瞄線，1 表示往下位移 1 條水平掃瞄線，0 表示不位移，意即使用原來之資料（水平掃瞄線位址停留…擴線）。其示意圖為圖 5。

圖 5 中，圈圈右上方之數字表示所產生之新水平掃瞄線之次序，如圖中第 3 個圈圈表示由原輸入水平線訊號 Line 22 ~ Line 25 產生 L27 ’new。若以運算式表示，則可以表示如下：

L27 ’new ＝ a*L22 + b*L23 + c*L24 + d* 25

其中，L28 ’ new 與 L27 ’ new 使用相同之 4 條水平線資料作運算，但其使用之係數不同（因程式寫法所造成，避免只是重複同一條線），其運算式如下：

L28 ’new ＝ e*L22 + f*L23 + g*L24 + h*L25

由兩個式子中，可以發現所使用之係數不同，故所新增加之水平線掃瞄部分，並不會與上一條新增之水平掃瞄線相同。水平線掃瞄位址每停留 1 次，就會增加 1 條新的水平掃瞄線。在這種狀況下，當程式中使用到最後一組水平掃瞄線畫面資料時，會剛好停留相當於我們想要增加的水平掃瞄線數目的次數，並且是以平均分散的原則來做擴線之動作。在上圖中，使用了 3 組水平線掃瞄資料卻得到 5 條新水平掃瞄線，因此達到了擴線的目的，同時也兼顧了平均分散之原則。至於係數之部分後面會再加以討論。而擴線時所使用之係數方面則有些許之不同。如同程式片段 3 中，phase 之值在每 1 條水平掃瞄線終結後均會更新，不會有停留之現象。inc_p 之值在每 1 條水平掃瞄線起始時更新。

```
if (ref_d1==`BIT0 && ref==`BIT1)
      inc_p <=#1 total_p[15:12];
if (ref_d2==`BIT1 && ref_d1==`BIT0)
begin
      total[15:12] <=#1 4'b0;
      total[11: 0] <=#1 total_p[11:0];
      phase <=#1 total_p[11:7];
end
```

程式片段 3

1.5 縮線

當水平線進行縮線時會得到以下 inc_p 值：

inc_p ＝ 1、1、1、2、1、1、1、2、1、1…

如圖 6 中之 L18 ’new 與 L19 ’new 所使用的水平掃瞄線資料組相差 2 條水平掃瞄線，其原因在於當進行到 L19 ’new 之計算時，其 inc_p 之值為 2，意即該新水平掃瞄線之資料組與上 1 條新水平掃瞄線資料組相差 2 條水平掃瞄線。由上圖中可以看出當使用了 5 組水平掃瞄線資料組時（每上下相鄰 4 條水平線為 1 組），卻只產生了 4 條新水平掃瞄線，因此也達到了所要求的縮線之效果，同時兼顧平均分散之原則。如同擴線中所提一樣，縮線之狀況下，inc_p[1:0]之值大於 1，因為使用之資料

組並非只是往下位移 1 條水平掃瞄線，而是 2 條水平掃瞄線以上。如同圖 6 中之 L18 'new 與 L19 'new，並使所使用之原輸入水平掃瞄線資料提前被使用完畢，以達到縮線之目的。

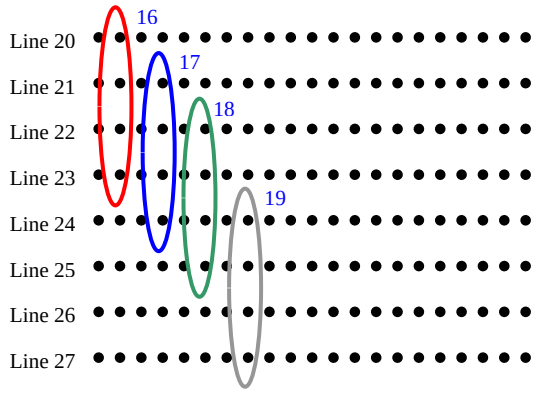


圖 6　縮線示意圖

1.6 規模縮放器所使用之係數

SC-coeff 即 Coefficient of Scaler 之意，在程式中因為會有水平線位址或 pixels 位址（擴、縮點時使用）之停留或跳躍，為了使運算後之顏色分佈能夠平均分散（水平線或 pixels 均同），並盡量與原輸入成比例之狀態，因而利用 SC-coeff 中之係數值來對顏色重新進行調整，而其係數值相當於權重比。為了解釋此行文字，以單色之水平線來解釋（圖中以 Green 之色階來表示），其圖如圖 7。

