RAiO

RA8889

應用程式介面

May 21, 2025

RAiO Technology Inc.

章節介紹

序言

在這份文件裡，我們將為您介紹 8 種關於 RA8889 的功能。有 MCU 資料寫入與顯示資料輸入格式、幾何圖形繪圖、DMA(Direct Memory Access)、BTE(Block Transfer Engine)、PIP(Picture in Picture)、文字以及 PWM(Pulse Width Modulation)功能…等等，並提供關於這些功能的應用程式介面。

在一開始，您需要在 Userdef.h 這個檔案裡面，搭配您的環境，選擇您所使用的 **MCU 通訊介面協定**、**LCD Panel**、**MCU 資料寫入與顯示資料輸入格式**、**集通字庫**與 **PLL 設定值**(SDRAM CLK、Core CLK、Pixel CLK)。詳細的說明與設定，如 PLL initialization 或 LCD initialization，請參考附錄 1/2 的說明。

MCU 通訊介面協定:RA8889 提供了 Parallel 8080、Parallel 6800、SPI-3 wire、SPI-4 wire、IIC 等五種 MCU 通訊介面協定。

SDRAM:RA8889 內建 128Mbits SDRAM。

MCU 資料寫入與顯示資料輸入格式:

8-bit MCU, 8bpp mode、8-bit MCU, 16bpp mode、8-bit MCU, 24bpp mode、16-bit MCU, 16bpp mode、16-bit MCU, 24bpp mode 1、16-bit MCU, 24bpp mode 2.

集通字庫:RA8889 可以經由外接集通字庫，在顯示上支援多種字體。

PLL 設定值(SDRAM CLK、Core CLK、Pixel CLK):SDRAM CLK(最大 166MHz)、Core CLK(最大 120MHz)、Pixel CLK(最大 80MHz)。

SDRAM CLK > Core CLK

在 16 位元色深情況下: Core CLK > 1.5*Pixel CLK

在 24 位元色深情況下: Core CLK > 2 * Pixel CLK

第一章: MCU 資料寫入與顯示資料輸入格式

概要:

RA8889 提供六種模式給 MCU 對 SDRAM 做資料寫入:

1.8-bit MCU 介面, 8bpp color depth mode (RGB 3:3:2)

2.8-bit MCU 介面, 16bpp color depth mode (RGB 5:6:5)

3.8-bit MCU 介面, 24bpp color depth mode (RGB 8:8:8)

4.16-bit MCU 介面, 16bpp color depth mode (RGB 5:6:5)

5.16-bit MCU 介面, 24bpp color depth mode 1 (RGB 8:8:8)

6.16-bit MCU 介面, 24bpp color depth mode 2 (RGB 8:8:8)

本章節將會幫助使用者簡單的瞭解如何使用這六種模式

1.1.1:8-bit MCU 介面, 8bpp color depth mode (RGB 3:3:2)

Order	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	R_0^7	R_0^6	R_0^5	G_0^7	G_0^6	G_0^5	B_0^7	B_0^6
2	R_1^7	R_1^6	R_1^5	G_1^7	G_1^6	G_1^5	B_1^7	B_1^6
3	R_2^7	R_2^6	R_2^5	G_2^7	G_2^6	G_2^5	B_2^7	B_2^6
4	R_3^7	R_3^6	R_3^5	G_3^7	G_3^6	G_3^5	B_3^7	B_3^6
5	R_4^7	R_4^6	R_4^5	G_4^7	G_4^6	G_4^5	B_4^7	B_4^6
6	R_5^7	R_5^6	R_5^5	G_5^7	G_5^6	G_5^5	B_5^7	B_5^6

表 1：8-bit MCU, 8bpp mode 資料格式

API :

使用 8 bits MCU 介面以及 8bpp color depth. MCU 寫資料到 SDRAM.

```
void MPU8_8bpp_Memory_Write
(
unsigned short x //x of coordinate
,unsigned short y // y of coordinate
,unsigned short w //width
,unsigned short h //height
,const unsigned char *data //8bit data
)
```

範例 :

```
MPU8_8bpp_Memory_Write(0,0,128,128 ,gImage_8);
MPU8_8bpp_Memory_Write(200,0,128,128 ,gImage_8);
MPU8_8bpp_Memory_Write(400,0,128,128 ,gImage_8);
```

gImage_8 是大小 128x128 的圖資，格式為 MCU 8 位元介面，色深 8 位元時所使用。

LCD 顯示示意圖:

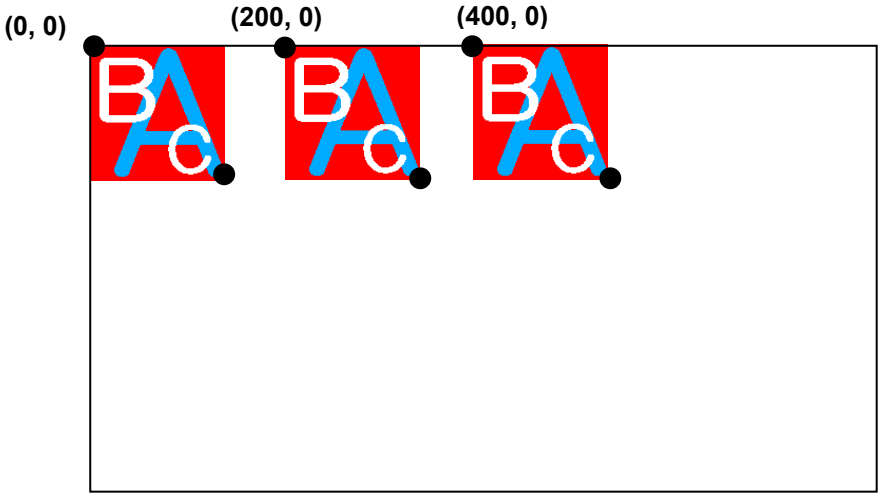


圖 1.1：使用 8 bits MCU 介面以及 8bpp color depth. MCU 寫資料到 SDRAM.

1.1.2:8-bit MCU 介面, 16bpp color depth mode (RGB 5:6:5)

Order	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	G_0^4	G_0^3	G_0^2	B_0^7	B_0^6	B_0^5	B_0^4	B_0^3
2	R_0^7	R_0^6	R_0^5	R_0^4	R_0^3	G_0^7	G_0^6	G_0^5
3	G_1^4	G_1^3	G_1^2	B_1^7	B_1^6	B_1^5	B_1^4	B_1^3
4	R_1^7	R_1^6	R_1^5	R_1^4	R_1^3	G_1^7	G_1^6	G_1^5
5	G_2^4	G_2^3	G_2^2	B_2^7	B_2^6	B_2^5	B_2^4	B_2^3
6	R_2^7	R_2^6	R_2^5	R_2^4	R_2^3	G_2^7	G_2^6	G_2^5

表 2 : 8-bit MCU, 16bpp mode 資料格式

API :

使用 8 bits MCU 介面以及 16bpp color depth. MCU 寫資料到 SDRAM.

```
void MPU8_16bpp_Memory_Write
(
unsigned short x //x of coordinate
,unsigned short y // y of coordinate
,unsigned short w //width
,unsigned short h //height
,const unsigned char *data //8bit data
)
```

範例:

```
MPU8_16bpp_Memory_Write (0,0,128,128,gImage_16);
MPU8_16bpp_Memory_Write (200,0,128,128,gImage_16);
MPU8_16bpp_Memory_Write (400,0,128,128,gImage_16);
```

gImage_16 是大小 128x128 的圖資，格式為 MCU 8 位元介面，色深 16 位元時所使用。

LCD 顯示示意圖:

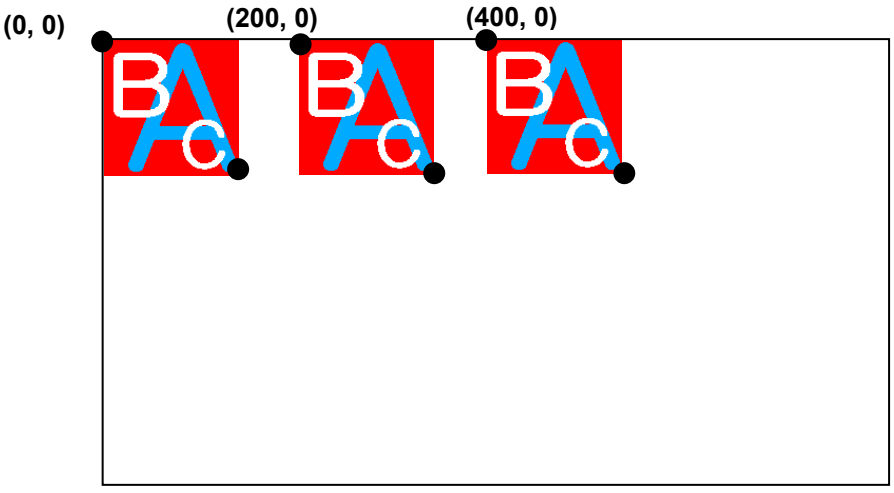


圖 1.2 : 使用 8 bits MCU 介面以及 16bpp color depth. MCU 寫資料到 SDRAM.

1.1.3 :8-bit MCU 介面, 24bpp color depth mode (RGB 8:8:8)

Order	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	B_0^7	B_0^6	B_0^5	B_0^4	B_0^3	B_0^2	B_0^1	B_0^0
2	G_0^7	G_0^6	G_0^5	G_0^4	G_0^3	G_0^2	G_0^1	G_0^0
3	R_0^7	R_0^6	R_0^5	R_0^4	R_0^3	R_0^2	R_0^1	R_0^0
4	B_1^7	B_1^6	B_1^5	B_1^4	B_1^3	B_1^2	B_1^1	B_1^0
5	G_1^7	G_1^6	G_1^5	G_1^4	G_1^3	G_1^2	G_1^1	G_1^0
6	R_1^7	R_1^6	R_1^5	R_1^4	R_1^3	R_1^2	R_1^1	R_1^0

表 3 : 8-bit MCU, 24bpp mode 資料格式

API :

使用 8 bits MCU 介面以及 24bpp color depth. MCU 寫資料到 SDRAM.

```
void MPU8_24bpp_Memory_Write
(
unsigned short x //x of coordinate
,unsigned short y // y of coordinate
,unsigned short w //width
,unsigned short h //height
,const unsigned char *data //8bit data
)
```

範例:

```
MPU8_24bpp_Memory_Write (0,0,128,128 ,gImage_24);
MPU8_24bpp_Memory_Write (200,0,128,128,gImage_24);
MPU8_24bpp_Memory_Write (400,0,128,128,gImage_24);
```

gImage_24 是大小 128x128 的圖資，格式為 MCU 8 位元介面，色深 24 位元時所使用。

LCD 顯示示意圖:

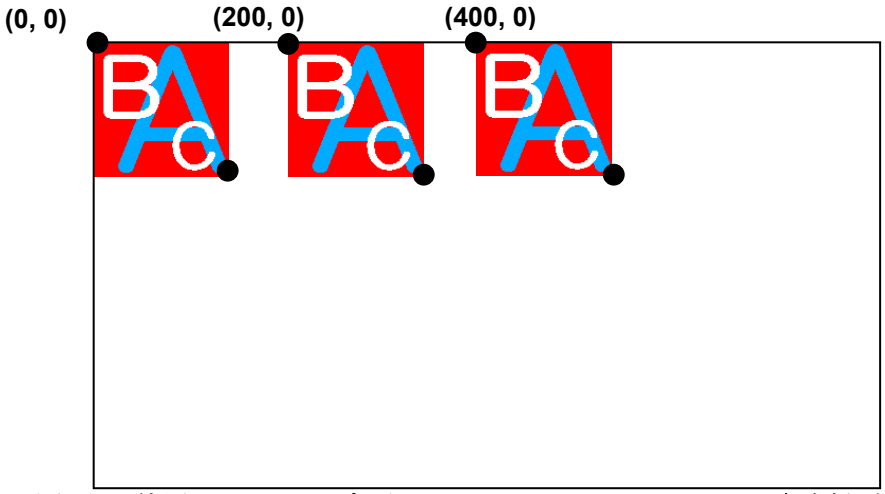


圖 1.3 : 使用 8 bits MCU 介面以及 24bpp color depth. MCU 寫資料到 SDRAM.

1.1.4:16-bit MCU 介面, 16bpp color depth mode (RGB 5:6:5)

Order	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	R_0^7	R_0^6	R_0^5	R_0^4	R_0^3	G_0^7	G_0^6	G_0^5	G_0^4	G_0^3	G_0^2	B_0^7	B_0^6	B_0^5	B_0^4	B_0^3
2	R_1^7	R_1^6	R_1^5	R_1^4	R_1^3	G_1^7	G_1^6	G_1^5	G_1^4	G_1^3	G_1^2	B_1^7	B_1^6	B_1^5	B_1^4	B_1^3
3	R_2^7	R_2^6	R_2^5	R_2^4	R_2^3	G_2^7	G_2^6	G_2^5	G_2^4	G_2^3	G_2^2	B_2^7	B_2^6	B_2^5	B_2^4	B_2^3
4	R_3^7	R_3^6	R_3^5	R_3^4	R_3^3	G_3^7	G_3^6	G_3^5	G_3^4	G_3^3	G_3^2	B_3^7	B_3^6	B_3^5	B_3^4	B_3^3
5	R_4^7	R_4^6	R_4^5	R_4^4	R_4^3	G_4^7	G_4^6	G_4^5	G_4^4	G_4^3	G_4^2	B_4^7	B_4^6	B_4^5	B_4^4	B_4^3
6	R_5^7	R_5^6	R_5^5	R_5^4	R_5^3	G_5^7	G_5^6	G_5^5	G_5^4	G_5^3	G_5^2	B_5^7	B_5^6	B_5^5	B_5^4	B_5^3

表 4：16-bit MCU, 16bpp mode 資料格式

API：

使用 16 bits MCU 介面以及 16bpp color depth. MCU 寫資料到 SDRAM.

```
void MPU16_16bpp_Memory_Write
(
unsigned short x //x of coordinate
,unsigned short y // y of coordinate
,unsigned short w //width
,unsigned short h //height
,const unsigned short *data //16-bit data
)
```

範例:

```
MPU16_16bpp_Memory_Write (0,0,128,128,pic1616);
MPU16_16bpp_Memory_Write (200,0,128,128,pic1616);
MPU16_16bpp_Memory_Write (400,0,128,128,pic1616);
```

pic1616 是大小 128x128 的圖資，格式為 MCU 16 位元介面，色深 16 位元時所使用。

LCD 顯示示意圖:

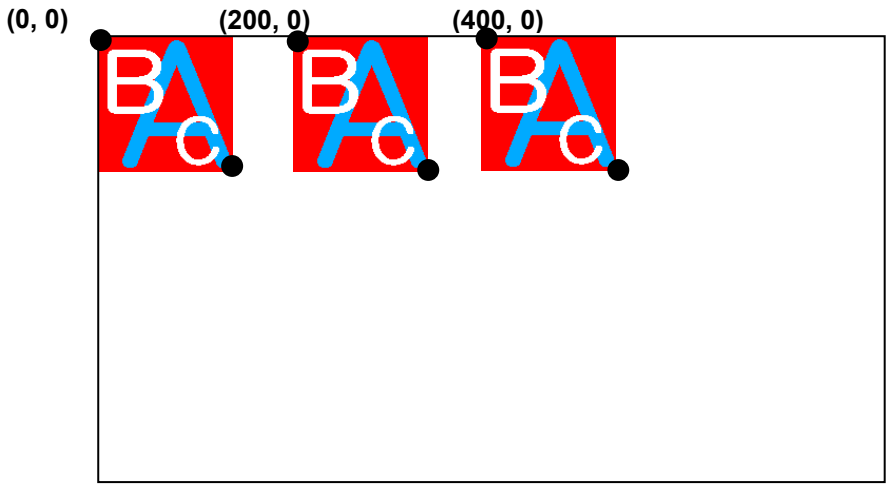


圖 1.4：使用 16 bits MCU 介面以及 16bpp color depth. MCU 寫資料到 SDRAM.

1.1.5 :16-bit MCU 介面, 24bpp color depth mode 1 (RGB 8:8:8)

Order	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	G_0^7	G_0^6	G_0^5	G_0^4	G_0^3	G_0^2	G_0^1	G_0^0	B_0^7	B_0^6	B_0^5	B_0^4	B_0^3	B_0^2	B_0^1	B_0^0
2	B_1^7	B_1^6	B_1^5	B_1^4	B_1^3	B_1^2	B_1^1	B_1^0	R_0^7	R_0^6	R_0^5	R_0^4	R_0^3	R_0^2	R_0^1	R_0^0
3	R_1^7	R_1^6	R_1^5	R_1^4	R_1^3	R_1^2	R_1^1	R_1^0	G_1^7	G_1^6	G_1^5	G_1^4	G_1^3	G_1^2	G_1^1	G_1^0
4	G_2^7	G_2^6	G_2^5	G_2^4	G_2^3	G_2^2	G_2^1	G_2^0	B_2^7	B_2^6	B_2^5	B_2^4	B_2^3	B_2^2	B_2^1	B_2^0
5	B_3^7	B_3^6	B_3^5	B_3^4	B_3^3	B_3^2	B_3^1	B_3^0	R_2^7	R_2^6	R_2^5	R_2^4	R_2^3	R_2^2	R_2^1	R_2^0
6	R_3^7	R_3^6	R_3^5	R_3^4	R_3^3	R_3^2	R_3^1	R_3^0	G_3^7	G_3^6	G_3^5	G_3^4	G_3^3	G_3^2	G_3^1	G_3^0

表 5 : 16-bit MCU, 24bpp mode 1 資料格式

API :

使用 16 bits MCU 介面以及 24bpp color depth mode1. MCU 寫資料到 SDRAM.

```
void MPU16_24bpp_Mode1_Memory_Write
(
unsigned short x //x of coordinate
,unsigned short y // y of coordinate
,unsigned short w //width
,unsigned short h //height
,const unsigned short *data //16-bit data
)
```

範例:

```
MPU16_24bpp_Mode1_Memory_Write(0,0,128,128,pic16241);
MPU16_24bpp_Mode1_Memory_Write(200,0,128,128,pic16241);
MPU16_24bpp_Mode1_Memory_Write(400,0,128,128,pic16241);
```

pic16241 是大小 128x128 的圖資，格式為 MCU 16 位元介面，色深 24 位元的 mode1 下所使用。

LCD 顯示示意圖:

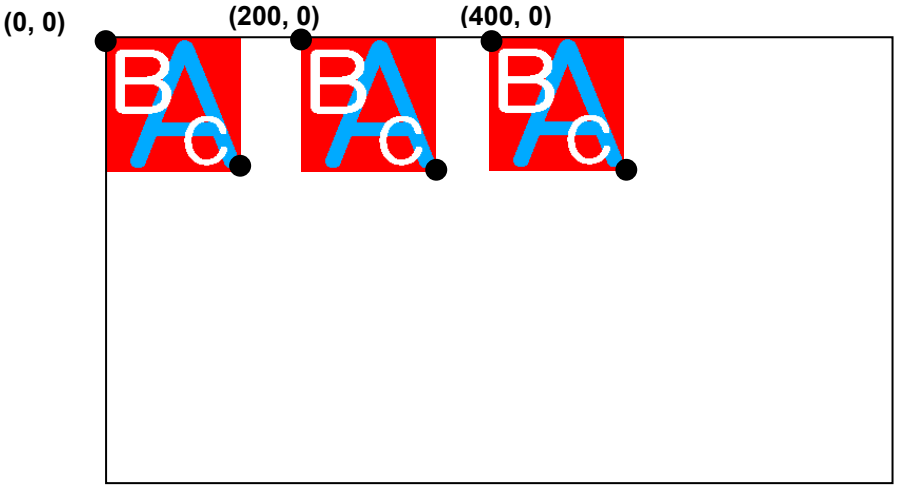


圖 1.5 : 使用 16 bits MCU 介面以及 24bpp color depth mode1. MCU 寫資料到 SDRAM

1.1.6:16-bit MCU 介面, 24bpp color depth mode 2 (RGB 8:8:8)

Order	Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	G_0^7	G_0^6	G_0^5	G_0^4	G_0^3	G_0^2	G_0^1	G_0^0	B_0^7	B_0^6	B_0^5	B_0^4	B_0^3	B_0^2	B_0^1	B_0^0
2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	R_0^7	R_0^6	R_0^5	R_0^4	R_0^3	R_0^2	R_0^1	R_0^0
3	G_1^7	G_1^6	G_1^5	G_1^4	G_1^3	G_1^2	G_1^1	G_1^0	B_1^7	B_1^6	B_1^5	B_1^4	B_1^3	B_1^2	B_1^1	B_1^0
4	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	R_1^7	R_1^6	R_1^5	R_1^4	R_1^3	R_1^2	R_1^1	R_1^0
5	G_2^7	G_2^6	G_2^5	G_2^4	G_2^3	G_2^2	G_2^1	G_2^0	B_2^7	B_2^6	B_2^5	B_2^4	B_2^3	B_2^2	B_2^1	B_2^0
6	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	R_2^7	R_2^6	R_2^5	R_2^4	R_2^3	R_2^2	R_2^1	R_2^0

表 6：16-bit MCU, 24bpp mode 2 資料格式

API：

使用 16 bits MCU 介面以及 24bpp color depth mode2. MCU 寫資料到 SDRAM.

```
void MPU16_24bpp_Mode2_Memory_Write
(
unsigned short x //x of coordinate
,unsigned short y // y of coordinate
,unsigned short w //width
,unsigned short h //height
,const unsigned short *data //16-bit data
)
```

範例:

```
MPU16_24bpp_Mode2_Memory_Write(0,0,128,128,pic1624);
MPU16_24bpp_Mode2_Memory_Write(200,0,128,128,pic1624);
MPU16_24bpp_Mode2_Memory_Write(400,0,128,128,pic1624);
```

pic1624 是大小 128x128 的圖資，格式為 MCU 16 位元介面，色深 24 位元的 mode2 下所使用。

LCD 顯示示意圖:

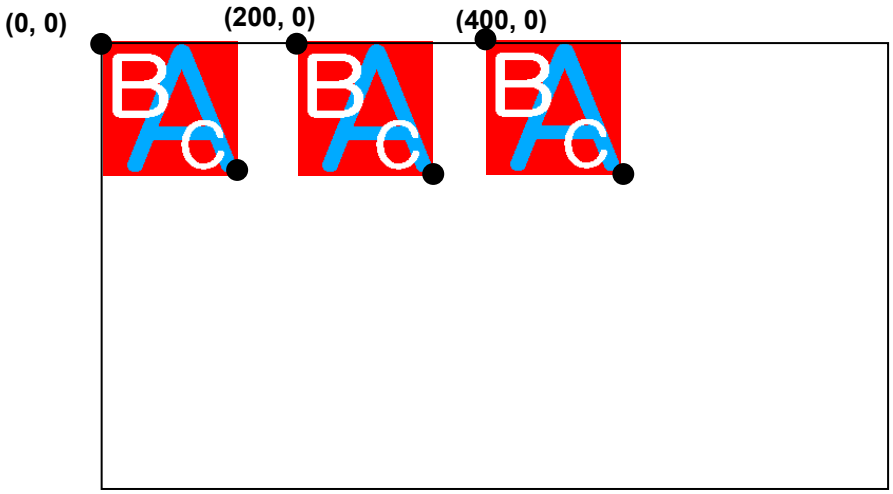


圖 1.6：使用 16 bits MCU 介面以及 24bpp color depth mode2. MCU 寫資料到 SDRAM.