依程式中之演算法（vratio = 5120），L37 'new ～ L40 'new 所使用之係數各為（0、255、0、0）；（－18、221、58、－6）；（－16、144、143、－16）；（－6、58、221、－18），所得到之新水平線之值各為：

L37 'new ：

（　0x200 + 255x180　+ 0x160　+ 0x140 ）÷ 256 = 179

L38 'new ：

（ -18x180 + 221x160 + 58x140　- 6x120 ）÷ 256 = 154

L39 'new ：

（ -16x160 + 144x140 + 143x120 -16x100 ）÷ 256 = 130

L40 'new ：

（　-6x160　+ 58x140 + 221x120 -18x100 ）÷ 256 = 125

L41 'new：

（　0x140 +　255x120 +　0x100 - 0x80 ）÷ 256 = 120

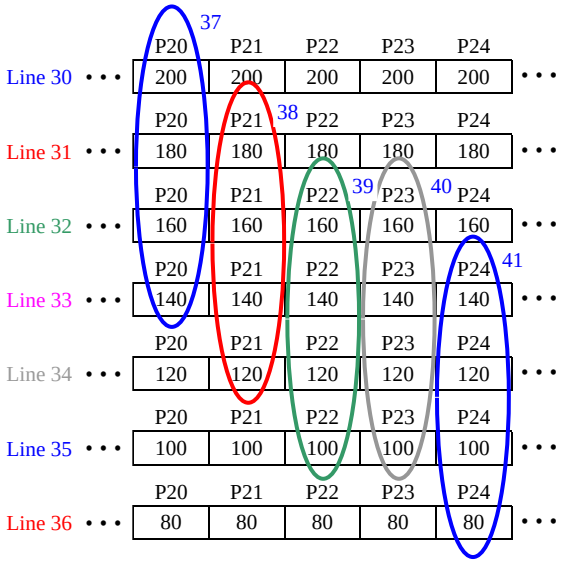


圖 7　運算使用資料示意圖

原 Line 31 ～ Line 34 之值為 180、160、140、120，經過擴線運算後其值為 179、154、130、125、120，其中 125 為經過運算後之新水平線，因此我們可以達到擴線但在其色階值上是人眼可以接受的範圍。

係數表（擴大時使用，另有縮小時之係數表）如表表 1，可以看到當 phase 值越大時，其比重漸漸偏向第 3 項之係數（coeff[16:9]）。而 inc_p 與 phase 均與 vratio 值有關，如程式片段 4 與程式片段 3。在程式片段 4 中 ratio（Y-buffer 中 ratio = vratio）會影響 total_p 之值，而 total_p 會影響 phase 之值。vratio 值是為了針對各種不同之輸出解析度而寫出

phase	coeff[35~0]							
	[35]	[34]~[27]	[26]	[25]~[18]	[17]	[16]~[9]	[8]	[7]~[0]
00	0	0	0	**255**	0	0	0	0
01	1	4	0	**255**	0	4	0	0
02	1	6	0	252	0	9	0	0
03	1	10	0	250	0	16	1	1
04	1	12	0	246	0	23	1	2
⋮								
1B	1	3	0	31	0	241	1	14
1C	1	2	0	23	0	246	1	12
1D	1	1	0	16	0	250	1	10
1E	0	0	0	9	0	252	1	6
1F	0	0	0	4	0	**255**	1	4

表 1　擴線及擴點所使用之係數

```
assign total_p = total + ratio;
```

程式片段 4

1.7 擴（縮）點

在規模縮放器中，擴（縮）完線之後的新水平掃瞄線（尚未進行擴縮點）會將其運算後之 pixels 之新資料交由 X-buffer 與 X-calculator 去做運算以進行擴（縮）點之動作。在 X-calculator 上其運作方式與 Y-calculator 相同，2 者均將由 Y-buffer 或 X-buffer 中所選出之 4 個 pixels 之資料依造權重重新計算出新值。所不同的是 Y-buffer 所取之資料是上下相鄰之 4 條水平掃瞄線上相同水平位置的 pixels，如圖 1；而 X-buffer 所取之資料為同 1 條水平掃瞄線上左右相鄰之 4 個 pixels 之資料，如圖 8。

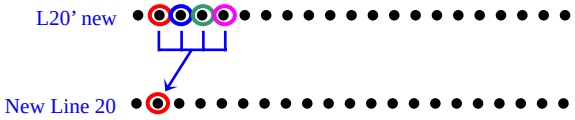


圖 8　擴點示意圖

如圖中所示 L20' new 只是過渡時其中的新水平掃瞄線，尚未經過擴（縮）點之動作。而 New Line 20 為 L20' new 經過擴（縮）點運算後（亦可不擴縮點）之最後輸出水平掃瞄線，往後只剩下諸如亮度、對比、色溫...等之修正，已經與縮放無關。

1.8 X-buffer 資料之取得

原則上 X-buffer 與 Y-buffer 在取得資料之概念上是相同的，但是因為程式上之些許差異，造成其排列之方式略有不同。今程式如下：

```
assign addr_nl0 = addr_l0 + inc_p;
assign addr_nl1 = addr_l1 + inc_p;
```

程式片段 5

```
case(addr_nl0[1:0])
      2'b00:
            begin
                  addr_r0 <=#1 addr_nl0[10:2];
                  addr_r1 <=#1 addr_nl1[10:2];
                  addr_r2 <=#1 addr_nl2[10:2];
                  addr_r3 <=#1 addr_nl3[10:2];
            end
      2'b01:
            begin
                  addr_r0 <=#1 addr_nl3[10:2];
                  addr_r1 <=#1 addr_nl0[10:2];
                  addr_r2 <=#1 addr_nl1[10:2];
                  addr_r3 <=#1 addr_nl2[10:2];
addr_type <=#1 addr_l0[1:0];
addr_l0 <=#1 addr_nl0;
```

程式片段 7

```
      case(addr_type)
2'b00:
      begin
            p_m1 <=#1 po_l0;
            p_z0 <=#1 po_l1;
            p_p1 <=#1 po_l2;
            p_p2 <=#1 po_l3;
      end
2'b01:
      begin
            p_m1 <=#1 po_l1;
            p_z0 <=#1 po_l2;
            p_p1 <=#1 po_l3;
            p_p2 <=#1 po_l0;
```

程式片段 6

程式片段 5 中，addr_nl0 ～ addr_nl3 會隨著 addr_l0 ～ addr_l3 之值與 inc_p 一直做變換。而在程式片段 14 中，addr_nl0 ～ addr_nl3 會 Feedback（回授）給 addr_l0 ～ addr_l3，如此一直循環將造成 addr_nl0 ～ addr_nl3 比 addr_l0 ～ addr_l3 要延遲 1 個 Display Clock 時間。

程式片段 6、7 中，可知程式中利用 addr_l0 來做為選取資料來源之依據。因 addr_l0 之變化為循序性之位移（相當於 addr_type 亦同），故可達到每 1 個 Display Clock 均可取出不同之 pixels 資料組（相鄰之資料組）。

由於在程式中之些許不同，造成 Y-buffer 與 X-buffer 在資料的排列上有些微差異，茲將其新 pixels 之運算之方式舉 1 例列出：

New Pixel 2 = a * P5 + b * P2 + c * P3 + d * P4

New Pixel 3 = e * P2 + f * P3 + g * P4 + h * P5

（假設擴點時，inc_p = 0）

New Pixel 3 = e * P7 + f * P4 + g * P5 + h * P6

（假設縮點時，inc_p = 2）

註 Pn：輸入 X-buffer 之 Pixel n 之值

以圖表示，其資料排列差異如下（不考慮係數）：

Y-buffer：

New Pixel = a *① + b *② + c *③ + d *④

X-buffer：

New Pixel = a *④ + b *① + c *② + d *③

以上即為 X-buffer 與 Y-buffer 在資料取得後排列上之差異。

2. 亮度與對比度調整理論及實現

主要是將輸入訊號，先分離成 RGB 三色，再將各色經由暫存器設定的各三原色的對比度與亮度值去做調整[1]，最後再合併起來，以得到想要顯現出來的影像。

因為亮度是一種程度的表現，所以用加法去表現它，在設定其值時，以－128 至＋127 為其範圍，0 為其內定值。而對比度是一種相對程度的表現，所以用乘法去做，以 0%至 199%為其範圍，100%為其內定值。