第二章 ： 幾何圖形繪圖

概要:

在許多的人機介面應用中，時常需要去顯示一些幾何圖形用來當作按鍵，感應器或者是其他的幾何圖形標誌。當使用者想要透過 MCU 韌體來達到這個需求時，使用者會耗費很多的系統資源或心力，去做數學運算或者是圖庫編纂。

RA8889 提供了幾何繪圖引擎，使用者只要下一些指令，就可以輕易的在 TFT-LCD 上達到幾何圖形顯示。使用者僅需選擇幾何圖形的顏色與它是不是應該被填滿。這份應用說明將會幫助使用者去了解，如何使用 RA8889 的幾何圖形繪圖功能。

2.1: 橢圓/圓 輸入

RA8889 提供繪製橢圓/圓形繪圖的功能，讓使用者以簡易或者低速的 MCU 就可以在 TFT LCD 模組上迅速做畫圓的工作。首先，先設定橢圓/圓形的中心點 REG[7Bh~7Eh]，其次，設定橢圓的長軸與短軸或圓的半徑 REG[77h~7Ah]，設定橢圓/圓的顏色 REG[D2h~D4h]，設定畫橢圓/圓 REG[76h]Bit5=0 和 Bit4=0，然後啟動繪圖 REG[76h]Bit7 = 1，RA8889 就會自動將橢圓/圓圖形寫入顯示資料用的記憶體，此外，使用者可以經由設定 REG[76h]Bit6= 1，來畫出實心橢圓/圓。

注意:橢圓/圓的中心必須要在 active windows 裡面
寫入程序請參照下圖:

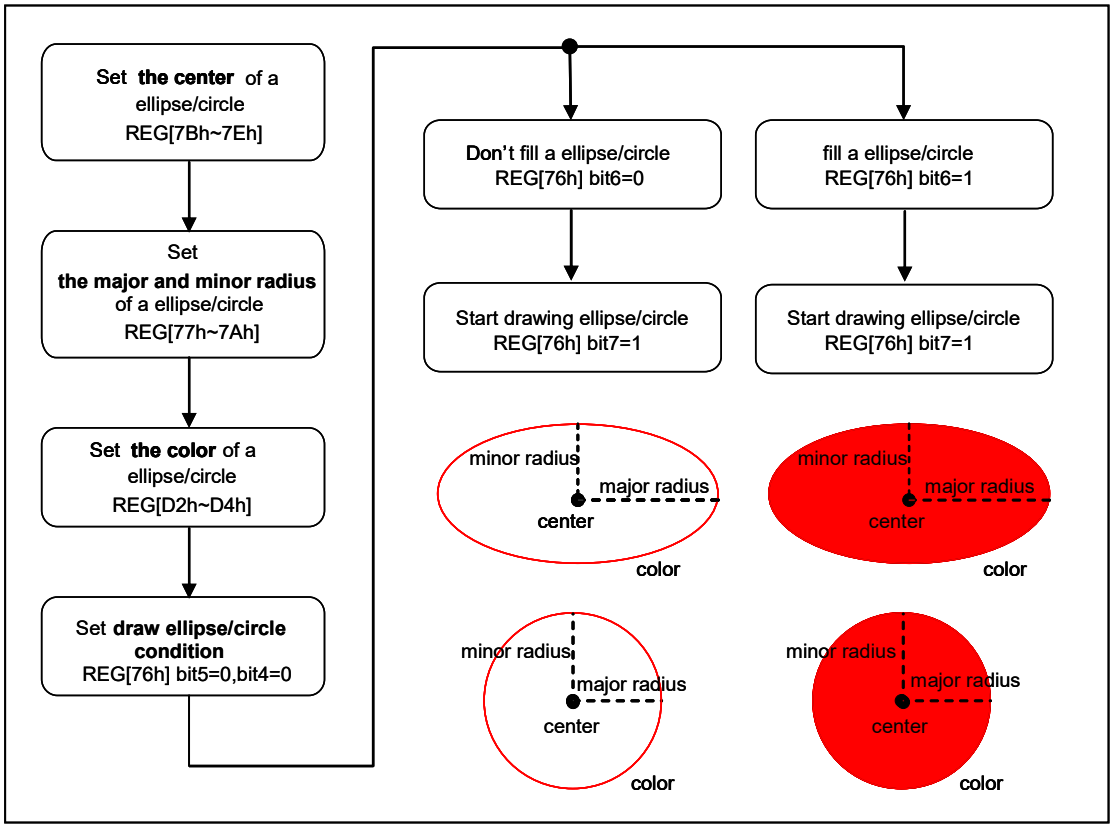


圖 2.1 畫圓形(填滿)與橢圓形(填滿)的程式流程圖

畫橢圓形或圓形的 **API :**

```
void Draw_Circle
(
unsigned long ForegroundColor //ForegroundColor: Set Draw Circle or Circle Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short R //Circle Radius
)
void Draw_Circle_Fill
(
unsigned long ForegroundColor
/*ForegroundColor: Set Draw Circle or Circle Fill color
ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short R //Circle Radius
)

void Draw_Ellipse
(
unsigned long ForegroundColor //ForegroundColor : Set Draw Ellipse or Ellipse Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Ellipse
,unsigned short Y_R // Radius Length of Ellipse
)
void Draw_Ellipse_Fill
(
unsigned long ForegroundColor //ForegroundColor : Set Draw Ellipse or Ellipse Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
```

```
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Ellipse
,unsigned short Y_R // Radius Length of Ellipse
)
```

範例 1(畫圓形):

```
Active_Window_XY(0,0);
Active_Window_WH(1366,768);                    //set[(0,0) to (1366,768)] can draw graph
+
//When color depth = 8bpp
Draw_Circle(0xfc,500,50,50);
Draw_Circle_Fill(0xfc,650,50,50);
Or
//When color depth = 16bpp
Draw_Circle(0xffe0,500,50,50);
Draw_Circle_Fill(0xffe0,650,50,50);
Or
//When color depth = 24bpp
Draw_Circle(0xffff00,500,50,50);
Draw_Circle_Fill(0xffff00,650,50,50);
```

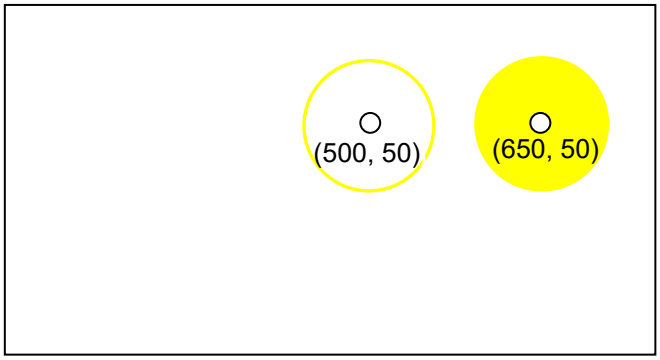


圖 2.2： 畫一個顏色為黃色的圓，圓心位置(500,50)，半徑 = 50，
和畫一個被黃色填滿的圓型，圓心位置(650,50)，半徑 = 50。

範例 2(畫橢圓形):

```
Active_Window_XY(0,0);
Active_Window_WH(1366,768);              //set[(0,0) to (1366,768)] can draw graph
+
//When color depth = 8bpp
Draw_Ellipse(0x1f,100,200,100,50);
Draw_Ellipse_Fill(0x1f,350,200,100,50);
Or
//When color depth = 16bpp
Draw_Ellipse(0x07ff,100,200,100,50);
Draw_Ellipse_Fill(0x07ff,350,200,100,50);
Or
//When color depth = 24bpp
Draw_Ellipse(0x00ffff,100,200,100,50);
Draw_Ellipse_Fill(0x00ffff,350,200,100,50);
```

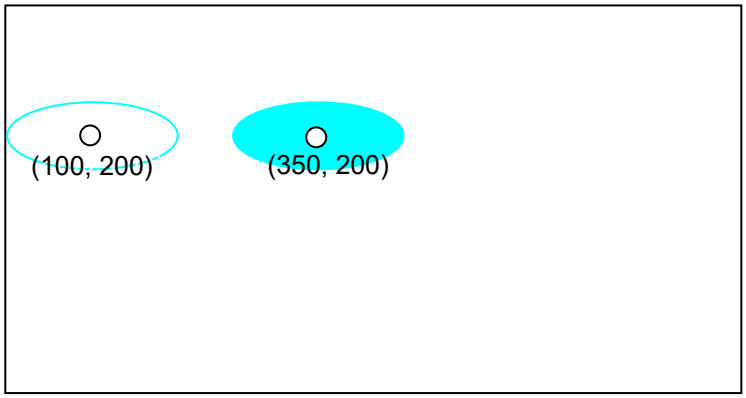


圖 2.3：畫一個青色的圓，圓心位置(100,200)、X 軸半徑 = 100、Y 軸半徑 = 50，和畫一個青色的實心圓，圓心位置(350,200)、X 軸半徑 = 100、Y 軸半徑 = 50。

2.2: 曲線輸入

RA8889 提供曲線繪圖功能，讓使用者以簡易或低速的 MCU 就可以在 TFT 模組上畫曲線。先設定曲線的中心點 REG[7Bh~7Dh]，曲線的長軸與短軸 REG[77h~7Ah]，曲線的顏色 REG[D2h~D4h]，曲線的相關參數為 REG[76h] Bit5=0 與 Bit4=1，REG[76h] Bit[1:0] 是橢圓的曲線部分，然後啟動繪圖設定 REG[76h] Bit7 = 1，RA8889 就會自動將曲線的圖形寫入顯示資料的記憶體。此外，使用者可以經由設定 REG[76h]Bit6 = 1 來畫出實心曲線。

注意曲線的中心必須要在 active windows 裡面
寫入程序請參照下圖:

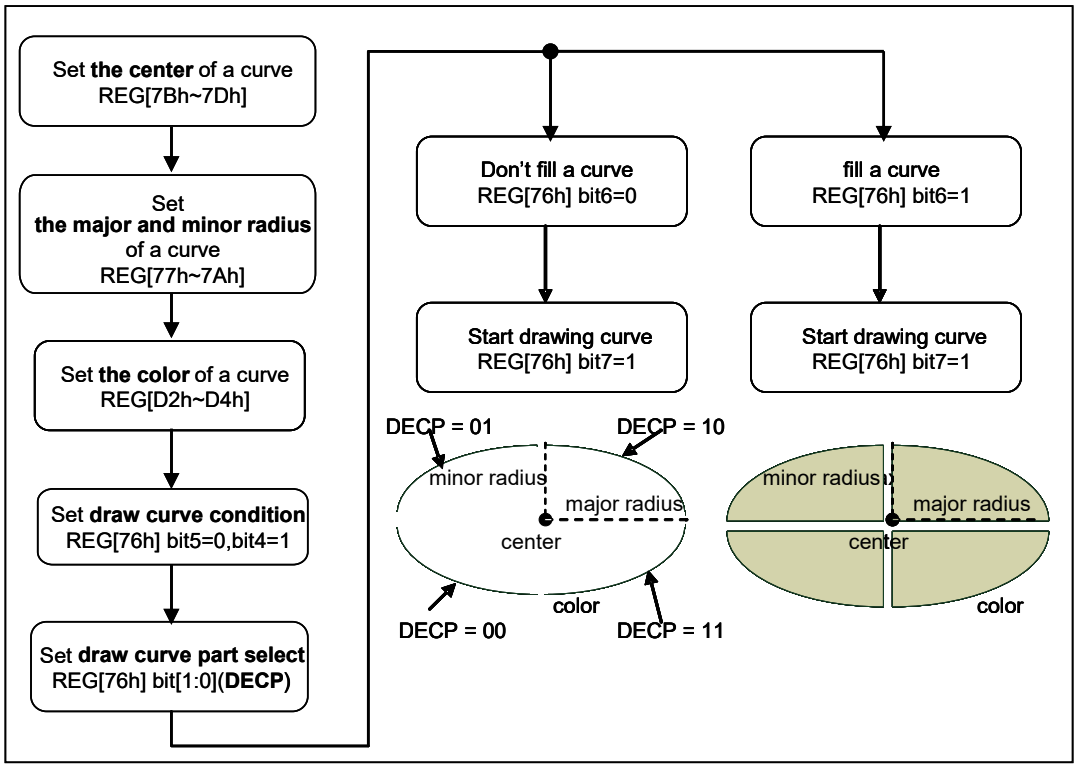


圖 2-4 畫圓弧與圓弧形填滿的程式流程圖

畫圓弧與畫圓弧填滿功能的 **API：**

```
void Draw_Left_Up_Curve
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)

void Draw_Right_Down_Curve
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)

void Draw_Right_Up_Curve
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)


void Draw_Left_Down_Curve
```

```
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)

void Draw_Left_Up_Curve_Fill
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)

void Draw_Right_Down_Curve_Fill
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)
```

```
void Draw_Right_Up_Curve_Fill
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)

void Draw_Left_Down_Curve_Fill
(
unsigned long ForegroundColor //ForegroundColor: Set Curve or Curve Fill color
/*ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short XCenter //coordinate X of Center
,unsigned short YCenter //coordinate Y of Center
,unsigned short X_R // Radius Width of Curve
,unsigned short Y_R // Radius Length of Curve
)
```

範例:

```
Active_Window_XY(0,0);                          //set[(0,0) to (1366,768)] can draw graph
Active_Window_WH(1366,768);
+
//When color deepth = 8bpp
Draw_Left_Up_Curve(0xe3,550,190,50,50);
Draw_Right_Down_Curve(0xff,560,200,50,50);
Draw_Right_Up_Curve(0xff,560,190,50,50);
Draw_Left_Down_Curve(0x1c,550,200,50,50);
Draw_Left_Up_Curve_Fill(0xe3,700,190,50,50);
Draw_Right_Down_Curve_Fill(0xff,710,200,50,50);
Draw_Right_Up_Curve_Fill(0xff,710,190,50,50);
Draw_Left_Down_Curve_Fill(0x1c,700,200,50,50);
Or
//When color deepth = 16bpp
Draw_Left_Up_Curve(0xf11f,550,190,50,50);
Draw_Right_Down_Curve(0xffff,560,200,50,50);
Draw_Right_Up_Curve(0xffff,560,190,50,50);
Draw_Left_Down_Curve(0x07e0,550,200,50,50);
Draw_Left_Up_Curve_Fill(0xf11f,700,190,50,50);
Draw_Right_Down_Curve_Fill(0xffff,710,200,50,50);
Draw_Right_Up_Curve_Fill(0xffff,710,190,50,50);
Draw_Left_Down_Curve_Fill(0x07e0,700,200,50,50);
Or
//When color deepth = 24bpp
Draw_Left_Up_Curve(0xff00ff,550,190,50,50);
Draw_Right_Down_Curve(0xffffff,560,200,50,50);
Draw_Right_Up_Curve(0xffffff,560,190,50,50);
Draw_Left_Down_Curve(0x00ff00,550,200,50,50);
Draw_Left_Up_Curve_Fill(0xff00ff,700,190,50,50);
Draw_Right_Down_Curve_Fill(0xffffff,710,200,50,50);
Draw_Right_Up_Curve_Fill(0xffffff,710,190,50,50);
Draw_Left_Down_Curve_Fill(0x00ff00,700,200,50,50);
```

本範例程式包含 8 個步驟:

1. Draw a pink upper left curve,Center(550,190), X_R = 50,Y_R=50
2. Draw a white lower right curve,Center(560,200), X_R = 50,Y_R=50
3. Draw a white upper right curve,Center(560,190), X_R = 50,Y_R=50
4. Draw a green lower left curve,Center(550,200), X_R = 50,Y_R=50
5. Draw a pink upper left curve fill,Center(700,190), X_R = 50,Y_R=50
6. Draw a white lower right curve fill,Center(710,200), X_R = 50,Y_R=50
7. Draw a white upper right curve fill,Center(710,190), X_R = 50,Y_R=50
8. Draw a green lower left curve fill,Center(700,200), X_R = 50,Y_R=50

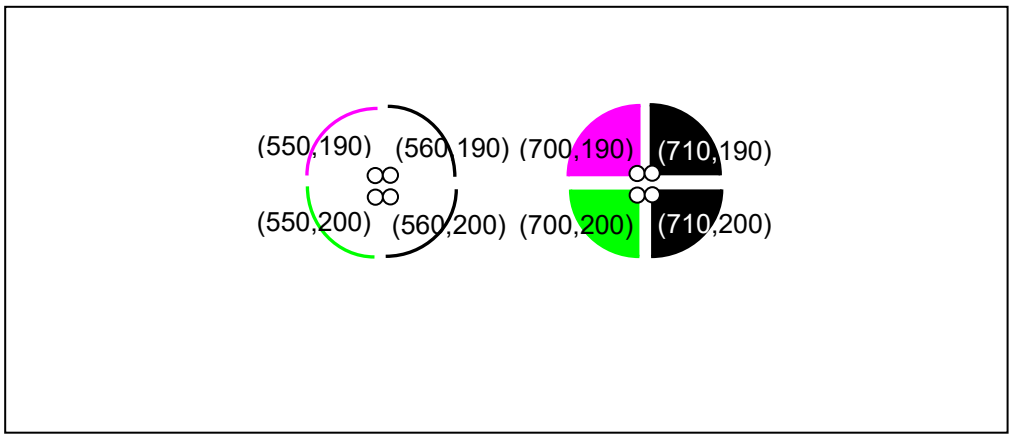


圖 2.5：畫圓弧與圓弧形填滿的 LCD 顯示示意圖

2.3: 方形輸入

RA8889 提供方形繪圖功能，讓使用者以簡易或低速的 MCU 就可以在 TFT 模組上畫方形。先設定方形的起始點 REG[68h~6Bh]與結束點 REG[6Ch~6Fh]，方形的顏色 REG[D2h~D4h]，然後設定畫方形設定 REG[76h]Bit5=1，Bit4=0 和啟動繪圖 REG[76h]Bit7 = 1，RA8889 就會自動將方形的圖形寫入顯示資料的記憶體，此外，使用者可以藉由設定 REG[76h]Bit6 = 1 來畫出實心方形。

注意:方形的起始點與結束點必須要在 active windows 裡面
寫入程序請參照下圖:

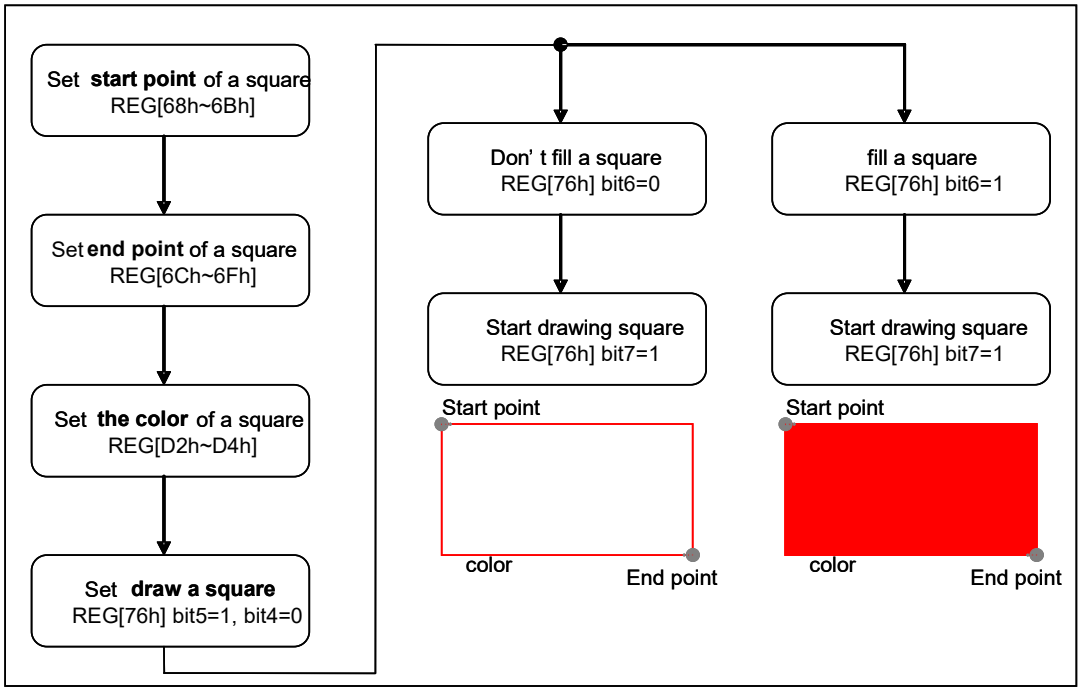


圖 2.6：畫方形與方形填滿的程式流程圖

畫方形的 **API:**

```
void Draw_Square
(
unsigned long ForegroundColor
/*ForegroundColor: Set Curve or Curve Fill color. ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 //Y of point2 coordinate
)

void Draw_Square_Fill
(
unsigned long ForegroundColor
/*ForegroundColor: Set Curve or Curve Fill color. ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 //Y of point2 coordinate
)
```

範例 ：

```
Active_Window_XY(0,0);
Active_Window_WH(1366,768);                    //set[(0,0) to (1366,768)] can draw graph
+
//when color depth = 8bpp
Draw_Square(0xe0,50,300,150,400);
Draw_Square_Fill(0xe0,200,300,300,400);
//Or
//When color deepth = 16bpp
Draw_Square(0xf800,50,300,150,400);
Draw_Square_Fill(0xf800,200,300,300,400);
//Or
//When color deepth = 24bpp
Draw_Square(0xff0000,50,300,150,400);
Draw_Square_Fill(0xff0000,200,300,300,400);
```