所以在運算時，先將輸入影像(單一色)乘以單一原色的對比度，取高位元組 8-bit，如此可做到百分比的操作。例如輸入影像為 p，對比度為 255，則相乘後，取高位元組([14]～[7])，即除 128，所以經對比度運算後的影像為 $p*2^8／128＝p*255／128≒p*199\%$；再多取一位溢位位元([15])與一位有號符號位元([16])，再作四捨五入運算，即將低位元組的 MSB([6])加到高位元組裡，這時的影像就為單一色 10-bit 的資料。

單一原色的亮度為一 8-bit 的位元組，其中 MSB([7])為有號符號位元(即為數值範圍為－128 至＋127)，我們製造一個 10bits 的位元組，高 2bits 為擴位的有號符號，即把第 8 位元([7])的值複製給第 9 位元([8])與第 10 位元([9])，再將此 10-bit 位元組與經對比度處理過的單一色 10-bit 的資料作加法，即得到最後的單一色 10-bit 的資料影像，最後藉由最高兩位元去判斷是否發生溢位情形。

首先將輸入影像乘以對比度，

```
g＝g_in * con_g_in          g_in＝p[23:16]
b＝b_in * con_b_in  其中    b_in＝p[15:8]
r＝r_in * con_r_in          r_in＝p[7:0]
```

取高位元組再多取一位溢位位元與作四捨五入運算，

```
g_con＝g[15:7] + g[6]
b_con＝b[15:7] + b[6]
r_con＝r[15:7] + r[6]
```

將亮度做成一個 10-bit 的位元組，高 2-bit 為擴位的有號符號，

```
bright_g_in＝{reg_bright_g[7], reg_bright_g[7], reg_bright_g}
bright_b_in＝{reg_bright_b[7], reg_bright_b[7], reg_bright_b}
bright_r_in＝{reg_bright_r[7], reg_bright_r[7], reg_bright_r}
```

將經過對比度處理的資料取一位有號符號位元再與將經過亮度處理的資料作加法，

```
g_tmp＝{1'b0,g_con}   +   bright_g_in
b_tmp＝{1'b0,b_con}   +   bright_b_in
r_tmp＝{1'b0,r_con}   +   bright_r_in
```

最後再藉由最高兩位元去判斷是否發生溢位情形，以決定最後的影像輸出。

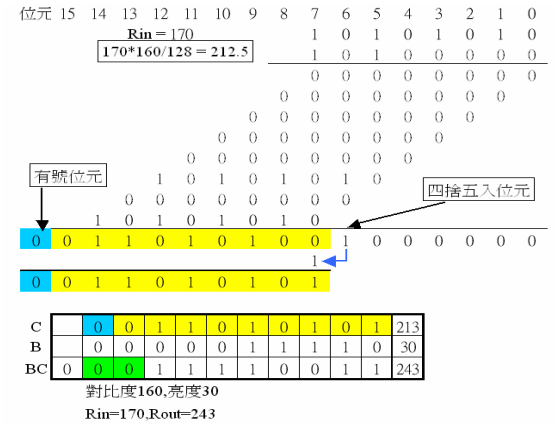
```
g_o＝(g_tmp[9:8]==2'b11)?8'h00:
      (g_tmp[9:8]==2'b00)?g_tmp[7:0]:8'hFF
b_o＝(b_tmp[9:8]==2'b11)?8'h00:
      (b_tmp[9:8]==2'b00)?b_tmp[7:0]:8'hFF
r_o＝(r_tmp[9:8]==2'b11)?8'h00:
      (r_tmp[9:8]==2'b00)?r_tmp[7:0]:8'hFF
```

即若最高兩位元([9:8])為[00]，則輸出為第 8 位元至第 1 位元([7:0])值；若最高兩位元為[01]或[10]，則輸出為 8'hff(全亮)；若最高兩位元為[11]，則輸出為 8'h00(全暗 Blank)。

r_tmp[9:8]	r_tmp[7:0]	有號十進制	r_o
00	00000000 ～ 01111111	0 ～ 127	r_tmp[7:0]
00	10000000 ～ 11111111	128 ～ 255	
01	00000000 ～ 11111111	256 ～ 511	8'hFF
10	00000000 ～ 11111111	-512 ～ -257	
11	00000000 ～ 11111111	-256 ～ -1	8'h00