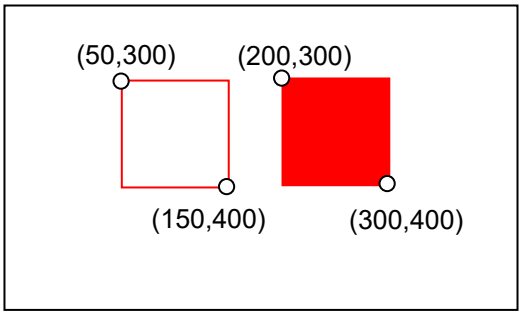


圖 2.7：從點(50,300)到點(150,400)，畫一個紅色的正方形，
以及從點(200,300)到點(300,400)，畫一個紅色被填滿的方形。

2.4:畫直線功能

RA8889 支援畫直線的繪圖功能，讓使用者以簡易或低速的 MCU 就可以在 TFT 模組上畫線，先設定起始點 REG[68h~6Bh]，結束點 REG[6Ch~6Fh]和線的顏色 REG[D2h~D4h]，然後設定畫線參數 REG[67h]Bit1 = 0，和啟動繪圖 REG[67h]Bit7 = 1，RA8889 就會自動將線的圖形寫入顯示資料的記憶體。

注意: :線的起始點與結束點必須要在 active windows 裡面
寫入程序請參照下圖:

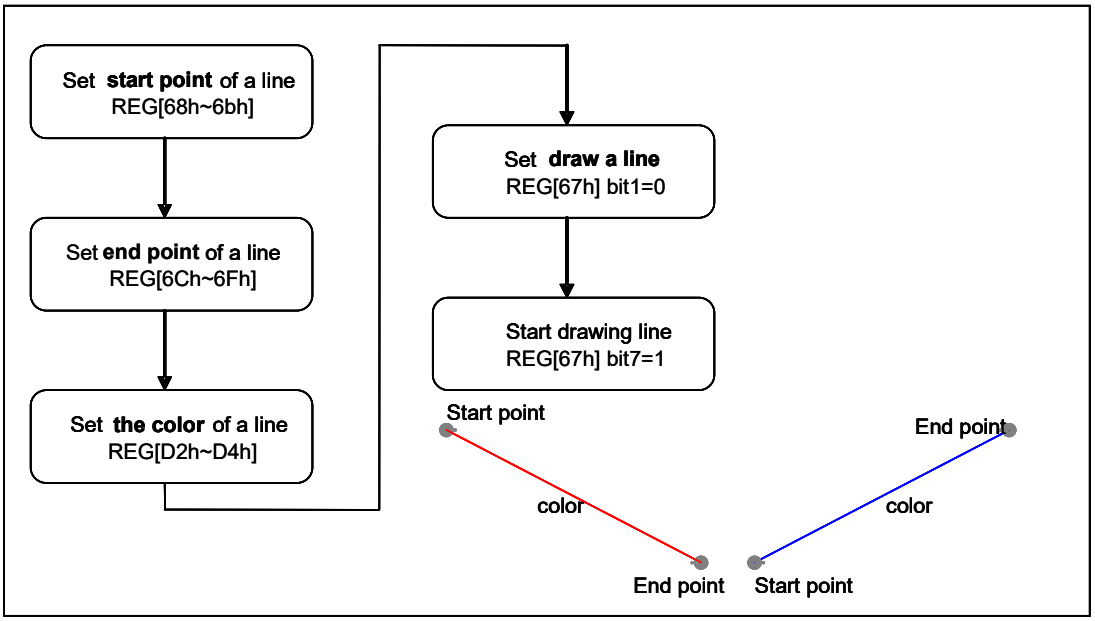


圖 2.8: 畫直線的程式流程圖

畫直線的 **API:**

```
void Draw_Line
(
unsigned long LineColor
/*LineColor : Set Draw Line color. Line Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 // Y of point2 coordinate
)
```

範例 ：

```
Active_Window_XY(0,0);
Active_Window_WH(1366,768);                    //set[(0,0) to (1366,768)] can draw graph
+
//When color deepth = 8bpp
Draw_Line(0xe0,10,10,800,700);
//Or
//When color deepth = 16bpp
Draw_Line(0xf800,10,10,800,700);
//Or
//When color deepth = 24bpp
Draw_Line(0xff0000,10,10,800,700);
```

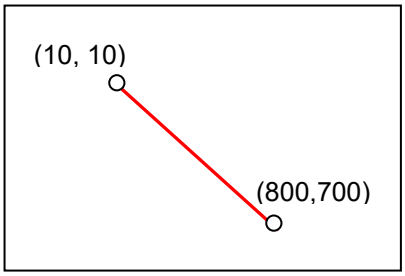


圖 2.9：從點(10,10)到點(800,700)，畫一條紅色的直線。

實際範例操作如下列影片連結:

https://www.youtube.com/watch?v=LCtWjkU4WQE

http://v.youku.com/v_show/id_XMTM0MzE0MjczNg==.html?f=26097824&from=y1.2-3.4.7

2.5:畫三角形的功能

RA8889 支援三角形的繪圖功能，讓使用者以簡易或低速的 MCU 就可以在 TFT 模組上畫三角形。先設定三角的第 0 點 REG[68h~6Bh]、第 1 點 REG[6Ch~6Fh]、第 2 點 REG[70h~73h]和三角形的顏色 REG[D2h~D4h]，設定畫三角形的參數 REG[67h] Bit1 = 1，然後啟動繪圖 REG[67h] Bit7 = 1，RA8889 就會自動將三角形的圖形寫入顯示資料的記憶體。此外，使用者可以透過設定 REG[67h] Bit5 = 1 來畫出實心三角形。

注意:三角形的第 0 點第 1 點與第 2 點與結束點必須要在 active windows 裡面
寫入程序請參照下圖:

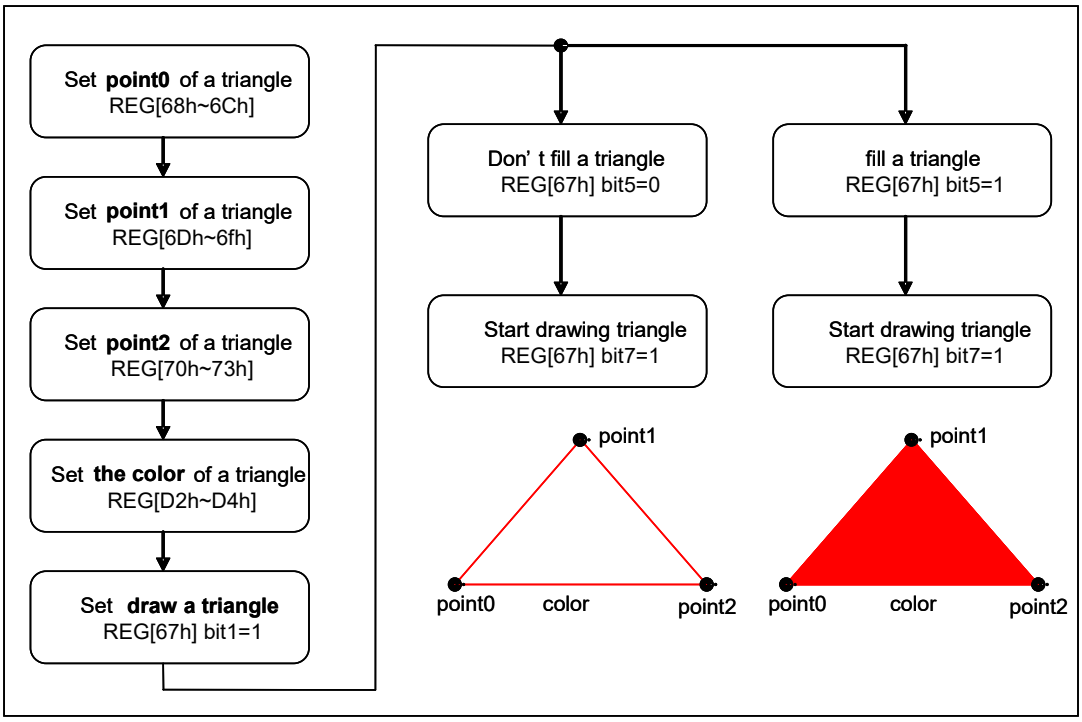


圖 2.10： 畫三角形與三角形填滿的程式流程圖

畫三角形的 **API:**

```
void Draw_Triangle
(
unsigned long ForegroundColor
/*ForegroundColor: Set Draw Triangle color. ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 //Y of point2 coordinate
,unsigned short X3 //X of point3 coordinate
,unsigned short Y3 //Y of point3 coordinate
)
void Draw_Triangle_Fill
(
unsigned long ForegroundColor
/*ForegroundColor: Set Draw Triangle color. ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 //Y of point2 coordinate
,unsigned short X3 //X of point3 coordinate
,unsigned short Y3 //Y of point3 coordinate
)
```

範例:

```
Active_Window_XY(0,0);
Active_Window_WH(1366,768);              //set[(0,0) to (1366,768)] can draw graph
+
//When color deepth = 8bpp
Draw_Triangle(0x07,150,0,150,100,250,100);
Draw_Triangle_Fill(0x03,300,0,300,100,400,100);
//Or
//When color deepth = 16bpp
Draw_Triangle(0x001f,150,0,150,100,250,100);
Draw_Triangle_Fill(0x001f,300,0,300,100,400,100);
//Or
//When color deepth = 24bpp
Draw_Triangle(0x0000ff,150,0,150,100,250,100);
Draw_Triangle_Fill(0x0000ff,300,0,300,100,400,100);
```

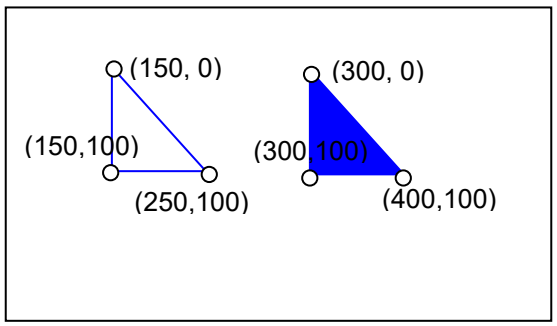


圖 2.11： 用(150,0)、(150,100)、(250,100)三點畫一個藍色的三角形，以及用(300,0) 、(300,100)、(400,100)三點畫一個藍色被填滿的三角形。

2.6:畫圓角方形的功能

RA8889 支援圓角方形繪圖功能，讓使用者以簡易或低速的 MCU 就可以在 TFT 模組上畫方形。經由設定方形的起始點 REG[68h~6Bh]與結束點 REG[6Ch~6Fh]，圓角的長軸與短軸 REG[77h~7Ah]，圓角方形的顏色 REG[D2h~D4h]，然後設定畫方形設定 REG[76h]Bit5=1，Bit4=1 和啟動繪圖 REG[76h]Bit7 = 1，RA8889 就會自動將圓角方形的圖形寫入顯示資料的記憶體，此外，使用者可以藉由設定 REG[76h]Bit6 = 1 來畫出實心的圓角方形。

提醒 1：(結束點 X – 起始點 X) 必須要大於(2*長軸 + 1) 且 (結束點 Y – 起始點 Y) 必須要大於(2*短軸 + 1)

提醒 2：起始點與結束點需要在 active windows 內

寫入程序請參照下圖:

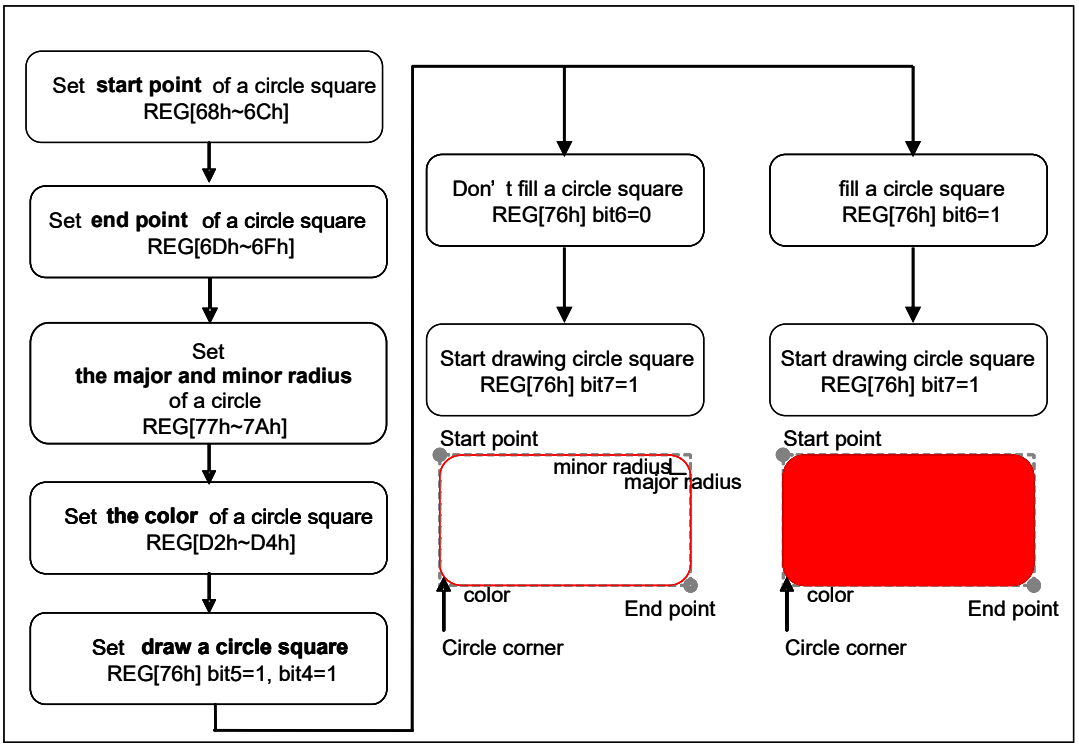


圖 2.12：畫圓角方形與圓角方形填滿的程式流程圖

畫圓角方形的 **API:**

```
void Draw_Circle_Square
(
unsigned long ForegroundColor
/*ForegroundColor : Set Circle Square color. ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 //Y of point2 coordinate
,unsigned short X_R //Radius Width of Circle Square
,unsigned short Y_R //Radius Length of Circle Square
)

void Draw_Circle_Square_Fill
(
unsigned long ForegroundColor
/*ForegroundColor : Set Circle Square color. ForegroundColor Color dataformat :
ColorDepth_8bpp : R3G3B2、ColorDepth_16bpp : R5G6B5、ColorDepth_24bpp : R8G8B8*/
,unsigned short X1 //X of point1 coordinate
,unsigned short Y1 //Y of point1 coordinate
,unsigned short X2 //X of point2 coordinate
,unsigned short Y2 //Y of point2 coordinate
,unsigned short X_R //Radius Width of Circle Square
,unsigned short Y_R //Radius Length of Circle Square
)
```

範例:

```
Active_Window_XY(0,0);
Active_Window_WH(1366,768);
+
//When color deepth = 8bpp
Draw_Circle_Square(0xe0,450,300,550,400,20,30);
Draw_Circle_Square_Fill(0xe0,600,300,700,400,20,30);
Or
//When color deepth = 16bpp
Draw_Circle_Square(0xf800,450,300,550,400,20,30);
Draw_Circle_Square_Fill(0xf800,600,300,700,400,20,30);
Or
//When color deepth = 24bpp
Draw_Circle_Square(0xff0000,450,300,550,400,20,30);
Draw_Circle_Square_Fill(0xff0000,600,300,700,400,20,30);
```

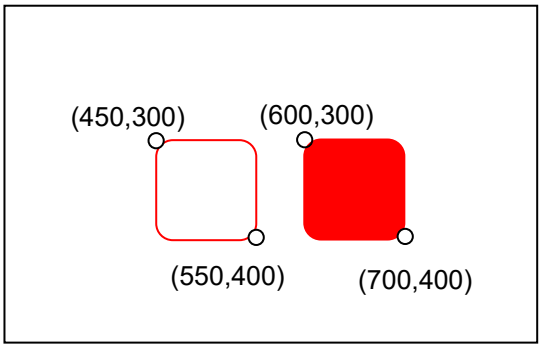


圖 2.13：長軸為 20，短軸為 30，從點(450,300)到點(550,400)，畫一個紅色的圓角方形，以及長軸為 20，短軸為 30，從點(600,300)到點(700,400)，畫一個紅色被填滿的圓角方形

第三章 ： 串列式 Flash/ROM 控制單元

RA8889 內建了串列式 Flash/ROM 的介面，來支援下列的傳輸模式: 4-BUS 正常讀取(Normal Read)、5-BUS 快速讀取 (FAST Read)、雙倍模式 0 (Dual mode 0)、雙倍模式 1 (Dual mode 1)、模式 0 (Mode 0)、 模式 3 (Mode 3)以及 Quad Mode。

串列式 Flash/ROM 記憶體功能可用在文字模式 (FONT Mode)、DMA 模式(直接記憶存取模式)。文字模式意指外部串列式 Flash/ROM 記憶體被當成字體點陣圖的來源。為了支援文字字體，RA8889 可與專業的字體供應商 — 上海集通公司的 FONT ROM 相容。DMA 模式意指串列式 Flash/ROM 可當作 DMA (Direct Memory Access) 的資料來源。使用者可以透過此模式，加快資料傳送到顯示記憶體(Display RAM) 的速度。

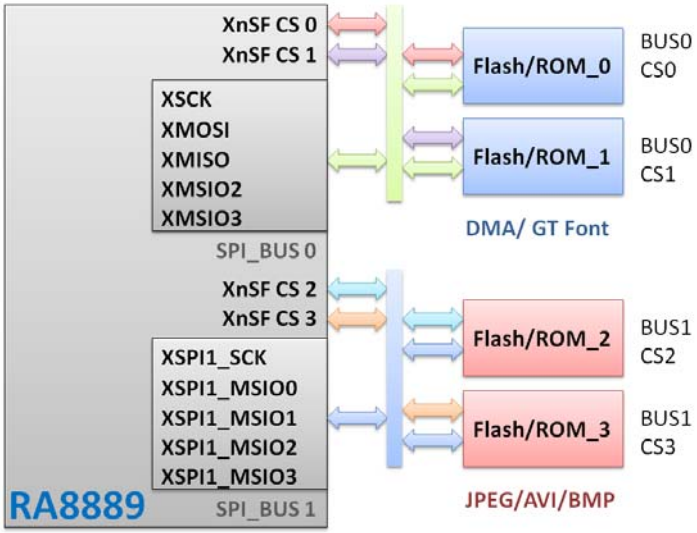


圖 3.1: RA8889 串列式 Flash/ROM 系統

REG [B7h] BIT[3:0]	Read Command code
000xb	1x read command code – 03h. Normal read speed. Single data input on xmiso. Without dummy cycle between address and data.
010xb	1x read command code – 0Bh. To some serial flash provide faster read speed. Single data input on xmiso. 8 dummy cycles inserted between address and data.
1x0xb	1x read command code – 1Bh. To some serial flash provide fastest read speed. Single data input on xmiso. 16 dummy cycles inserted between address and data.
xx10b	2x read command code – 3Bh. Interleaved data input on xmiso & xmosi. 8 dummy cycles inserted between address and data phase. (mode 0)
xx11b	2x read command code – BBh. Address output & data input interleaved on xmiso & xmosi. 4 dummy cycles inserted between address and data phase. (mode 1)

REG [B6h] BIT[7:6]	Read Command code
01b	4x read command code – 6Bh. Address output & data input interleaved on xmiso & xmosi & xsio2 & xsio3.
10b	4x read command code – EBh. Address output & data input interleaved on xmiso & xmosi & xsio2 & xsio3

3.1.1:DMA block mode

DMA block mode 是一個將圖片從外部快閃記憶體搬(Serial Flash Memory)移到 RA8889 用的顯示記憶體(SDRAM)，DMA 功能的使用的單位是像素(pixel)。關於 DMA 功能的流程圖，請參照以下的敘述。

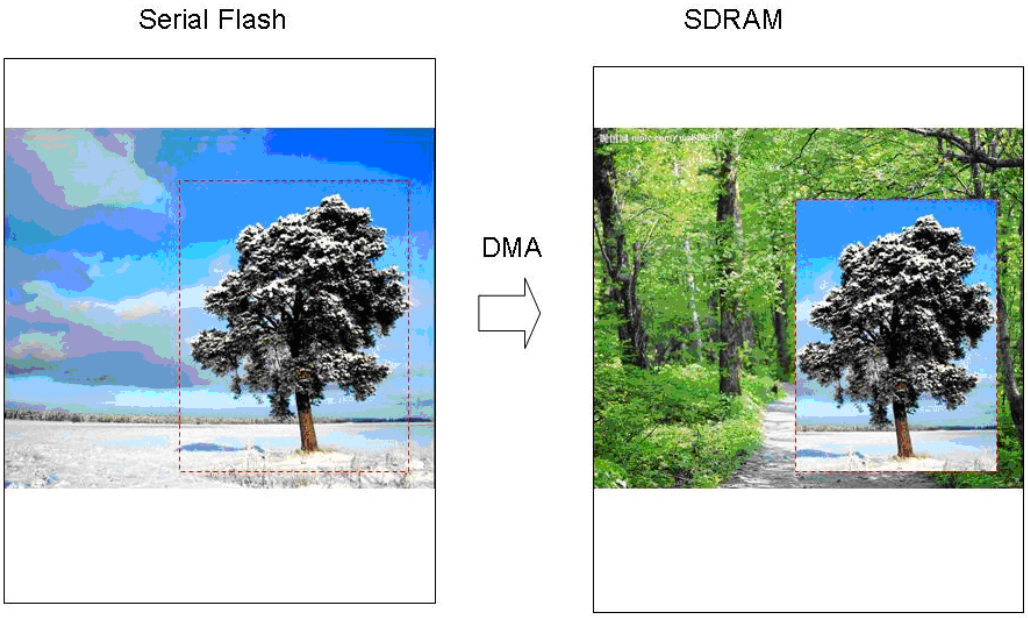


圖 3.2 : DMA 功能

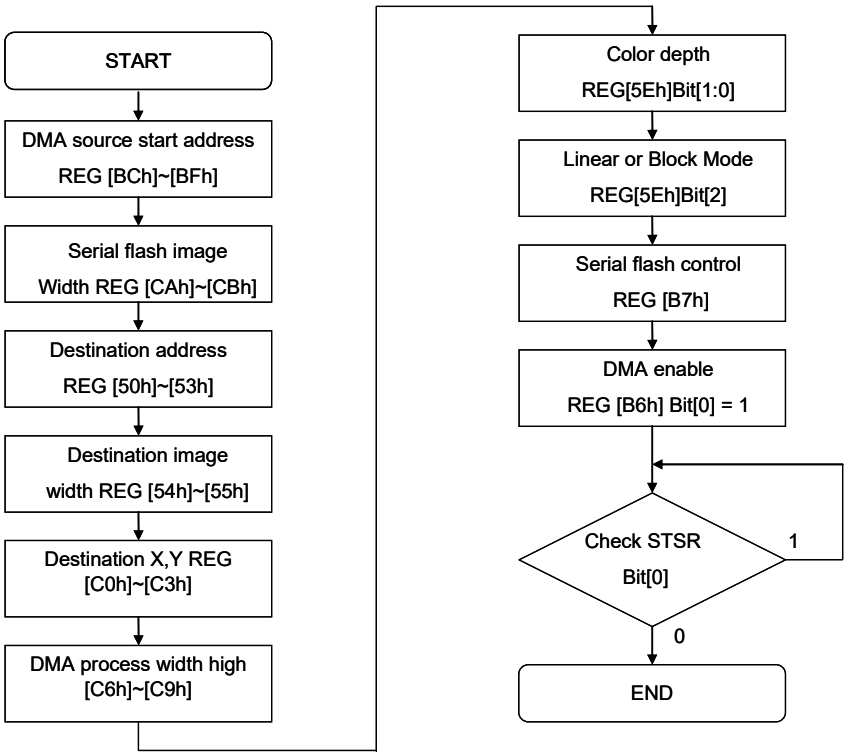


圖 3.3 : 啟動 DMA 功能的程式流程 – With Check Flag operation (STSR.0)