原圖　Bright(B=30)圖　Contrast(C=160)圖　Bright & Contrast 圖

圖 10　比對實例



2.1 以硬體描述語言實現方法：

基本公式運算：

Rout = Rin * Contrast + Bright
= 170 * 160 / 128 + 30
= 243.5
≒244

舉例說明：

輸入	以硬體描述語言實現方法	基本公式運算
C = 160	Rout = 243	Rout = 244
B = 30		
Rin = 170		

3. γ修正理論與實現

顯示器的強度(Intensity)並非與輸入訊號成正比(非線性關係)，這種非線性特性稱為 Gamma 特性[2]。此非線性曲線可用輸入訊號為 x，亮度呈現 x^r 的函數表示。

所以以下是使用查表法去做γ修正動作。

公式為：

$$\text{Viewing\_gamma} = \frac{1}{\text{Image\_gamma}} * \text{LUT\_gamma} * \text{Display\_gamma}$$

各參數簡述如下：

(1) Image_gamma：為輸入影像的γ值，一般訂為 γ_{NTSC}=2.2，γ_{PAL}=2.8，γ_{RGB}=1，γ_{MAC}=1.8；

(2) Display_gamma：如圖 10 所示，因製程技術的關係，每一製造廠生產出來的顯示器γ值都會不一樣，所以製造廠需提供顯示器γ值，一般訂為 γ_{CRT}=2.5，γ_{LCD}=1.6；

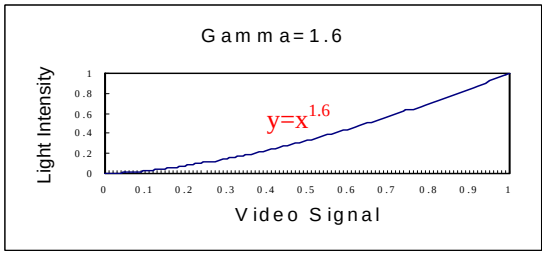


圖 10　LCD Transfer Function in Physics

(3) Viewing_gamma：為最後我們用眼睛去看的結果，理想狀況為 1，即為看到的影像為原始影像，一般會因外在環境的影響，γ值從 1 至 1.5 變化。

(4) LUT_gamma：公式就為

$$\text{LUT\_gamma} = \frac{\text{Image\_gamma}}{\text{Display\_gamma}}$$

如圖 11 所示，為輸入訊號經γ修正器與顯示器後，最後希望看到的是與原輸入影像一樣的畫面，即沒有失真。

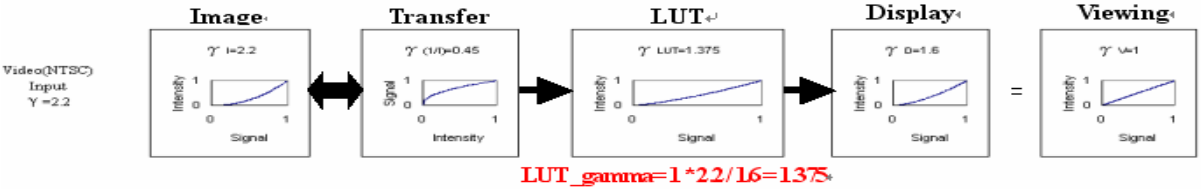


圖 11　GAMMA Transfer Function Diagram

為了實現流暢的色調表現(多色階)，所以輸入單一原色的 8-bit(256 色階)資料，在此模組內轉換成 10-bit(1024 色階)，以產生多色調情形，讓兩相鄰的色階更靠近，即γ修正性能提高，以完成色調失真的補正。此外，另外的目的是為了更能調整、控制中間色調並且能運用自如，表現出自然真實的顏色。

GAMMA 模組主要是將接收的輸入訊號，轉換為要與顯現影像沒有失真情形發生的輸出訊號。GAMMA 區塊模組如圖 12 所示，其中內含三塊三原色記憶體模組，用以儲存三原色之 Gamma Table，如圖 13 所示，其中這三塊記憶體模組皆有做 BIST(Build-in Self Test)功能。操作時依照是否要做γ修正，以決定是否要輸出經γ修正後的影像資料。並且在此模組中，做色階升階的動作，由 8 bits(256 色)升至 10 bits(1024 色)，讓色彩更鮮豔，以供下一級的影像顏色濃淡補正器去做處理。

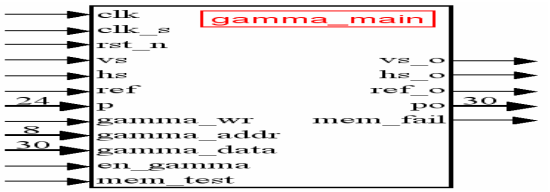


圖 12 GAMMA Block Diagram

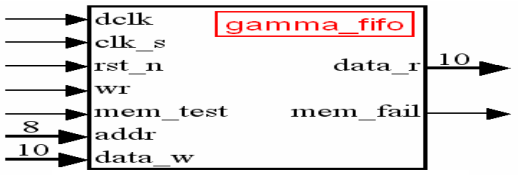


圖 13 GAMMA_FIFO Block Diagram

以硬體描述語言實現方法：

γ 修正器是將 Gamma Table 寫入記憶體中，當啟動 γ 修正時，即將 Gamma Table 中所對應的值輸出；步驟如下：

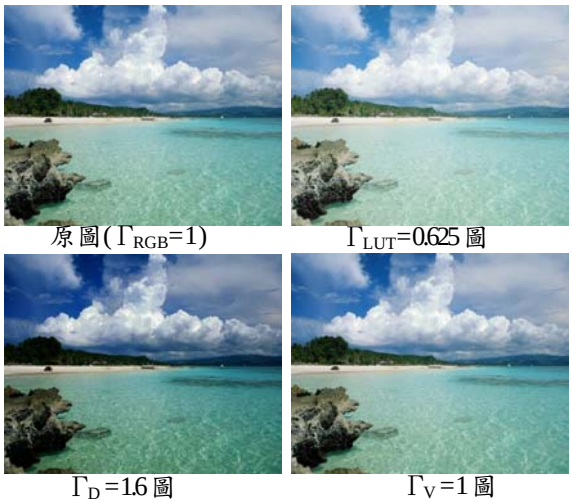
系統 reset 後，由 register 輸入記憶體的位址與數值皆給定初始值 0，這時的 en_gamma 訊號為 0，由 register 輸入 Gamma Table 的位址與數值依序填入記憶體中，即從位址 8'h00 至 8'hFF 依序做初始化的動作。即 Address=8'h01、Data=10'd04，Address=8'h02、Data=10'd08……Address=8'hFF、Data=10'd1020。這時 wr 訊號為 0(寫入記憶體的動作)。當完成初始化後，wr 訊號為 1。

當做完初始化的動作後，等 gamma_wr 訊號從 0 變成 1，就從韌體中，把 Gamma Table 一筆一筆寫入記憶體中(位址 8'h00 至 8'hFF)。這時 wr 訊號為 0，當 gamma_wr 再變成 0 後(即完成填表動作)，wr 訊號就變成 1。由於是將 Gamma Table 一筆一筆寫入記憶體中，所以 en_gamma 訊號為 0。

若不啟動 γ 修正，則輸出為輸入訊號中各色後面加上二位元 0，以組成 30bits 的資料影像，供下一模組去作處理。當要執行 γ 修正時，設定 en_gamma 訊號為 1，記憶體的位址變成由輸入影像資料所給定，即是將輸入影像資料當做位址，去查表(LUT)後，輸出所對應到的數值，它就是經 γ 修正後的影像資料。這時 wr 訊號為 1。

4. 影像顏色濃淡補正理論與實現

在處理影像過後，通常是輸出於顯示器，而顯示器在顯示各色階時又有兩種規格：6-bit 和 8-bit。即是原始輸入影像資料的各色階為 8-bit，經過 GAMMA 升階後為 10-bit，但是輸出至顯示器需為 6-bit 或 8-bit，所以必需降階，