3.1.2 DMA 功能的 API 在 LCD 上的顯示結果:

我們提供下列 7 組 API:

NOR FLASH :

```
void SPI_NOR_initial_DMA
(
char mode//SPI mode :
0:Single_03h,1:Single_0Bh,2:Single_1Bh,3:Dual_3Bh,4:Dual_BBh,5:Quad_6Bh,6:Quad_EBh
,char BUS //BUS : 0 = Use BUS0, 1 = Use BUS1
,char SCS //SCS : 0 = Use SCS0, 1 = Use SCS1 ,2 = Use SCS2 ,3 = Use SCS3
,char flash //0 : MXIC , 1 : Winbond
,unsigned char addr_24b_32b //flash 24bit or 32bit addr
)

void DMA_24bit
(
,unsigned char Clk //Clk : SPI Clock = System Clock /{(Clk)*2} , SPI CLK recommend <=90MHz
,unsigned short X1 //X of DMA Coordinate
,unsigned short Y1 //Y of DMA Coordinate
,unsigned short X_W //DMA Block width
,unsigned short Y_H //DMA Block height
,unsigned short P_W //DMA Picture width
,unsigned long Addr //DMA Source Start address
)

void DMA_32bit
(
,unsigned char Clk //Clk : SPI Clock = System Clock /{(Clk)*2} , SPI CLK recommend <=90MHz
,unsigned short X1 //X of DMA Coordinate
,unsigned short Y1 //Y of DMA Coordinate
,unsigned short X_W //DMA Block width
,unsigned short Y_H //DMA Block height
,unsigned short P_W //DMA Picture width
,unsigned long Addr //DMA Source Start address
)
```

```
void SPI_NOR_DMA_png
(
unsigned long dma_page_addr //dma pic addr in flash
,unsigned long pic_buffer_Layer //pic_buffer_Layer : read pic buffer in sdram
,unsigned long Show_pic_Layer //Show_pic_Layer : write pic into sdram addr
,unsigned int picture_Width
,unsigned int picture_Height
)
```

NAND FLASH :僅支援 W25N01GV

```
void SPI_NAND_initial_DMA
(
char SCS //SCS : 0 = Use SCS0, 1 = Use SCS1 ,2 = Use SCS2 ,3 = Use SCS3
,char BUS //BUS : 0 = Use BUS0, 1 = Use BUS1
)

void SPI_NAND_DMA
(
unsigned long dma_page_addr //dma pic addr in flash
,unsigned long X_coordinate   //pic write to sdram coordinate of x
,unsigned long Y_coordinate   //pic write to sdram coordinate of y
,unsigned long des_canvas_width //recommend= canvas_image_width
,unsigned int picture_Width
,unsigned int picture_Height
,unsigned long pic_buffer_Layer   //pic_buffer_Layer : read pic buffer in sdram
,unsigned long Show_pic_Layer   //Show_pic_Layer : write pic into sdram addr
,unsigned char chorma   //0: no transparent,1:Specify a color as transparent
,unsigned long Background_color   //transparent color
)

void SPI_NAND_DMA_png
(
unsigned long dma_page_addr //dma pic addr in flash
,unsigned long pic_buffer_Layer//pic_buffer_Layer : read pic buffer in sdram
,unsigned long Show_pic_Layer //Show_pic_Layer : write pic into sdram addr
,unsigned int picture_Width
,unsigned int picture_Height
)
```

範例 ：

SPI_NOR_initial_DMA (3,0,1,1,0);

+

(Flash = 128Mbit or under 128Mbit)

DMA_24bit(2,0,0,800,480,800,0);

Or

(Flash over 128Mbit)

DMA_32bit(2,0,0,800,480,800,0);

Condition :

Mode = 3:Dual_3Bh, BUS=0, Use BUS0, SCS = 1 : Use SCS1. flash =1,winbond flash,

addr_24b_32b =0,24bit addr.

Clk =2 :SPI Clock = System Clock /{(Clk)*2} , SPI CLK recommend<=90MHz

(X1,Y1) = (0,0) : DMA Coordinate = (0,0).

X_W : DMA Block width = 800 . Y_H : DMA Block height = 480.

P_W : DMA Picture width =800 . Addr :DMA Source Start address = 0.

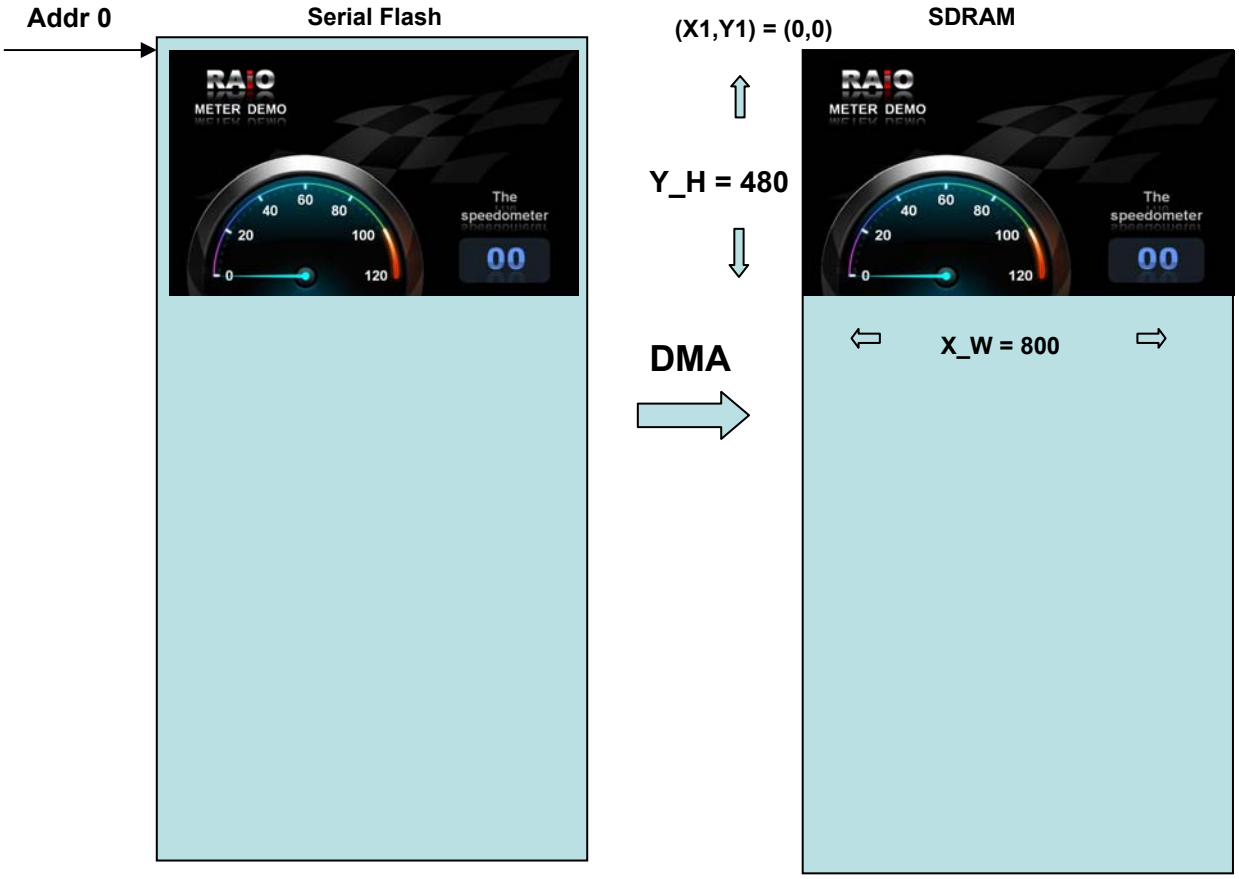


Figure 3.4: 使用 DMA 功能將圖資從 Flash 中讀出並寫入 SDRAM

3.2.1： 媒體解碼單元(Media Decode Unit)

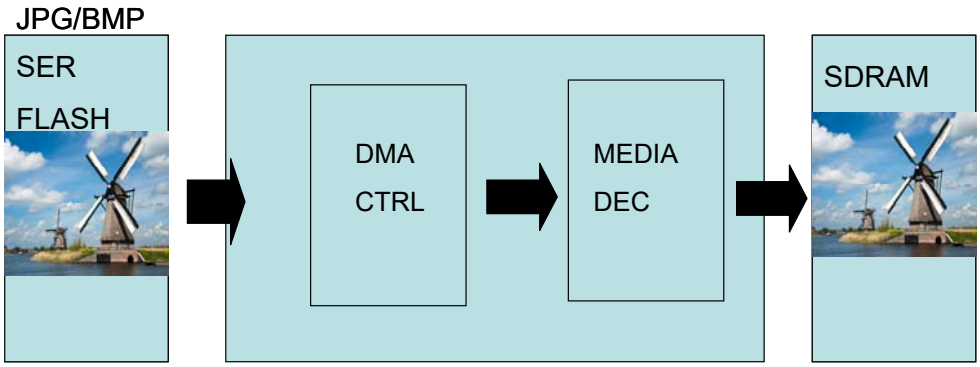
RA8889 支援媒體解碼單元，包含 JPEG（ISO/IEC 10918-1 Baseline profile，YUV444，YUV422，YUV420，YUV400，並不支援重啟間隔格式），BMP（raw data）和 AVI（motion jpeg）格式。 RA8889 可以自動區分上述三種格式，並將其自動解析為相對的解碼器。 在視頻功能中，RA8889 提供自動播放，暫停和停止功能。使用者應事先將圖像或視頻下載到串列快閃記憶體中，並通過設定 DMA、CANVAS 和 PIP 相對的暫存器讓他們顯示在 LCD 螢幕上。

圖像寫入的位址，請參考 CANVAS 相關暫存器。 由於視頻顯示在 PIP1 或 PIP2 窗口中，因此使用者應在播放視頻之前設定 PIP 相關暫存器。 此外，RA8889 還提供中斷和忙碌標誌進行檢查。

注意：

1.關於串列快閃存介面，請使用支援 Quad mode 串列快閃記憶體，建議核心時脈頻率高於 100MHz。
2. AVI frame rate 可以是 30、29.97、25、24、23.97、20 和 15。
3. PIP 的色深應與主要視窗的色深一致。
4. AVI / JPG 的寬度和高度必須是 8 的倍數。
5.串列快閃記憶體的 DMA 長度應等於圖像或視頻的檔案容量。

3.2.2:硬體圖像的解碼流程



Reference CANVAS REG for write SDRAM data

Figure 3-5

3.2.3 ： 圖像解碼流程圖

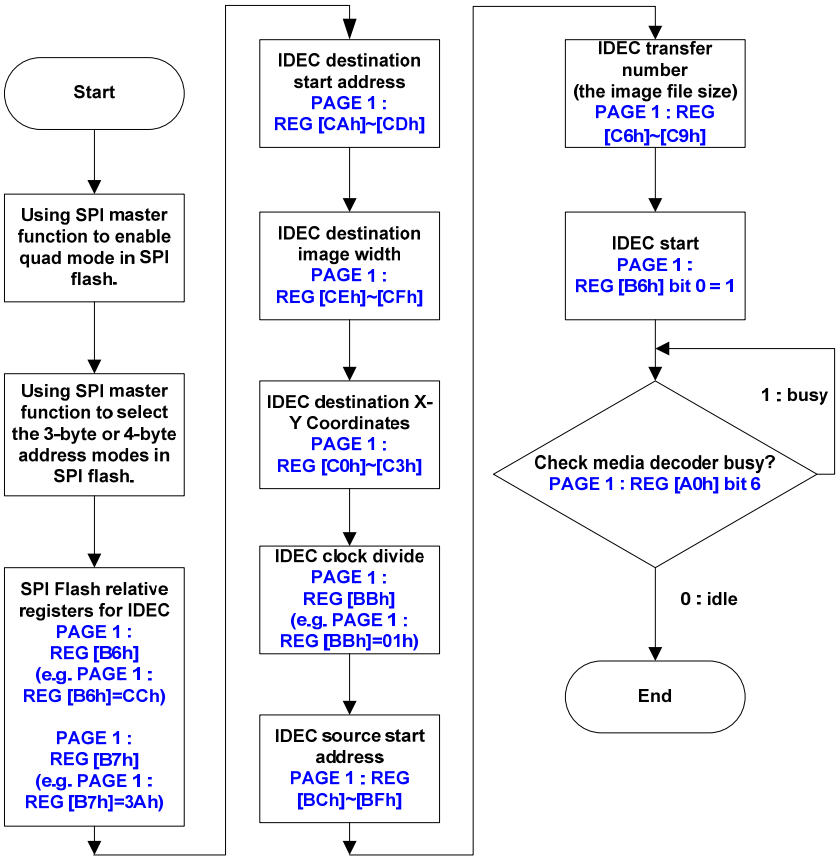
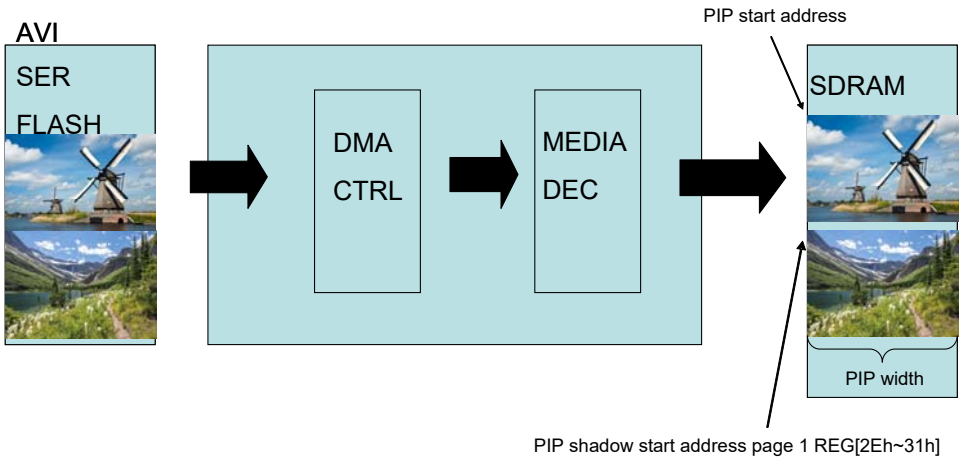


Figure 3-6

註 ：**IEDC** 意即支持媒體譯碼單元的影像格式**(JPEG,BMP** 及 **AVI)**

3.2.4 : AVI 的解碼流程



Reference PIP REG for write SDRAM data

Figure 3-7

3.2.5: AVI 解碼流程圖

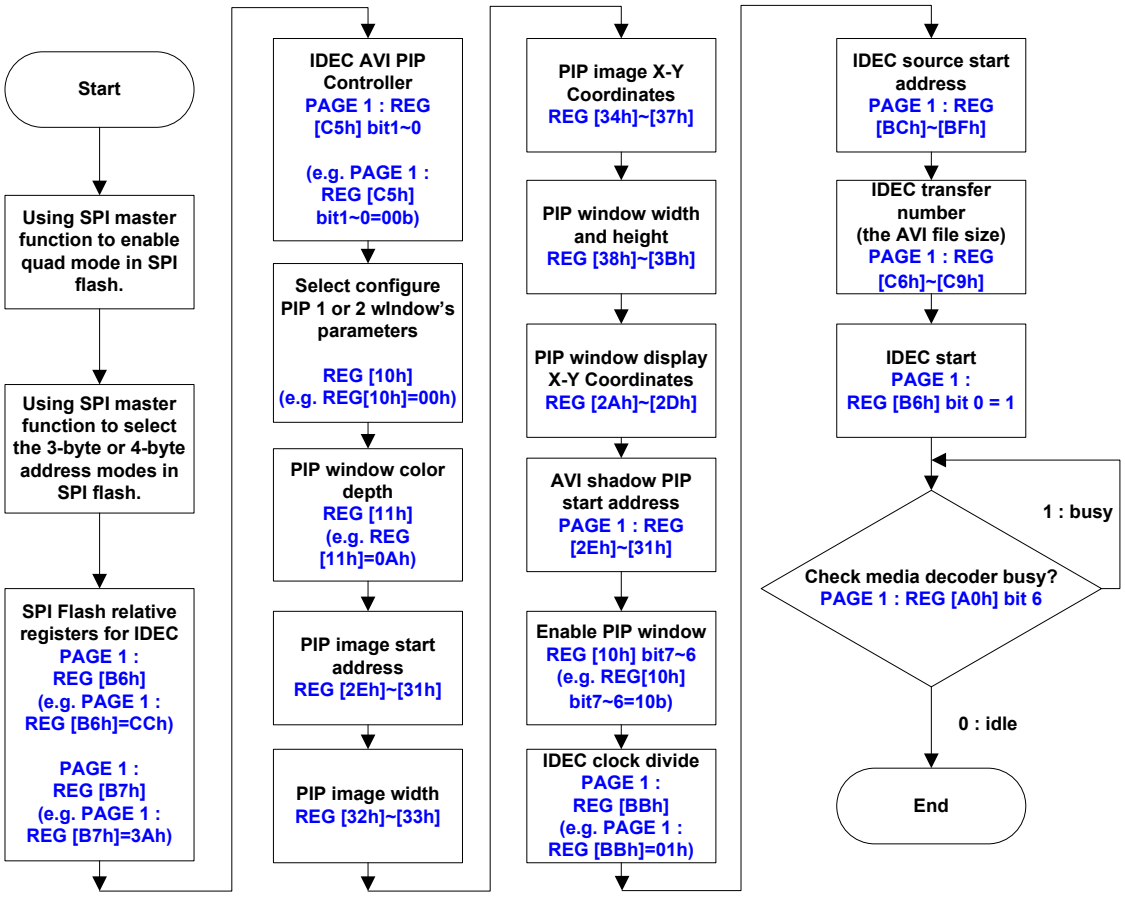


Figure 3-8

註 ：IEDC 意即支持媒體譯碼單元的影像格式(JPEG,BMP 及 AVI)

3.2.6 媒體譯碼(Media decode)功能 API 範例與 LCD 影響顯示結果:

以下提供 7 個相關的 API:

```
void AVI_window
(
unsigned char ON_OFF //0 : turn off AVI window, 1 :turn on AVI window
);
```

NOR FLASH :

```
void SPI_NOR_initial_JPG_AVI
(
char flash //0 : MXIC , 1 : Winbond
,unsigned char addr_24b_32b//flash addr : 0:24bit addr, 1:32bit addr
,char BUS //BUS : 0 = Use BUS0, 1 = Use BUS1
,char SCS //SCS : 0 = Use SCS0, 1 = Use SCS1 ,2 = Use SCS2 ,3 = Use SCS3
,char SCK_Divide //media decode divide : IDEC Clock = CORE CLK/2^SCK_Divide ,range:0~3,
recommend<= 60MHz
)

void JPG_NOR
(
unsigned long addr // JPG pic addr in flash
,unsigned long JPGsize //JPG pic size
,unsigned long IDEC_canvas_width //recommend= canvas_image_width
,unsigned short x //JPG pic write to sdram coordinate of x
,unsigned short y //JPG pic write to sdram coordinate of y
)
void AVI_NOR
(
unsigned long addr // AVI addr in flash
,unsigned long vediosize //AVI size
,unsigned long shadow_buffer_addr//shadow buffer addr
,unsigned PIP_buffer_addr //PIP buffer addr
,unsigned long x //show AVI to coordinate of x
,unsigned long y //show AVI to coordinate of y
,unsigned long height //vedio height
,unsigned long width   //vedio width
,unsigned long PIP_width // PIP Image width, recommend= canvas_image_width
)
```

NAND FLASH： 僅支援 W25N01GV

```
void SPI_NAND_initial_JPG_AVI
(
char BUS   //BUS : 0 = Use BUS0, 1 = Use BUS1
,char SCS //SCS : 0 = Use SCS0, 1 = Use SCS1 ,2 = Use SCS2 ,3 = Use SCS3
,unsigned long buffer //AVI shadow buffer addr,for load avi data to sdram buffer
,char SCK_Divide //media decode divide : IDEC Clock = CORE CLK/2^SCK_Divide ,range:0~3,
recommend<= 60MHz
)
void JPG_NAND
(
unsigned long addr//Pic addr in flash
,unsigned long JPGsize //Pic size
,unsigned long IDEC_canvas_width //recommend= canvas_image_width
,unsigned short x //write pic coordinate of x
,unsigned short y //write pic coordinate of y
)

void AVI_NAND
(
unsigned long addr //vedio addr in flash
,unsigned long vediosize //vedio size
,unsigned long shadow_buffer_addr //shadow buffer addr
,unsigned PIP_buffer_addr //PIP buffer addr
,unsigned long x //show vedio coordinate of x
,unsigned long y //show vedio coordinate of y
,unsigned long height //vedio height
,unsigned long width //vedio width
,unsigned long PIP_canvas_Width //recommend= canvas_image_width
)
```

JPG 範例 :

```
SPI_NOR_initial_JPG_AVI (1,0,1,2,1);
JPG_NOR (1152000,42237,canvas_image_width,0,0);
```

Condition :
flash =1,winbond flash, addr_24b_32b =0,24bit addr, BUS=1, Use BUS1, SCS = 2 : Use SCS2. ,
SCK_Divide : media decode divide : IDEC Clock = CORE CLK/2^SCK_Divide ,range:0~3,
recommend<= 60MHz
Pic Addr = 1152000 , JPGsize = 42237, IDEC_canvas_width = canvas_image_width ,
(X1,Y1) = (0,0) : DMA Coordinate = (0,0).