原圖(Γ_{RGB}=1)　Γ_{LUT}=0.625 圖

Γ_D=1.6 圖　Γ_V=1 圖

圖 14　比對實例

而色階降低會造成影像失真現象，即出現輪廓線(contour)或色塊產生。至今有許多研究都希望能將此影響降至最低，一般來說都是使用顫動法(dither)來處理[3]，使顯示器畫面的低階畫質能表現(模擬)出高階的畫質出來，使色塊與輪廓線不會那麼明顯，其中又以貝爾型矩陣(Bayer Matrix)最廣為流傳[4]，所以以下是使用 Bayer Matrix 來做 dithering。

Bayer Matrix 的通式為：

$$D_{2n} = \begin{pmatrix} 4D_n & 4D_n+2U_n \\ 4D_n+3U_n & 4D_n+U_n \end{pmatrix}$$

其中 U_n 為 n*n 的單位矩陣，n≠0，

所以例如 2*2 型矩陣、4*4 型矩陣分別為：

$$D_2 = \begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix}$$

$$D_4 = \begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

而裡面的數值即為其門檻值(threshold value)。再利用 2 值法將各行列值(當成門檻值)來與對應座標點的圖素之濃度做比較，以決定要顯示 0(原色調)或 1(原色調加一階)。如此做可以使輪廓線或色塊變的比較不明顯。

接下來 Dither 再利用視覺色彩與空間色彩

的關係[5]，以少階的色彩模擬出高階畫質出來。

在視覺色彩方面，因為人的視網膜辨識色階的能力約是 1%。如圖 15 所示，所以低於 Code 100 以下的色階必須做 dither 動作，亦即 10 bits 降成 8 bits 情況下，色階是由 1024 階(0.1%)降成 256 階(0.4%)，小於人的辨識能力(1%)，因此可以不做 dither 動作，直接從 10 bits 色階中刪去最低的兩位元，變成 8 bits 輸出，若要確保影像的柔和度，建議做 dithering；當色階從 10 bits(0.1%)降成 6 bits(1.6%)情況時，大於人的辨識能力，一定要做 dither 動作，才會讓影像畫質柔化。再依照該點所在位置的不同，該點所呈現的色彩便不同，那要如何調配，就是 Bayer Matrix 內各元素的門檻值 2 值法，決定該點的紅、綠、藍原色要以原色調或原色調加一階輸出，以少階的色彩模擬出高階的畫質出來。

在空間色彩方面，例如在 1024 階中，我們取紫色([RGB] = [160,32,240])出來塗滿一個區域面積，但是當輸出為 6 bits 的顯示器時，在 64 階中並沒有紫色可以表現，那我們就必須用 36%紅色、9%綠色與 55%藍色來調配模擬成紫色，那要如何調配，也就是 Bayer Matrix 內各元素的排列法。如圖 16 所示，為 1024 階中的紫色，而圖 18 則以 36%紅色、9%綠色與 55%藍色來調配模擬成紫色。

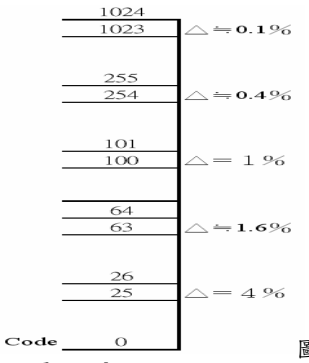


圖 15 The Ratio Between

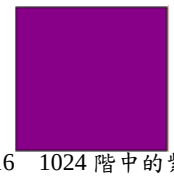


圖 16　1024 階中的紫色

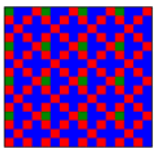


圖 17　64 階中的模擬紫色

Dither 區塊模組如圖 18 所示，此模組的功能有二，一為降階動作，二為模擬作用。當 en_dither=0 時，輸出訊號為輸入訊號(10 bits)去除最低 2 bits，以形成輸出訊號(8-bit)，供給 8-bit 的 Panel 使用；若 en_dither=1 時，則輸出訊號為經過 dithering 修正過後的訊號，供給 6 bits 的 Panel 使用。

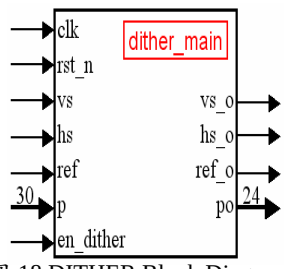


圖 18 DITHER Block Diagram

以硬體描述語言實現方法：

系統 reset 後，先將一個 frame 做畫面切割，每一點給定一個 Bayer Matrix 的表示代碼，如圖 19 所示，其中的

```
M_00 = 16'b0000_0000_0000_0000；   // 0
M_01 = 16'b1111_1111_0000_0000；   // 8
        ⋮
M_32 = 16'b1111_1111_1111_1000；   //13
M_33 = 16'b1111_1000_0000_0000；   // 5
```