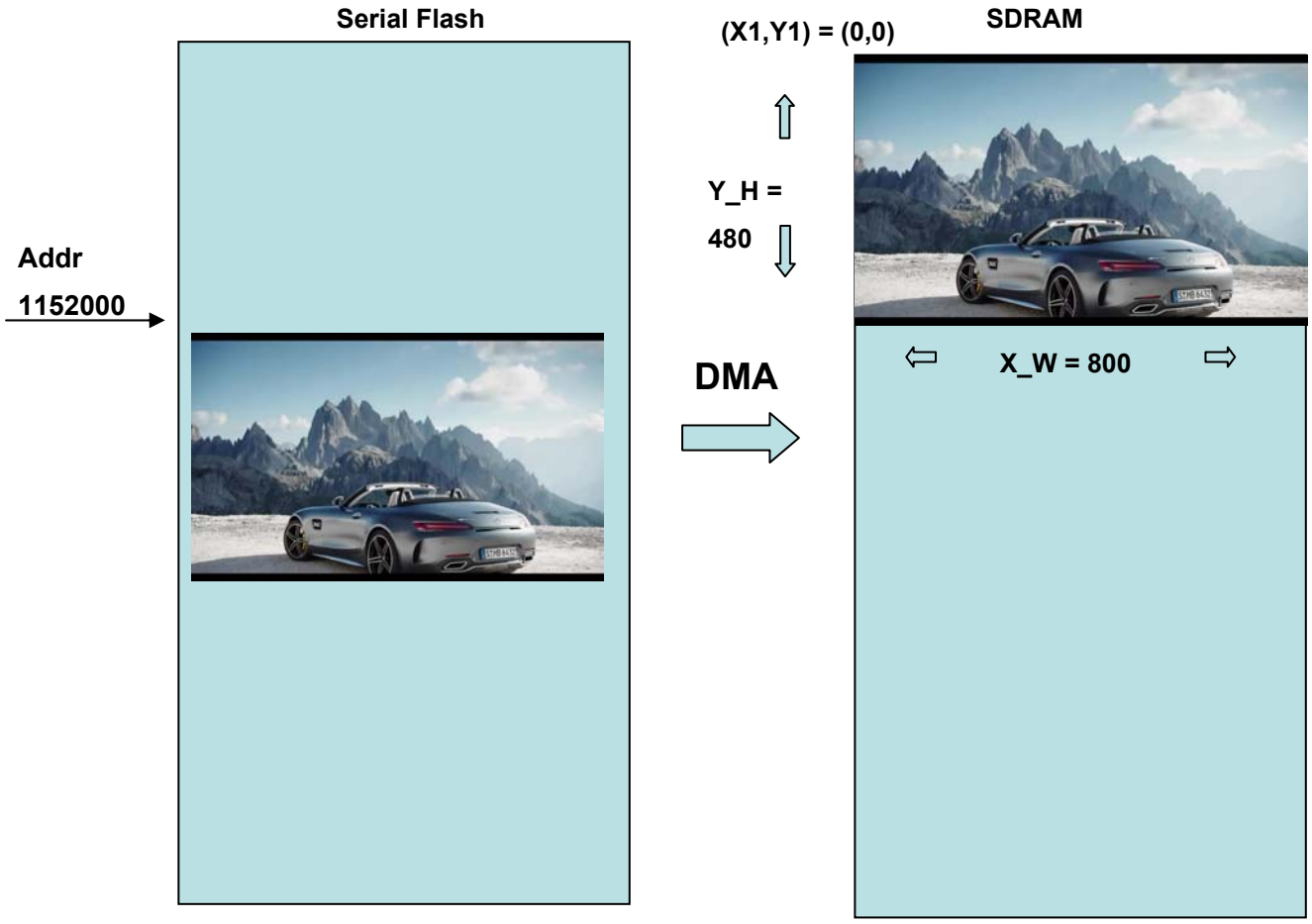


Figure 3.9: 使用 Media decode 功能將存在外接串列 Flash 的 JPG 圖資寫入到 SDRAM 中

AVI 範例 :

```
SPI_NOR_initial_JPG_AVI (1,0,1,2,1);
AVI_NOR(1296674,13327214,shadow_buff,shadow_buff+2400,0,0,322,548,canvas_image_width);
AVI_window(1);
```

Condition :
flash =1,winbond flash, addr_24b_32b =0,24bit addr, BUS=1, Use BUS1, SCS = 2 : Use SCS2. ,
SCK_Divide : media decode divide : IDEC Clock = CORE CLK/2^SCK_Divide ,range:0~3,
recommend<= 60MHz
vedio Addr = 1296674, videosize= 13327214, shadow_buff = shadow_buffer_addr,
PIP_buffer_addr= shadow_buffer_addr+2400 , (X1,Y1) = (0,0) : show vedio Coordinate = (0,0),
vedio height = 322, vedio width =548 , PIP_width = canvas_image_width
AVI_window = 1,turn ON AVI window

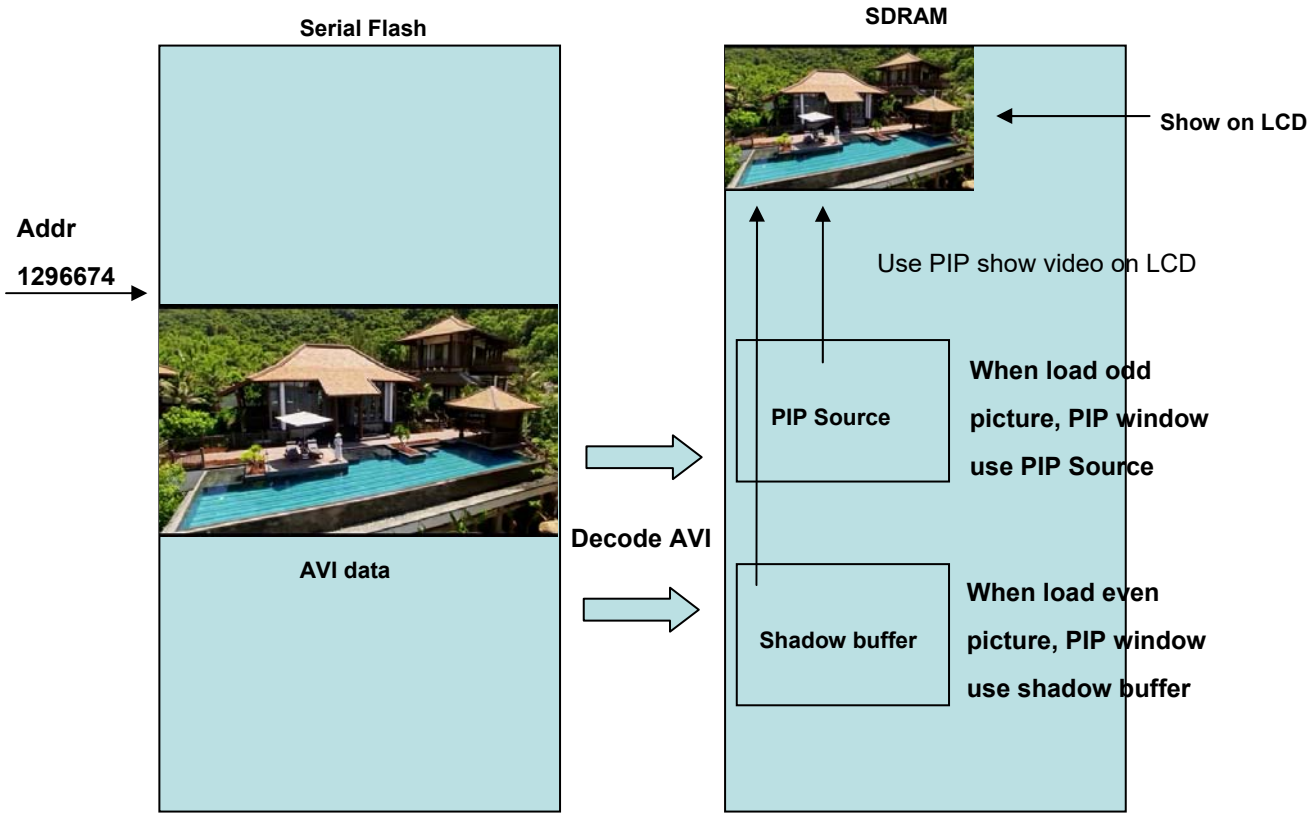


Figure 3.10: 使用 Media decode 功能將存在外接串列 Flash 的 AVI 檔，寫入到 SDRAM 中，並顯示在 LCD 上

第四章 ：Block Transfer Engine

概要:

RA8889 內建了增加區域運算效能的區域運算引擎(BTE)，當有一個區塊資料需要被搬移或是和某些特定資料做一些邏輯運算，RA8889 可以經由 BTE 硬體加速來簡化 MCU 的程式，這邊我們將要來討論 BTE 引擎的操作與功能。在使用 BTE 功能之前，使用者必須選擇相對應的 BTE 功能，RA8889 支援 13 種 BTE 功能。關於功能描述，請參照表 4-1。每個 BTE 功能針對不同的應用最多支援 16 種光柵運算(ROP)。它們可以提供給光柵運算來源與光柵運算目的地不同的邏輯結合。通過結合 BTE 功能與光柵運算，使用者可以達到許多非常有用的應用操作。請參考章節後面的詳細描述。這份應用說明集中在一些常用的 BTE 功能，如 **Solid Fill, Pattern Fill with ROP, Pattern Fill with Chroma key(w/o ROP),MCU Write with ROP, MCU Write with Chroma key(w/o ROP), Memory Copy with ROP, Memory Copy with chroma key(w/o ROP), MCU Write with Color Expansion, MCU Write with Color Expansion and chroma key, Memory copy with Alpha Blending.** "如果我們的客戶需要其他 BTE 功能的解說，請洽詢瑞佑科技業務部或代理商。

表 4-1：BTE 操作功能

BTE Operation REG[91h] Bits [3:0]	BTE Operation
0000b	MCU Write with ROP.
0010b	Memory copy with ROP.
0100b	MCU Write with Chroma key (w/o ROP)
0101b	Memory copy with Chroma key (w/o ROP)
0110b	Pattern Fill with ROP
0111b	Pattern Fill with Chroma key
1000b	MCU Write with Color Expansion
1001b	MCU Write with Color Expansion and Chroma key
1010b	Memory copy with Alpha blending
1011b	MCU Write with Alpha blending
1100b	Solid Fill
1110b	Memory copy with Color Expansion
1111b	Memory copy with Color Expansion and Chroma key
Other combinations	Reserved

表 4-2 : ROP Function

ROP Bits REG[91h] Bit[7:4]	Boolean Function Operation
0000b	0 (Blackness)
0001b	~S0・~S1 or ~ (S0+S1)
0010b	~S0・S1
0011b	~S0
0100b	S0・~S1
0101b	~S1
0110b	S0^S1
0111b	~S0+~S1 or ~ (S0・S1)
1000b	S0・S1
1001b	~ (S0^S1)
1010b	S1
1011b	~S0+S1
1100b	S0
1101b	S0+~S1
1110b	S0+S1
1111b	1 (Whiteness)

Note:

1. ROP 功能 S0: 來源 0 的資料, S1: 來源 1 的資料, D: 目的地的資料
2.以 pattern fill 功能來說,目的地的資料是由來源來表示。

Example:

當 ROP 功能設定為 Ch (1100b), 目的地的資料 = 來源 0 的資料
當 ROP 功能設定為 Eh (1110b), 目的地的資料 = 來源 0 + 來源 1
當 ROP 功能設定為 2h (0010b), 目的地的資料 = ~來源 0・來源 1
當 ROP 功能設定為 Ah (1010b), 目的地的資料 = 來源 1 資料

BTE Access Memory Method

伴隨著這些設定，BTE 的來源/目的地的資料被當作一個顯示的區塊，下面的例子解釋了將來源 0/來源 1/目的地定義成區塊的方法。

BTE Chroma Key (Transparency Color) Compare

當啟用 BTE 通透色時，BTE 程序會比對來源 0 的資料和背景色暫存器的資料。資料相等的部分在目的地並不會有所改變，資料不相等的部分才會由來源 0 寫入到目的地。

當來源色彩深度 = 256 色時，

來源 0 紅色部分只比對 REG[D5h]Bit[7:5]

來源 0 綠色部分只比對 REG [D6h] Bit [7:5],

來源 0 藍色部分只比對 REG [D7h] Bit [7:6]

當來源色彩深度 = 65k 色時，

來源 0 紅色部分只比對 REG[D5h]Bit[7:3]

來源 0 綠色部分只比對 REG [D6h] Bit [7:2]

來源 0 藍色部分只比對 REG [D7h] Bit [7:3]

當來源色彩深度 = 16.7M 色時，

來源 0 紅色部分只比對 REG[D5h]Bit[7:0]

來源 0 綠色部分只比對 REG [D6h] Bit [7:0]

來源 0 藍色部分只比對 REG [D7h] Bit [7:0]

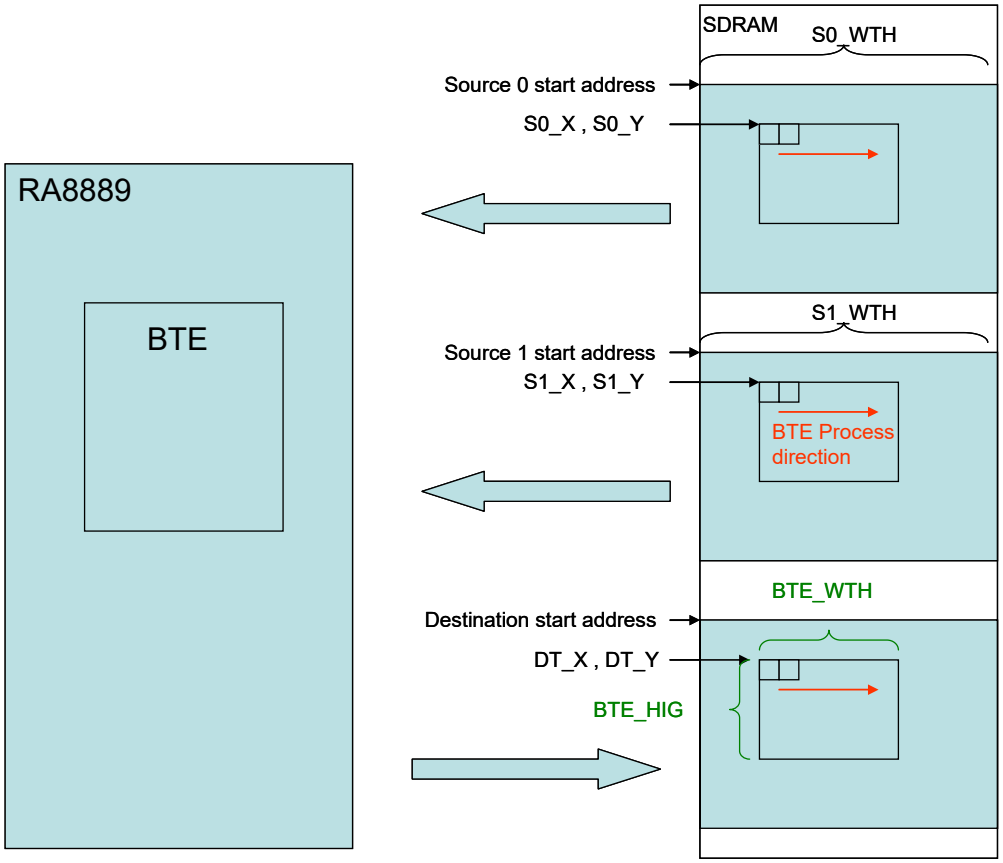


圖 4-1 : Memory Access of BTE Function

4.1.1: Solid Fill

Solid Fill BTE 功能提供使用者可將特定的區域 (BTE 來源區域) 以特定顏色來填滿。這個功能用在將 SDRAM 裡的一個大區塊填滿為同樣的一個顏色，它的顏色是經由 BTE 前景色來設定。

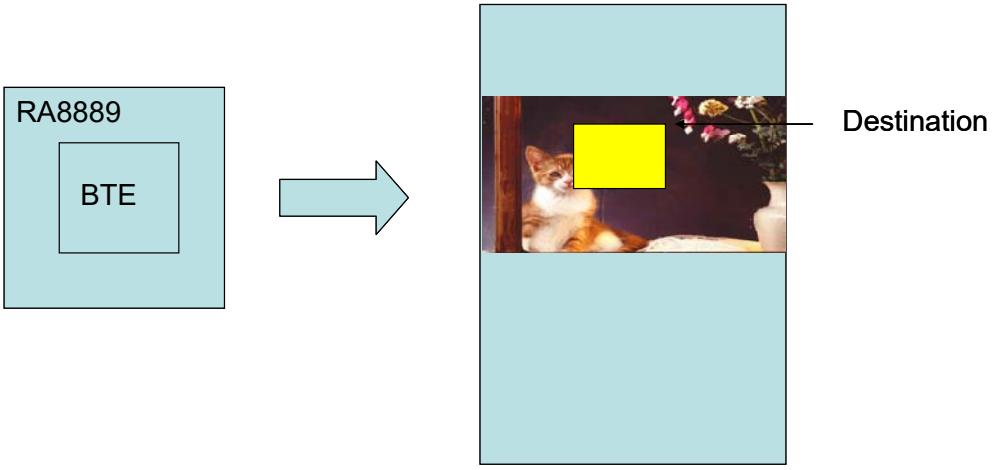


圖 4-2 : Hardware Data Flow

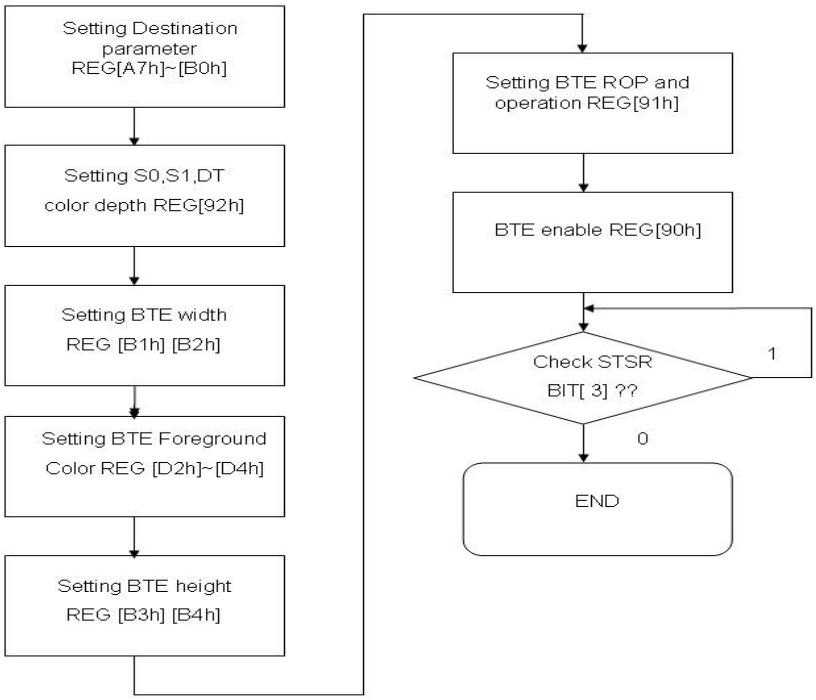


圖 4-3 : 流程圖

4.1.2. BTE 的 Solid Fill 功能在 LCD 上的顯示結果:

以下為 API 程式與範例說明，圖 4-4 為使用 Solid Fill 功能填滿一個紅色的方形區塊。

```
void BTE_Solid_Fill
(
unsigned long Des_Addr //start address of destination
,unsigned short Des_W // image width of destination (recommend = canvas image width)
, unsigned short XDes //coordinate X of destination
,unsigned short YDes //coordinate Y of destination
,unsigned long Foreground_color //Solid Fill color
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
)
```

範例:

```
BTE_Solid_Fill(0,canvas_image_width,0,0,0xe0,200,200); //When color depth = 8bpp
or
BTE_Solid_Fill(0,canvas_image_width,0,0,0xf800,200,200); //When color depth = 16bpp
or
BTE_Solid_Fill(0,canvas_image_width,0,0,0xFF0000,200,200);//When color depth = 24bpp
```

條件:

```
start address of destination = 0, ImageWidth = canvas_image_width, coordinate of destination = (0,0)
Foreground Color: 0xe0 (8bpp) (R3G3B2)、0xf800(16bpp)(R5G6B5)、0xFF0000(24bpp)(R8G8B8)
BTE Window Size = 200x200
```

程式執行結果:

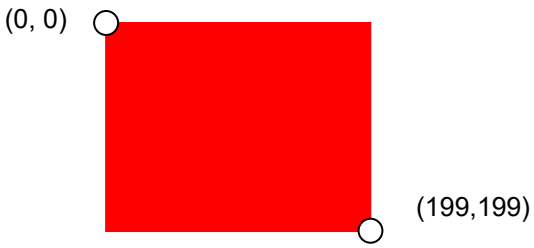


圖 4-4：使用 Solid Fill 功能將[(0, 0) 到(199,199)]用紅色填滿

4.2.1:Pattern Fill with ROP

“Pattern Fill with ROP” 功能可設定一個在 SDRAM 中的特定方形記憶區塊，並填入重覆的特定圖形樣板。圖形樣板是 8x8/16x16 的像素圖形，存放在 SDRAM 中的非顯示區的特定位置，圖形樣板並且可以配合 16 種光柵運算和目的地資料做邏輯運算。此操作可以用來加速某些需要在特定區域重覆貼入某一種圖形，像是背景圖案等應用。

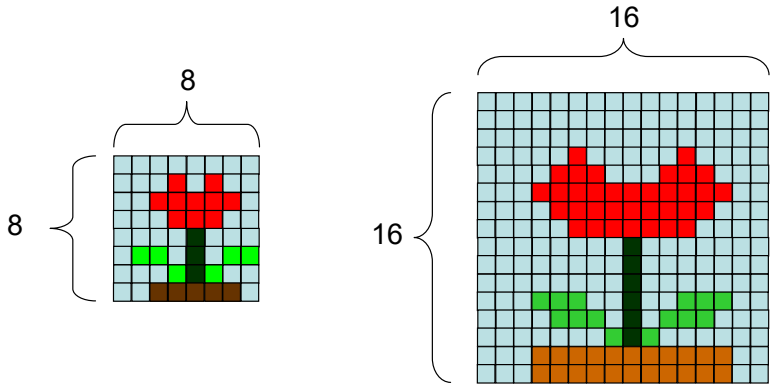


圖 4-5 : Pattern Format

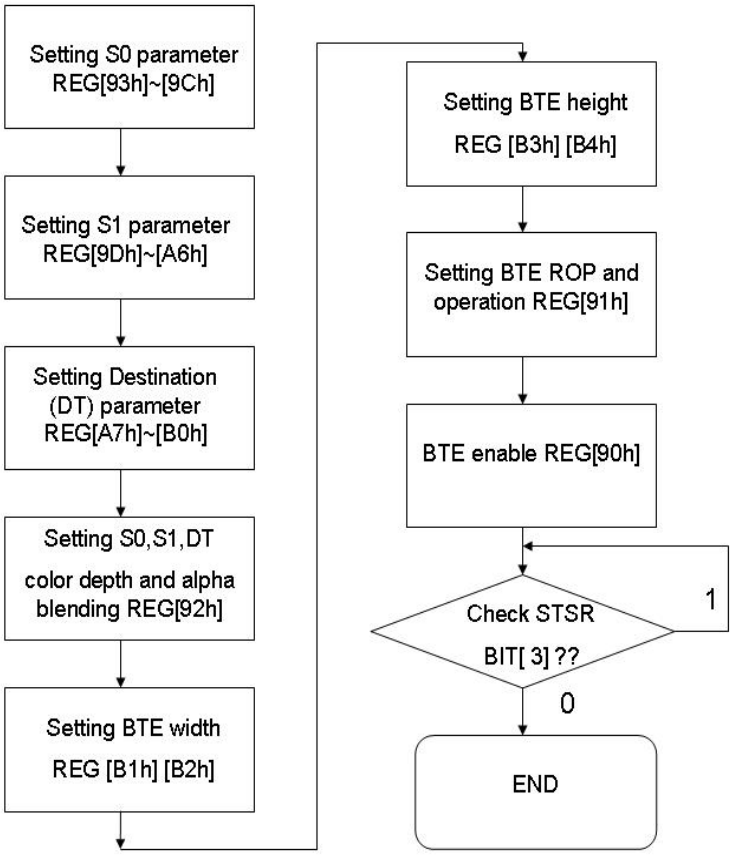


圖 4-6 : 流程圖

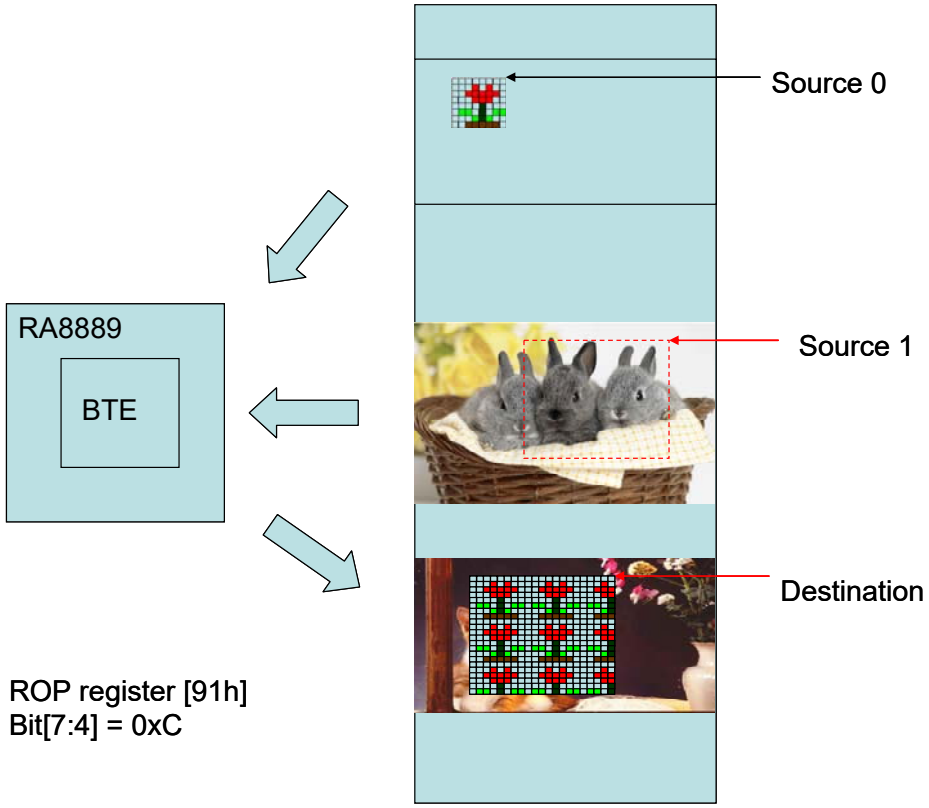

Source 0
Source 1
Destination
RA8889
BTE
ROP register [91h]
Bit[7:4] = 0xC

圖 4-7：硬體資料流程

4.2.2. BTE 的 Pattern Fill with ROP 功能在 LCD 上的執行顯示結果:

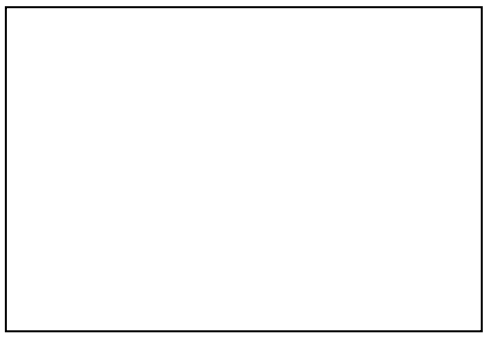


圖 4-8: 在這個範例中，SDRAM 目前的資料

圖 4-9 (16x16) 圖形樣版

API for pattern fill with ROP:

```
void BTE_Pattern_Fill
(
unsigned char P_8x8_or_16x16 //0 : use 8x8 Icon , 1 : use 16x16 Icon.
,unsigned long S0_Addr //Start address of Source 0
,unsigned short S0_W //image width of Source 0 (recommend = canvas image width)
,unsigned short XS0 // coordinate X of Source 0
,unsigned short YS0 // coordinate Y of Source 0
,unsigned long S1_Addr //Start address of Source 1
,unsigned short S1_W //image width of Source 1 (recommend = canvas image width)
,unsigned short XS1 //coordinate X of Source 1
,unsigned short YS1 //coordinate Y of Source 1
,unsigned long Des_Addr // start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
, unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned int ROP_Code
/*ROP_Code :
    0000b       0(Blackness)
    0001b       ~S0!E~S1 or ~(S0+S1)
    0010b       ~S0!ES1
    0011b       ~S0
    0100b       S0!E~S1
    0101b       ~S1
    0110b       S0^S1
    0111b       ~S0 + ~S1 or ~(S0 + S1)
    1000b       S0!ES1
    1001b       ~(S0^S1)
```

```
    1010b       S1
    1011b       ~S0+S1
    1100b       S0
    1101b       S0+~S1
    1110b       S0+S1
    1111b       1(whiteness)*/
,unsigned short X_W //Width of BTE Winodw
,unsigned short Y_H //Length of BTE Winodw
)
```

範例:

```
SPI_NOR_initial_DMA (3,0,1,1,0);
+
//MCU_8bit_ColorDepth_8bpp
DMA_24bit(2,0,0,800,480,800,15443088);
MPU8_8bpp_Memory_Write(0,0,16,16,Icon_8bit_8bpp);
or
//MCU_8bit_ColorDepth_16bpp
MA_24bit(2,0,0,800,480,800,14675088);
MPU8_16bpp_Memory_Write(0,0,16,16,Icon_8bit_16bpp);
or
//MCU_8bit_ColorDepth_24bpp
DMA_24bit(2,0,0,800,480,800,0);
MPU8_24bpp_Memory_Write(0,0,16,16,Icon_8bit_24bpp);
or
//MCU_16bit_ColorDepth_16bpp
MA_24bit(2,0,0,800,480,800,14675088);
MPU16_16bpp_Memory_Write(0,0,16,16,Icon_16bit_16bpp);
or
//MCU_16bit_ColorDepth_24bpp_Mode_1
DMA_24bit(2,0,0,800,480,800,0);
MPU16_24bpp_Mode1_Memory_Write(0,0,16,16,Icon_16bit_24bpp_mode1);
or
//MCU_16bit_ColorDepth_24bpp_Mode_2
DMA_24bit(2,0,0,800,480,800,0);
MPU16_24bpp_Mode2_Memory_Write(0,0,16,16,Icon_16bit_24bpp_mode2);
+
BTE_Pattern_Fill(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,100,100);
```

條件:

P_8x8_or_16x16 = 1 , Pattern size = 16x16
Source 0 : Start Address = 0, Image Width = canvas_image_width, coordinate (0,0)
Source 1 : Start Address = 0, Image Width = canvas_image_width, coordinate (0,0)
Destination : Start Address = 0, Image Width = canvas_image_width, coordinate (500,200)
ROP_Code = 12 : Destination data = Source 0 data. BTE Window Size = 100x100

Step 1 :執行 DMA 功能從外接 FLASH 讀一張圖寫入到 SDRAM

(0, 0)

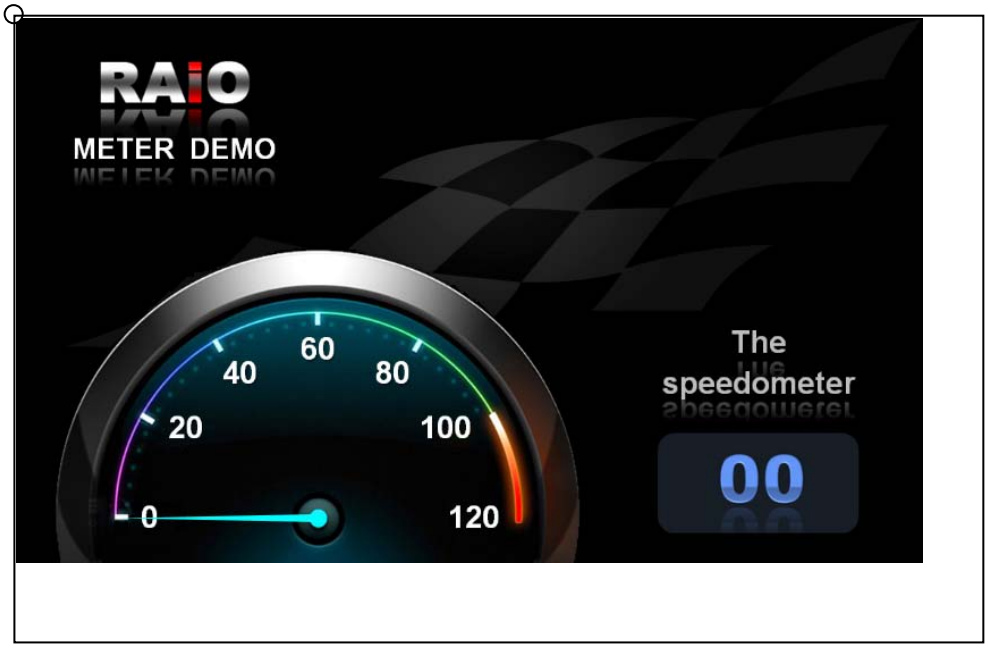


Figure 4-10: 使用 DMA 功能從外接 FLASH 讀一張圖寫入到 SDRAM

Step 2 : 寫入一個 16x16 icon 到記憶體(SDRAM)

(0, 0)

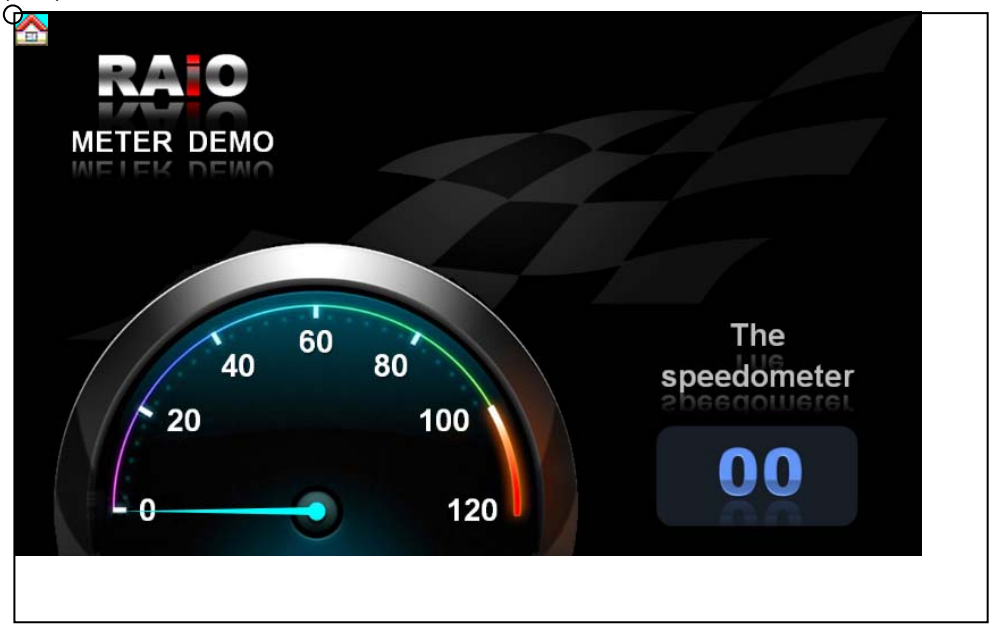


Figure 4-11: 寫入一個 16x16 icon 到記憶體(SDRAM)

Step 3：執行 Pattern fill 功能

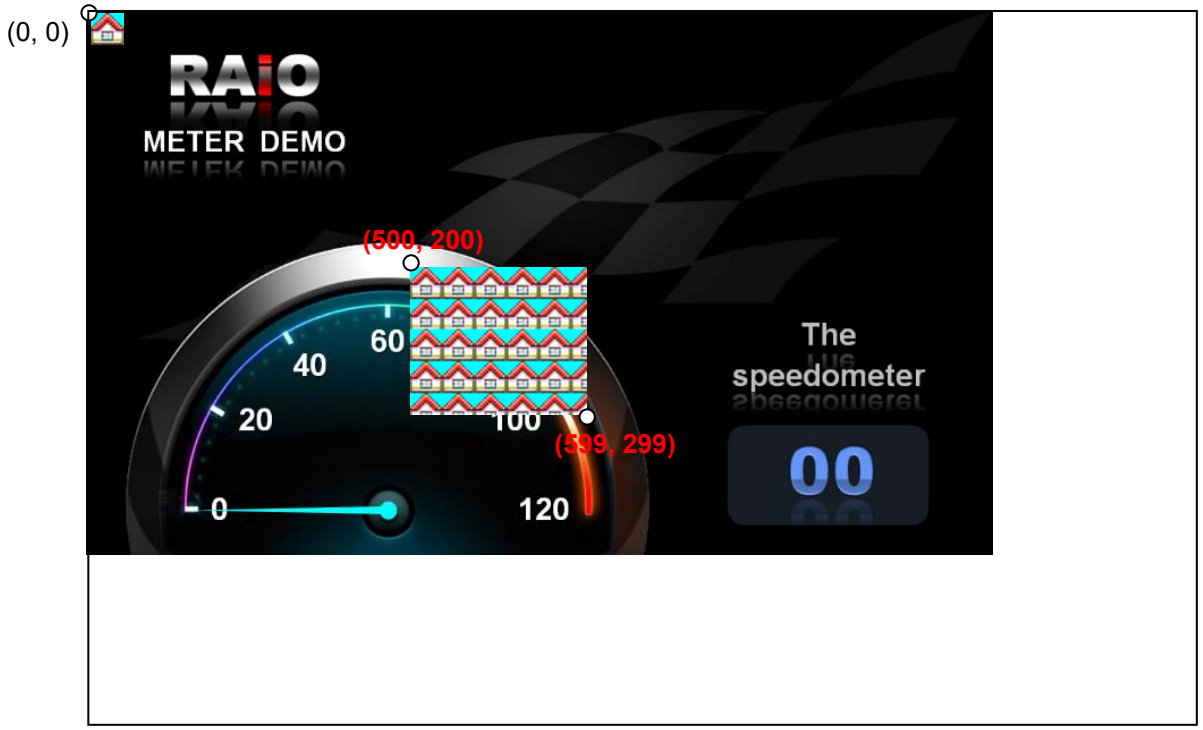


Figure 4-12: pattern fill 功能完成後的樣子，來源範圍從(0, 0)到(15, 15)，且目的地範圍從(500,200)到(599,299)

4.2.3:Pattern Fill with Chroma Key(w/o ROP)

“Pattern Fill with Chroma Key” 功能可設定在一個 DDRAM 中的特定方形記憶體，並填入重覆的特定圖形樣板，此功能與「Pattern Fill with ROP」功能有相同的功能，不同的是，加入通透性的功能。也就是對於特定的「通透色」，此 BTE 功能會予以忽略。通透色是在 REG[D5h]~[D7h]中被設定，當圖形樣板中的顏色與通透色一致時，那部份的“*目的地資料*”將不會改變。

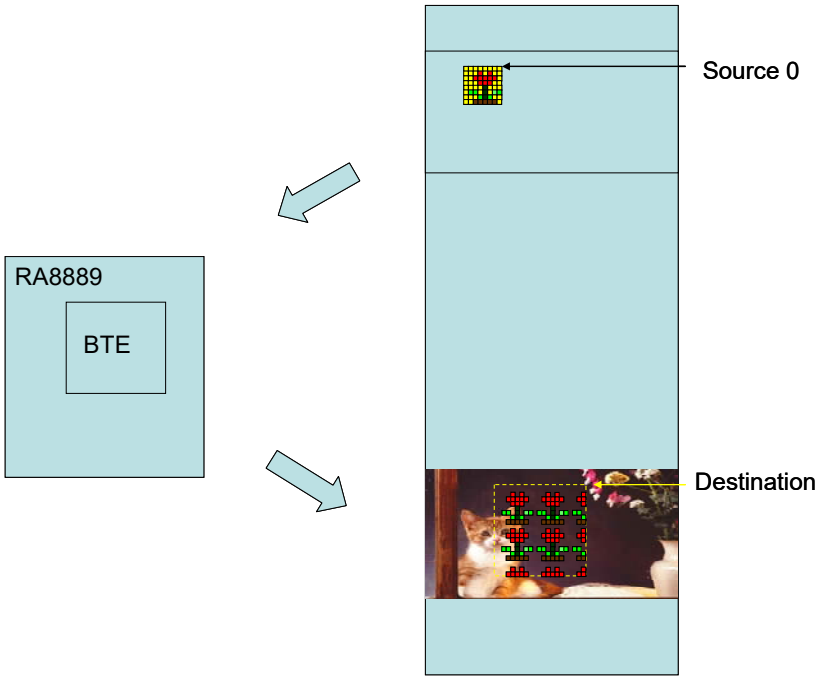


圖 4-10：硬體資料流程

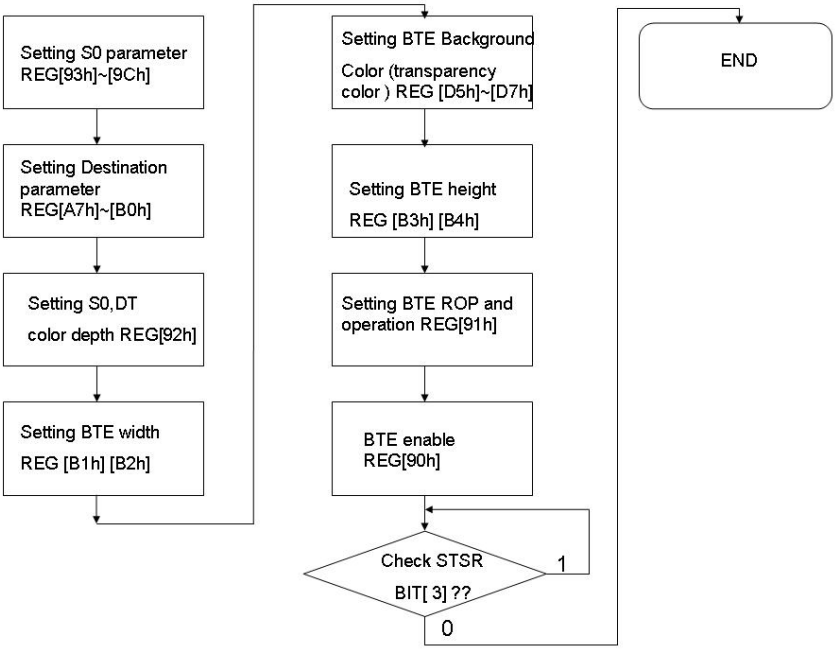


圖 4-11：流程圖

4.2.4: BTE Pattern Fill with Chroma key(w/o ROP)功能在 LCD 上的顯示結果:

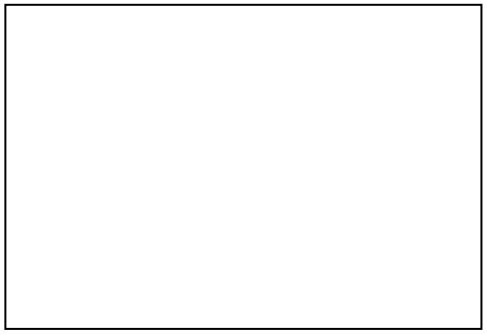


圖 4-13 (16x16) 圖形樣版

圖 4-12: 在這個範例中，SDRAM 目前的資料

API:

```
void BTE_Pattern_Fill_With_Chroma_key
(
unsigned char P_8x8_or_16x16 //0 : use 8x8 Icon , 1 : use 16x16 Icon.
,unsigned long S0_Addr //Start address of Source 0
,unsigned short S0_W //image width of Source 0 (recommend = canvas image width)
,unsigned short XS0 //coordinate X of Source 0
,unsigned short YS0 //coordinate Y of Source 0
,unsigned long S1_Addr //Start address of Source 1
,unsigned short S1_W //image width of Source 1 (recommend = canvas image width)
,unsigned short XS1 //coordinate X of Source 1
,unsigned short YS1 //coordinate Y of Source 1
,unsigned long Des_Addr //Des_Addr : start address of Destination
,unsigned short Des_W //Des_W : image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned int ROP_Code
/*ROP_Code :
    0000b       0(Blackness)
    0001b       ~S0!E~S1 or ~(S0+S1)
    0010b       ~S0!ES1
    0011b       ~S0
    0100b       S0!E~S1
    0101b       ~S1
    0110b       S0^S1
    0111b       ~S0 + ~S1 or ~(S0 + S1)
    1000b       S0!ES1
    1001b       ~(S0^S1)
```

```
    1010b       S1
    1011b       ~S0+S1
    1100b       S0
    1101b       S0+~S1
    1110b       S0+S1
    1111b       1(whiteness)*/
,unsigned long Background_color //Transparent color
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
)
```