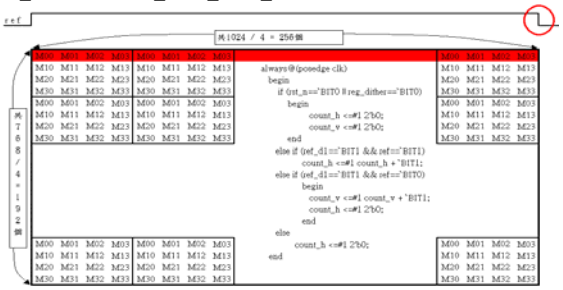


圖 19　畫面切割法(以 15 吋 LCD 為例)

若 en_dither=0，則將輸入影像各色的 10 bits 色階去其最低兩位元，成 8 bits 色階後輸出；若 en_dither=1 時，則將輸入影像各色的 10 bits 色階去其最低四位元後，成 6 bits 色階，拿來作處理，即利用 Bayer Matrix 的 Threshold 功能(門檻條件)，決定該點是否做加一階動作。而門檻條件的設定則是以輸入影像的最低 4 bits 來作判斷，因為 4 bits 的變化可顯現出 16 階的色差，剛好等於 Bayer Matrix 的空間大小以及做降階的需求。

```
4'b0000：   sel_dither_r <=#1 matrix_selector[0]；
4'b0001：   sel_dither_r <=#1 matrix_selector[1]；
        ⋮
4'b1110：  sel_dither_r <=#1 matrix_selector[14]；
4'b1111：  sel_dither_r <=#1 matrix_selector[15]；
```

若不做加一階動作，則以該點的色調輸出；若有做加一階動作，則以該點的色調加一色階輸出，若該點已是最濃狀態，則不做處理。

```
po_r = (inc_r==`BIT1) ? (reg_dither==`BIT1) ?{ p1_r[9:3] +
        9'b10, 3'b0 }：{ p1_r[9:1] + 9'b10, `BIT0 }：p1_r；
if (reg_dither==`BIT0)
        po <=#1 { p[29:22],p[19:12],p[9:2] }；
else    po <=#1 { po_g[9:2],po_b[9:2],po_r[9:2] }；
```

舉例說明：

輸入	不做 dithering 法	做 dithering 法
Rin = 170	Rout = 170	
不做加一階動作		Rout = 168
做加一階動作		Rout = 172

原圖

no_dither

dither

圖 20 比對實例(6 bits)

5. 結論

由於是用 Verilog 硬體描述語言來設計電路，所以可以用模組化(Module)的方式，將硬體分成幾個更小的部分(Partition)來設計，這不但可以使設計的時間縮短，確保每個模組功能正常，除錯(Debug)容易外，設計好的模組還可以重複使用，但要特別注意的是介面(Interface)溝通的問題。本文各影像處理模組將平板顯示器的各項影像處理功能完全涵蓋，不論色彩、色溫、對比度、以及規模的縮放。

本控制器採用的演算法著重於硬體的實現，以硬體電路的精簡與晶片的面積為首要考慮因素，初期系統的效率與表現已有相當的水準。目前已完成演算法之硬體實現與驗證，並完成晶片製作。

未來研究將側重於 Dither 演算法的研究，在學術上，大致分成四類，分別為 Random dither、Ordered dither、Clustered dither 以及 Error diffusion。

參考文獻

[1] 陳連春，數位化影像技術入門，建興出版社，2001 年，p.94～117

[2] Charles A. Poynton，A Technical Introduction to Digital Video，John Wiley & Sons，1996 年，p.1～184

[3] Kang , Henry R.，Digital color halftoning，SPIE/IEEE series on imaging science & engineering，1999，p.205～302

[4] 楊武智，最新影像數位信號處理基礎，全華科技圖書股份有限公司，1995 年，p.68～75

[5] 楊武智，高畫質電視影像技術—HDTV 技術，全華科技圖書股份有限公司，1993 年，p.1～58