範例:

```
SPI_NOR_initial_DMA (3,0,1,1,0);
+
//MCU_8bit_ColorDepth_8bpp
DMA_24bit(2,0,0,800,480,800,15443088);
MPU8_8bpp_Memory_Write(0,0,16,16,Icon_8bit_8bpp);
BTE_Pattern_Fill_With_Chroma_key(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,0x1f,100,100);
or
//MCU_8bit_ColorDepth_16bpp
DMA_24bit(2,0,0,800,480,800,14675088);
MPU8_16bpp_Memory_Write(0,0,16,16,Icon_8bit_16bpp);/
BTE_Pattern_Fill_With_Chroma_key(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,0x07ff,100,100);
or
//MCU_8bit_ColorDepth_24bpp
DMA_24bit(2,0,0,800,480,800,0);
MPU8_24bpp_Memory_Write(0,0,16,16,Icon_8bit_24bpp);
BTE_Pattern_Fill_With_Chroma_key(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,0x00ffff,100,100);
or
//MCU_16bit_ColorDepth_16bpp
DMA_24bit(2,0,0,800,480,800,14675088);
MPU16_16bpp_Memory_Write(0,0,16,16,Icon_16bit_16bpp);
BTE_Pattern_Fill_With_Chroma_key(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,0x07ff,100,100);
or
//MCU_16bit_ColorDepth_24bpp_Mode_1
```

```
DMA_24bit(2,0,0,800,480,800,0);
MPU16_24bpp_Mode1_Memory_Write(0,0,16,16,Icon_16bit_24bpp_mode1);
BTE_Pattern_Fill_With_Chroma_key(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,0x00ffff,100,100);
or
//MCU_16bit_ColorDepth_24bpp_Mode_2
DMA_24bit(2,0,0,800,480,800,0);
MPU16_24bpp_Mode2_Memory_Write(0,0,16,16,Icon_16bit_24bpp_mode2);
BTE_Pattern_Fill_With_Chroma_key(1,0, canvas_image_width,0,0,0, canvas_image_width,0,0,0,
canvas_image_width,500,200,12,0x00ffff,100,100);
```

條件:

P_8x8_or_16x16 = 1 , Pattern size = 16x16
Source 0 : Start Address = 0, Image Width = canvas_image_width,Coordinate = (0,0)
Source 1 : Start Address = 0, Image Width = canvas_image_width, Coordinate = (0,0)
Destination : Start Address = 0, Image Width = canvas_image_width, Coordinate = (500,200)
ROP_Code = 12 : Destination data = Source 0 data. BTE Window Size = 200x200
Background_color = Transparency color = 0x1f(8bpp) , 0x07ff(16bpp) ,0x00ffff(24bpp)(blue-green)

Step 1： 執行 **DMA** 功能從外接 **FLASH** 讀一張圖寫入到 **SDRAM**

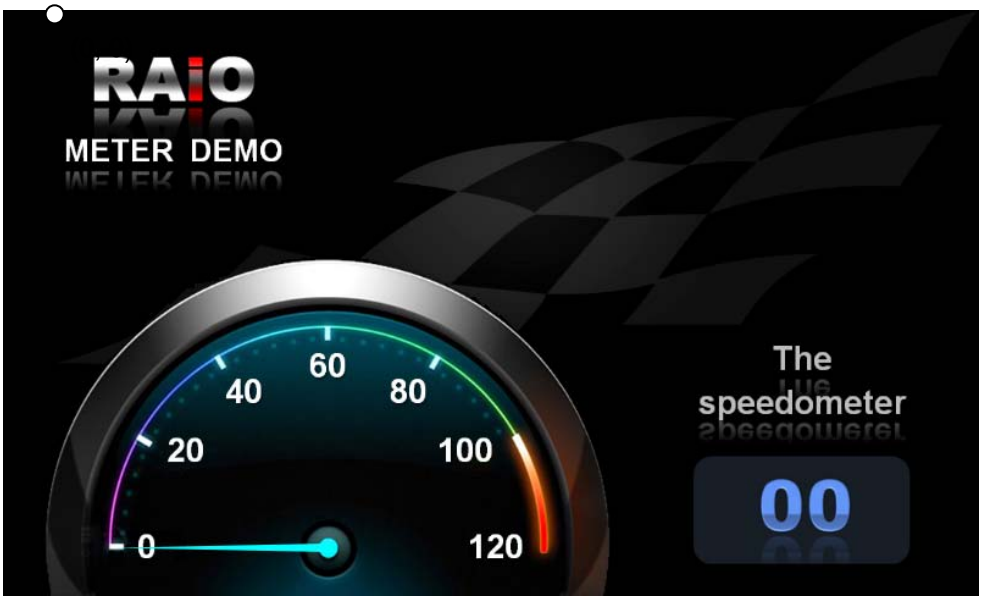


Figure 4-14: 執行 **DMA** 功能從外接 **FLASH** 讀一張圖寫入到 **SDRAM**

Step 2 : 寫入一個 16x16 icon 到記憶體(SDRAM)

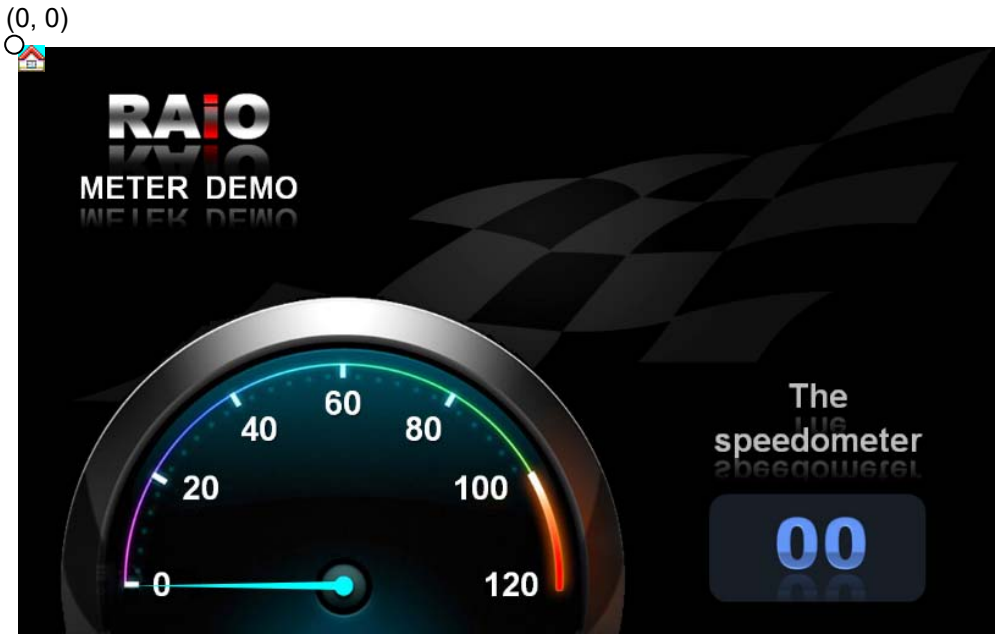


Figure 4-15: 寫入一個 16x16 icon 到記憶體(SDRAM)

Step 3 :執行 Pattern fill with chroma key function

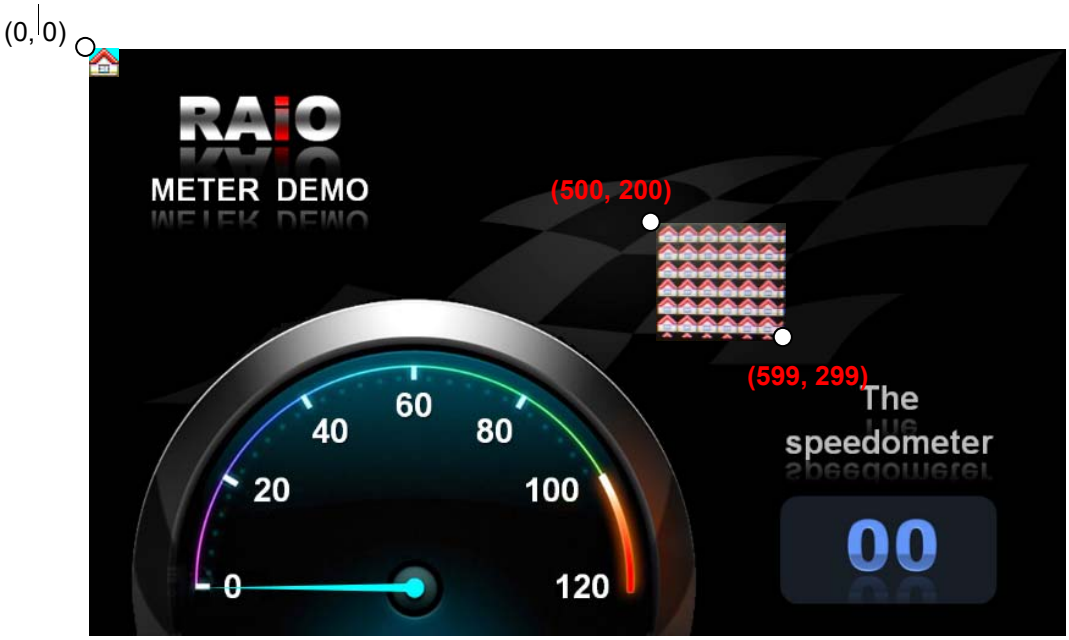


Figure 4-16: pattern fill with chroma key 功能完成後的樣子，來源範圍從(0, 0)到(15, 15)，且目的地範圍從(500,200)到 (599,299)

4.3.1: MCU Write with ROP

MCU Write with ROP 的 BTE功能，增加了MCU介面寫入SDRAM透過。的資料傳送速度MCU，Write BTE搭配光柵運算將資料填入特定的SDRAM，區域Write BTE功能支援全部16。種光柵運算Write BTE功能需要MCU。端提供資料

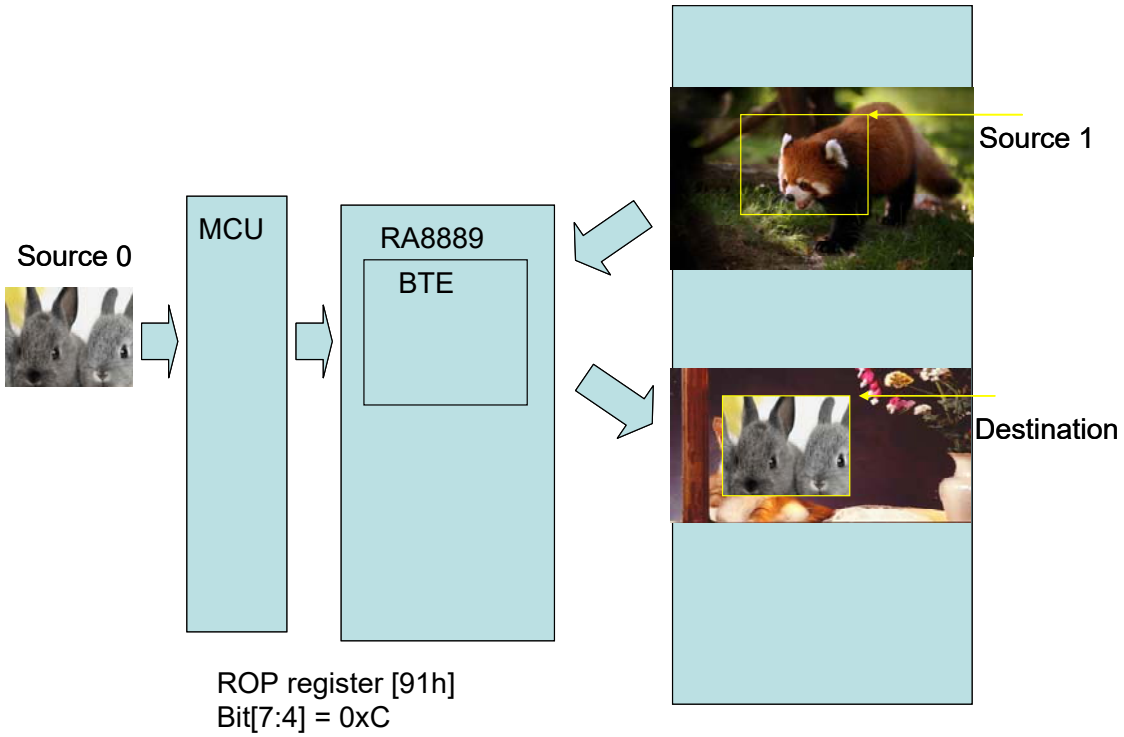


圖 4-17：硬體資料流程

以下是建議的程式流程以及暫存器設定：

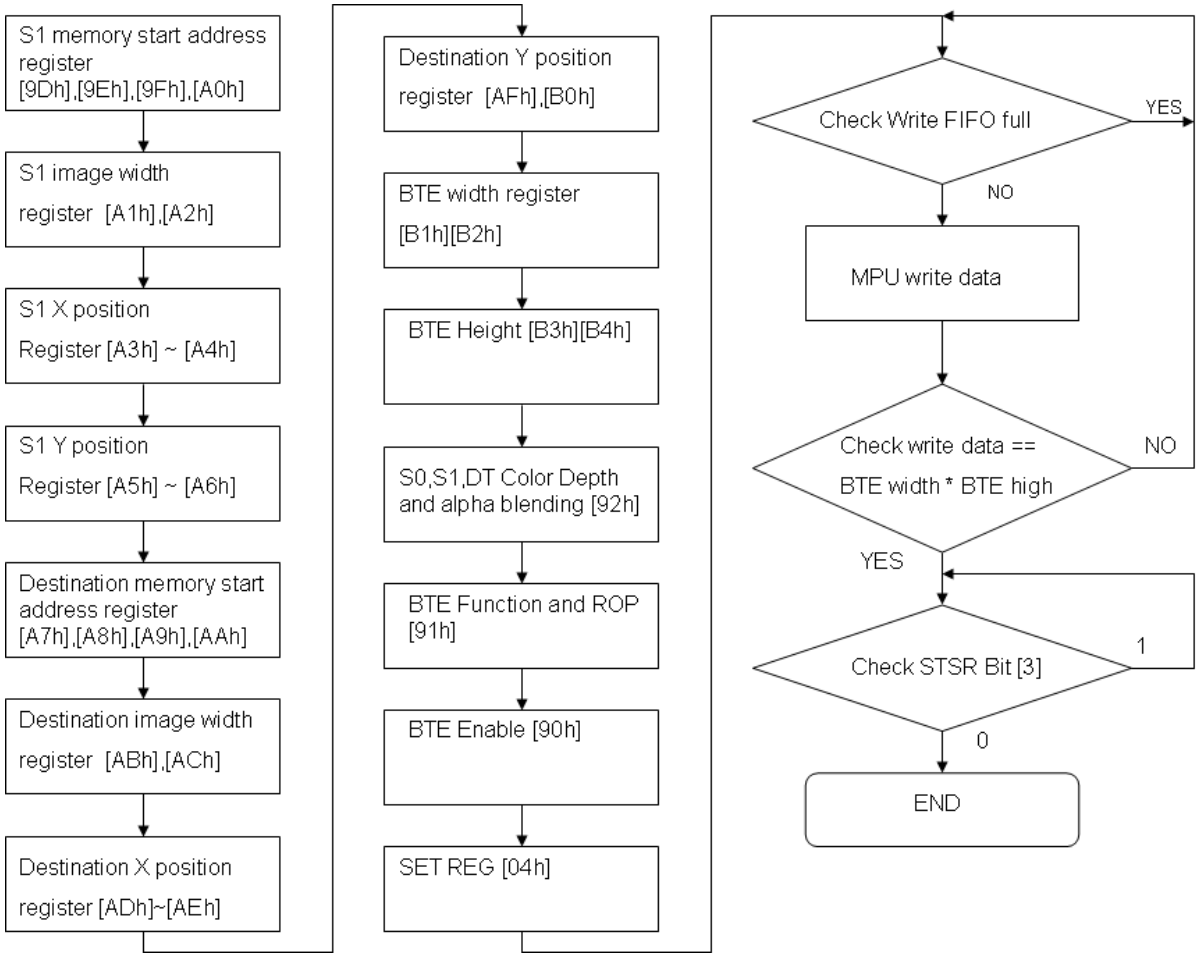


圖 4-18：流程圖

4.3.2: BTE MCU Write with ROP 功能在 LCD 上的顯示結果

在 BTE MCU Write with ROP 中，我們針對 MCU 8bit 和 16bit 各提供一組 API 供使用者使用，圖 4-19 為 SDRAM 的資料，圖 4-20 為 MCU 端的圖資，使用 BTE MCU Write with ROP 功能，搭配上 ROP 參數，可以將 MCU 端輸入的圖資做邏輯運算後寫入 SDRAM。以下為 API 程式與範例解說:

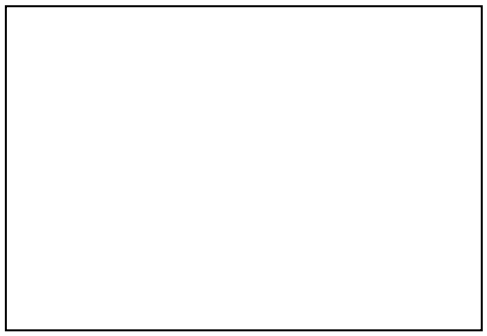


圖 4-19: 在這個範例中，SDRAM 目前的資料

圖 4-20：MCU 寫入的資料(128x128)

```
void BTE_MCU_Write_MCU_8bit
(
unsigned long S1_Addr //Start address of Source 1
,unsigned short S1_W //image width of Source 1 (recommend = canvas image width)
,unsigned short XS1 //coordinate X of Source 1
,unsigned short YS1 //coordinate Y of Source 1
,unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned int ROP_Code
/*ROP_Code :
    0000b       0(Blackness)
    0001b       ~S0!E~S1 or ~(S0+S1)
    0010b       ~S0!ES1
    0011b       ~S0
    0100b       S0!E~S1
    0101b       ~S1
    0110b       S0^S1
    0111b       ~S0 + ~S1 or ~(S0 + S1)
    1000b       S0!ES1
    1001b       ~(S0^S1)
    1010b       S1
    1011b       ~S0+S1
    1100b       S0
```

```
    1101b       S0+~S1
    1110b       S0+S1
    1111b       1(whiteness)*/
,unsigned short X_W // Width of BTE Window
,unsigned short Y_H // Length of BTE Window
,const unsigned char *data // 8-bit data
)


void BTE_MCU_Write_MCU_16bit
(
unsigned long S1_Addr //Start address of Source 1
,unsigned short S1_W //image width of Source 1 (recommend = canvas image width)
,unsigned short XS1 //coordinate X of Source 1
,unsigned short YS1 //coordinate Y of Source 1
,unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned int ROP_Code
/*ROP_Code :
    0000b       0(Blackness)
    0001b       ~S0!E~S1 or ~(S0+S1)
    0010b       ~S0!ES1
    0011b       ~S0
    0100b       S0!E~S1
    0101b       ~S1
    0110b       S0^S1
    0111b       ~S0 + ~S1 or ~(S0 + S1)
    1000b       S0!ES1
    1001b       ~(S0^S1)
    1010b       S1
    1011b       ~S0+S1
    1100b       S0
    1101b       S0+~S1
    1110b       S0+S1
    1111b       1(whiteness)*/
,unsigned short X_W // Width of BTE Window
```

```
,unsigned short Y_H // Length of BTE Window
,const unsigned short *data // 16-bit data
)
```

範例:

```
//Use 8bit MCU, 8bpp color depth
BTE_MCU_Write_MCU_8bit(0,canvas_image_width,0,0,0,canvas_image_width ,100,100,12,128,128,
gImage_8);
or
/Use 8bit MCU, 16bpp color depth
BTE_MCU_Write_MCU_8bit(0,canvas_image_width,0,0,0,canvas_image_width,100,100,12,128,128,
gImage_16);
or
//Use 8bit MCU, 24bpp color depth
BTE_MCU_Write_MCU_8bit(0,canvas_image_width,0,0,0,canvas_image_width,100,100,12,128,128,
gImage_24);
or
//Use 16bit MCU, 16bpp color depth
BTE_MCU_Write_MCU_16bit(0,canvas_image_width,0,0,0,canvas_image_width,100,100,12,128,128
,pic1616);
or
//Use 16bit MCU, 24bpp color depth and data format use mode 1
BTE_MCU_Write_MCU_16bit(0,canvas_image_width,0,0,0,canvas_image_width,100,100,12,128,128
,pic16241);
or
//Use 16bit MCU, 24bpp color depth and data format use mode 2
BTE_MCU_Write_MCU_16bit(0,canvas_image_width,0,0,0,canvas_image_width,100,100,12,128,128
,pic1624);
```

條件:

Source 0 from MCU.

Source 1 : Start Address = 0, Image Width = canvas_image_width, Coordinate = (0,0) .

Destination: Start Address = 0, Image Width = canvas_image_width, Coordinate = (100,100) .

ROP Code = 12 → Destination data= Source 0 data, Don't care Source 1.

BTE Window Size = 128x128 .

實際畫面:

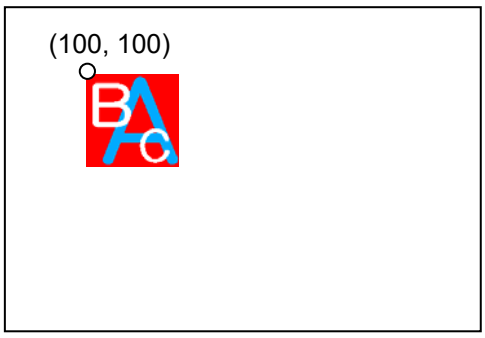


圖 4-21: 用 BTE MCU Write 將圖片寫入 SDRAM 中，目的地位置從(100,100)到(227,227)

4.3.3:MCU Write With Chroma Key (w/o ROP)

BTE 通透性寫入功能(MCU Write With Chroma Key),可以加增加 MCU 端寫入顯示資料至 SDRAM 的傳送速度,一旦 BTE 通透性寫入功能開始,BTE 引擎會持續動作直到所有的像素都被寫入為止。

BTE 通透性寫入功能可用來更新一個 SDRAM 的特定區域,而由 MCU 提供顯示資料來源,不同於前面章節提到的 MCU Write with ROP 的 BTE 功能,BTE 通透性寫入功能會忽略某些特定顏色的操作,此特定的通透色可由使用者設定,在 RA8889 中,此特定通透色設定於暫存器中的「BTE 背景色」中,當讀到來源資料的顏色,為符合通透色設定時,RA8889 便會忽略這部份的顯示資料,不執行寫入的功能。此功能在處理將一張圖片的部分圖形複製到 SDRAM 時很有幫助,不需被複製的地方,在來源圖片中便以「通透色」來處理,在 BTE 通透性寫入功能執行時,便不會進入被寫入 SDRAM。利用此功能可以很快的在任意背景圖上,寫入一個前景圖案。如圖 4-22 為例,來源圖案為一個紅色的背景搭配黃色的圓形圖案,藉著設定紅色為「通透色」,並且執行 BTE 通透性寫入功能,就相當於將一個黃色的圓形圖案,貼到目的地位置的功能。BTE 通透性寫入功能在來源和目的資料設定皆支援"線性"和"區塊"定址模式。

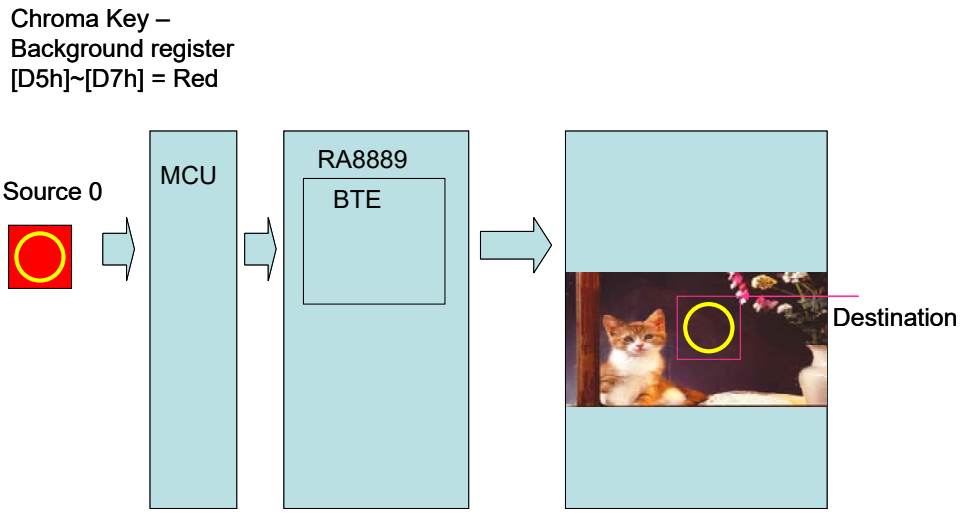


圖 4-22：硬體資料流程

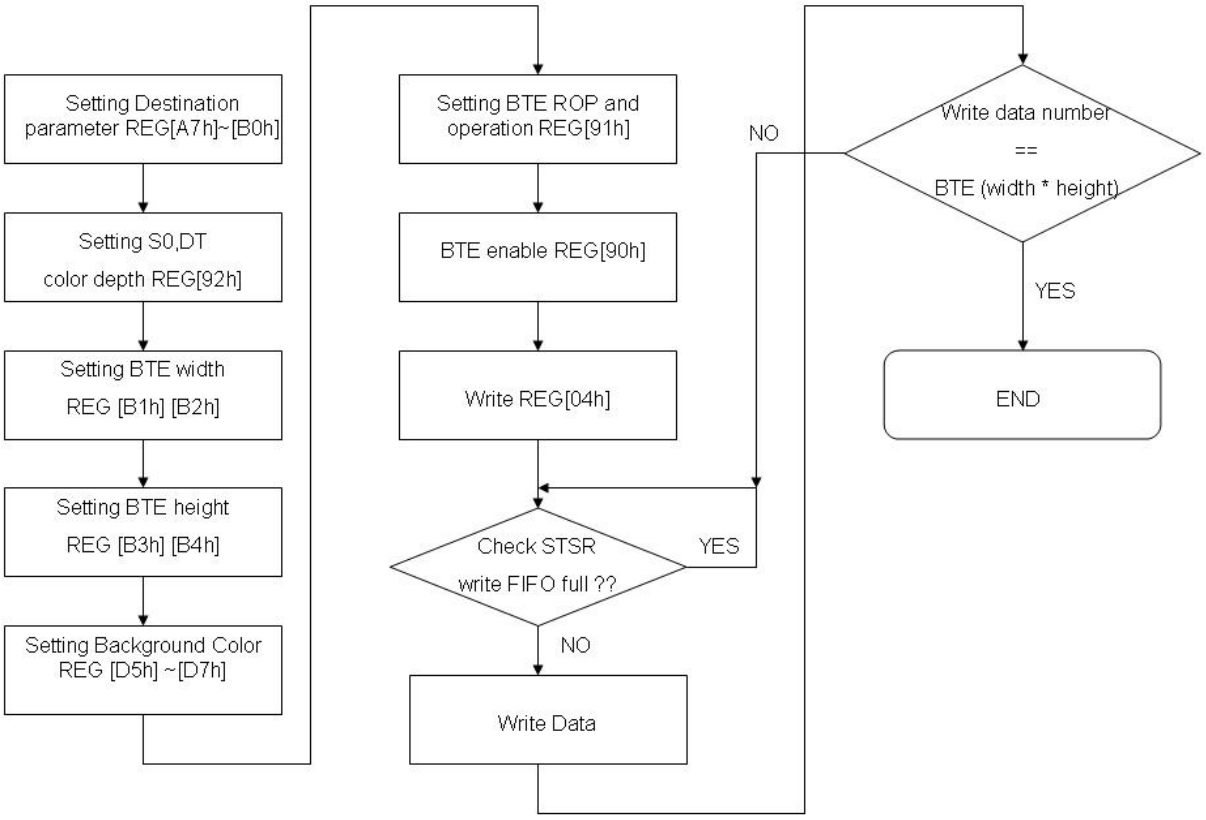

Setting Destination parameter REG[A7h]~[B0h]
Setting S0,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting Background Color REG [D5h] ~[D7h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Write REG[04h]
Check STSR write FIFO full ??
YES
NO
Write Data
Write data number == BTE (width * height)
NO
YES
END

圖 4-23：流程圖

4.3.4:BTE MCU Write With Chroma Key 功能在 LCD 上的顯示結果:

BTE 通透性寫入功能，可以增加 MCU 端寫入顯示資料至 SDRAM 的傳送速度，
一旦BTE，性寫入功能開始通透BTE 。引擎會持續動作直到所有的像素都被寫入為止
不同於前面章節提到的 MCU Write with ROP 的 BTE 功能，MCU Write with chroma key 將忽略 MCU 設定的通透色，」此特定通透色設定於暫存器中的BTE 背景中「色，當讀到來源資料的顏色為被設定的
，通透色時RA8889。不執行寫入的功能，便會忽略這部份的顯示資料舉例來說，如圖 4-25，來源圖片有一個藍色字母 A 與兩個白色字母 B 與 C 在搭配上紅色背景。經由設定紅色為通透色，然後使用 MCU Write with chroma key 功能，就會將背景紅色濾掉，只剩下藍、白色字母的資料。

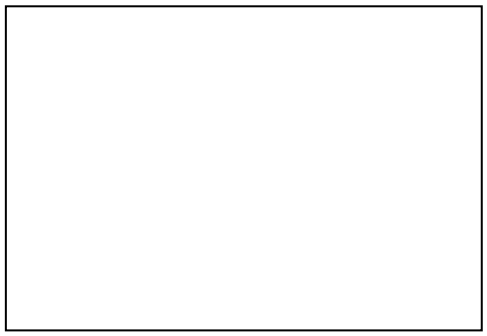


圖 4-24: 在這個範例中，SDRAM 目前的資料

圖 4-25 :MCU 寫入的資料(128x128)

API:

```
void BTE_MCU_Write_Chroma_key_MCU_8bit
(
unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned long Background_color //transparency color
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
,const unsigned char *data // 8-bit data
)


void BTE_MCU_Write_Chroma_key_MCU_16bit
(
unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned long Background_color //transparency color
```

```
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
,const unsigned short *data // 16-bit data
)
```

範例:

```
//Use 8bit MCU , 8bpp color depth
BTE_MCU_Write_Chroma_key_MCU_8bit(0,canvas_image_width,100,100,0xe0,128,128,gImage_8);
or
//Use 8bit MCU , 8bpp color depth
BTE_MCU_Write_Chroma_key_MCU_8bit(0,canvas_image_width,100,100,0xf800,128,128,gImage_16);
or
//Use 8bit MCU , 24bpp color depth
BTE_MCU_Write_Chroma_key_MCU_8bit(0,canvas_image_width,100,100,0xff0000,128,128,gImage_24);
or
//Use 16bit MCU , 16bpp color depth
BTE_MCU_Write_Chroma_key_MCU_16bit(0,canvas_image_width,100,100,0xf800,128,128,pic1616);
//Use 16bit MCU , 24bpp color depth and data format use mode 1
BTE_MCU_Write_Chroma_key_MCU_16bit(0,canvas_image_width,100,100,0xff0000,128,128,pic16241);
or
//Use 16bit MCU , 24bpp color depth and data format use mode 2
BTE_MCU_Write_Chroma_key_MCU_16bit(0,canvas_image_width,100,100,0xff0000,128,128,pic1624);
```

Condition:

Source 0 from MCU,

Destination: Start Address = 0, Image Width = canvas_image_width, Coordinate = (100,100) .

BTE Window Size = 128x128 .

Transparency color = 0xe0 , 0xf800 ,0xff0000 (Red)

In BTE Chroma Key (Transparency color) function Enable, BTE process compare source 0 data and background color register data.

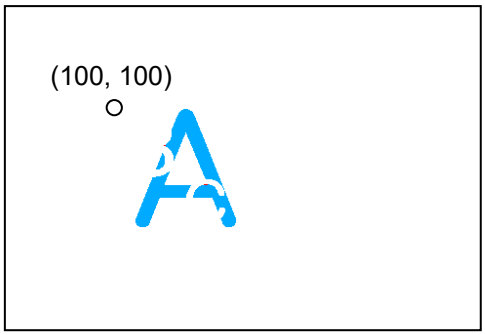


圖 4-26: BTE MCU Write With Choma Key，通透色為紅色，目的地位置從(100,100)到(227,227)。

4.4.1: Memory Copy with ROP

Memory Copy (move) with ROP 的 BTE 功能，是將 SDRAM 的一個特定區塊的資料，搬移到另外一個區塊。這個功能可以加速一個區塊的資料複製，而且能省下很多 MCU 端的執行時間以及負擔。

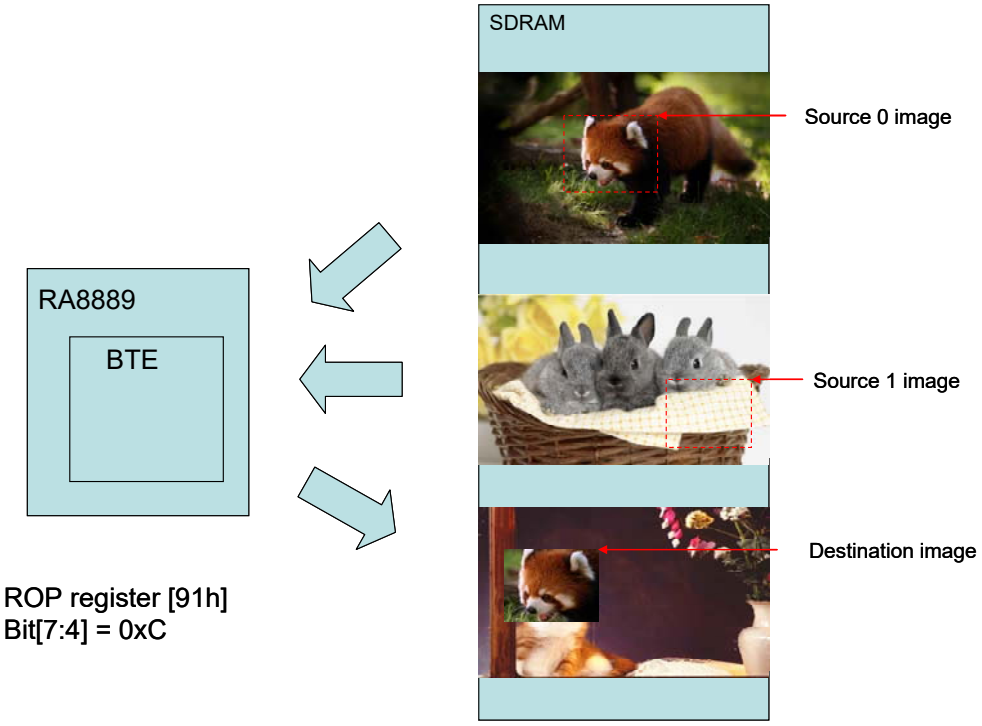


圖 4-27：硬體資料流程

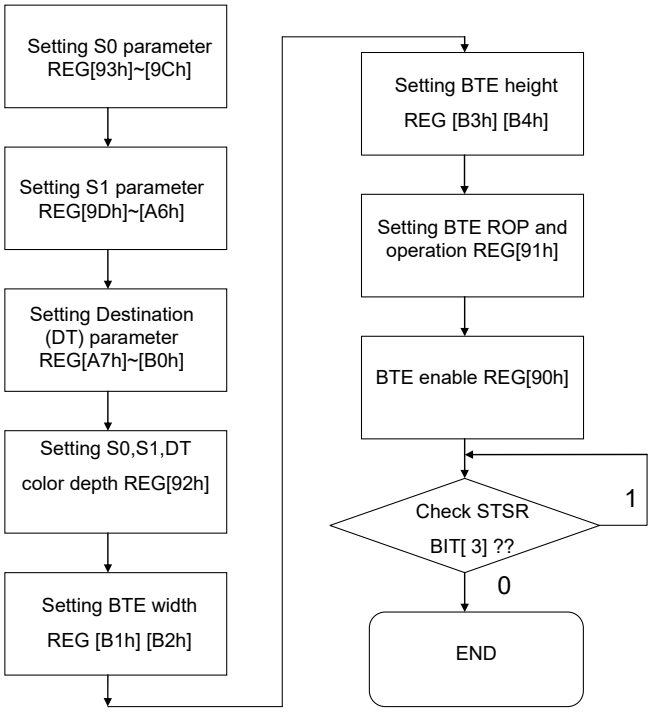


圖 4-28：流程圖

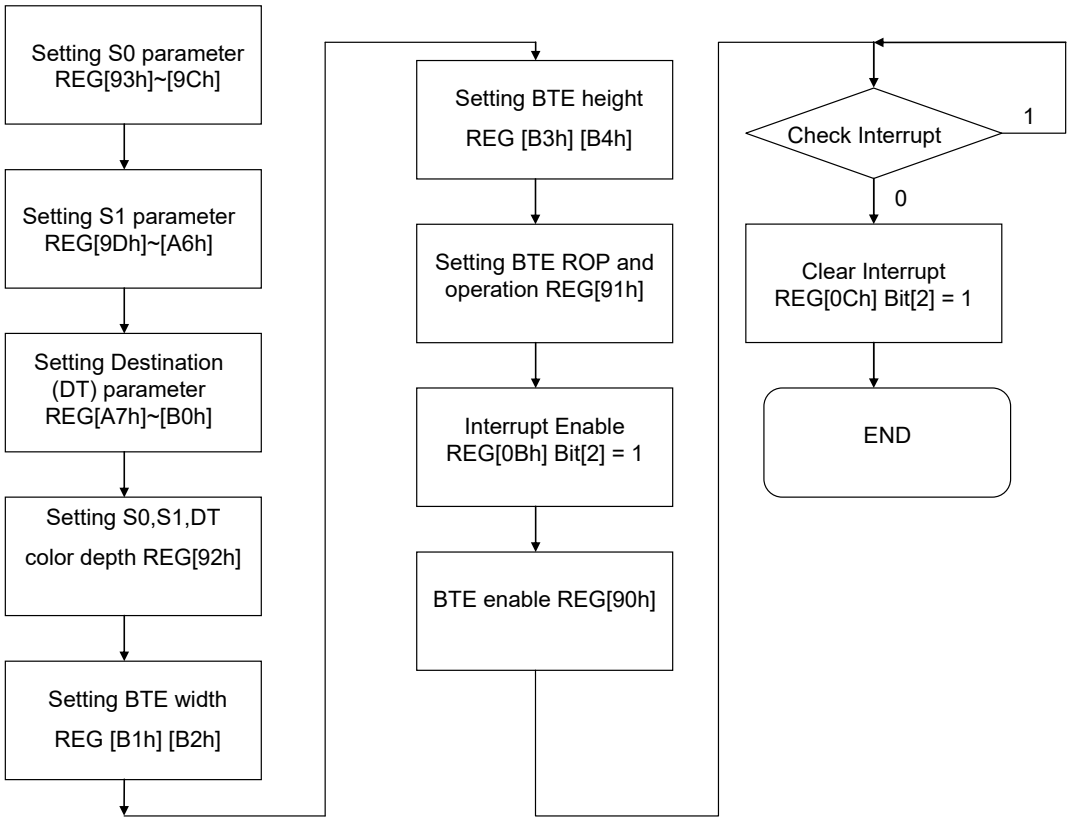


圖 4-29：流程圖 – Check Int

4.4.2: BTE Memory Copy 功能在 LCD 上的顯示結果:

以下為 BTE Memory Copy API 功能的解說與範例，先利用了 BTE Solid Fill 功能畫出一塊填滿紅色的方形，以及用畫圓功能畫出一個黃色的實心圓(如圖 4-30)，再透過 BTE Memory Copy 複製一個一樣的圖案(如圖 4-31)。

```
void BTE_Memory_Copy
(
unsigned long S0_Addr //Start address of Source 0
,unsigned short S0_W //image width of Source 0 (recommend = canvas image width)
,unsigned short XS0 //coordinate X of Source 0
,unsigned short YS0 //coordinate Y of Source 0
,unsigned long S1_Addr //Start address of Source 1
,unsigned short S1_W //image width of Source 1 (recommend = canvas image width)
,unsigned short XS1 //coordinate X of Source 1
,unsigned short YS1 //coordinate Y of Source 1
,unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned int ROP_Code
/*ROP_Code :
    0000b       0(Blackness)
    0001b       ~S0!E~S1 or ~(S0+S1)
    0010b       ~S0!ES1
    0011b       ~S0
    0100b       S0!E~S1
    0101b       ~S1
    0110b       S0^S1
    0111b       ~S0 + ~S1 or ~(S0 + S1)
    1000b       S0!ES1
    1001b       ~(S0^S1)
    1010b       S1
    1011b       ~S0+S1
    1100b       S0
    1101b       S0+~S1
    1110b       S0+S1
    1111b       1(whiteness)*/
,unsigned short X_W //X_W : Width of BTE Window
```

```
,unsigned short Y_H //Y_H : Length of BTE Window
)
```

範例:

```
/*Source 0 : Start Address = 0, Image Width = canvas_image_width, Coordinate = (0,0) .
Source 1 : Start Address = 0, Image Width = canvas_image_width, Coordinate = (0,0)
Destination: Start Address = 0, Image Width = canvas_image_width, Coordinate = (500,200) .
ROP Code = 12 →Destination = Source 0 , Don't care Source 1.
BTE Window Size = 128x128 .*/
BTE_Solid_Fill(0,canvas_image_width,0,0,0xe0,200,200); //8bpp color depth
Draw_Circle_Fill(0xffff00,100,100,50); //8bpp color depth
or
BTE_Solid_Fill(0,canvas_image_width,0,0,0xf800,200,200); //16bpp color depth
Draw_Circle_Fill(0xffff00,100,100,50); //16bpp color depth
or
BTE_Solid_Fill(0,canvas_image_width,0,0,0xFF0000,200,200); //24bpp color depth
Draw_Circle_Fill(0xffff00,100,100,50); //24bpp color depth
+
BTE_Memory_Copy(0,canvas_image_width,0,0,0,canvas_image_width,0,0,0,canvas_image_width,
500,200,12,200,200);
```

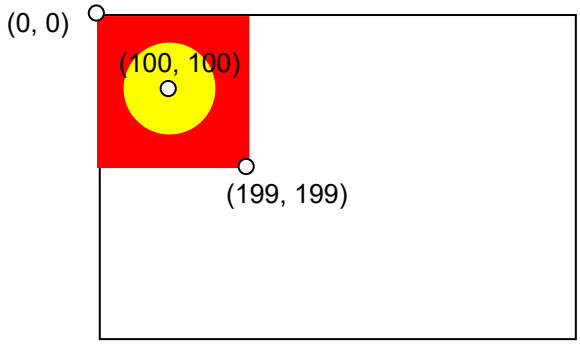


圖 4-30: 繪一黃色的實心圓與紅色實心矩形

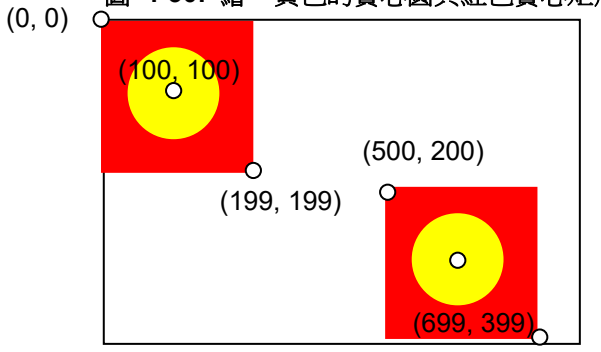


圖 4-31：使用 BTE Memory Copy with ROP 功能複製一個一樣的圖案。

4.4.3:Memory Copy With Chroma Key (w/o ROP)

Memory Copy With Chroma Key (w/o ROP)的 BTE 功能，是將 SDRAM 的一個特定區塊的資料，搬移到另外一個區塊，而且濾掉特定的通透色。而通透色設定是在背景色暫存器。當通透色與被複製的資料是一致時，RA8889 便會忽略這部份的顯示資料，不執行寫入的功能，故該目的地的區塊資料不會有所改變。在這個功能下，來源 0、來源 1 以及目的地的資料區塊都是在 SDRAM 裡面。

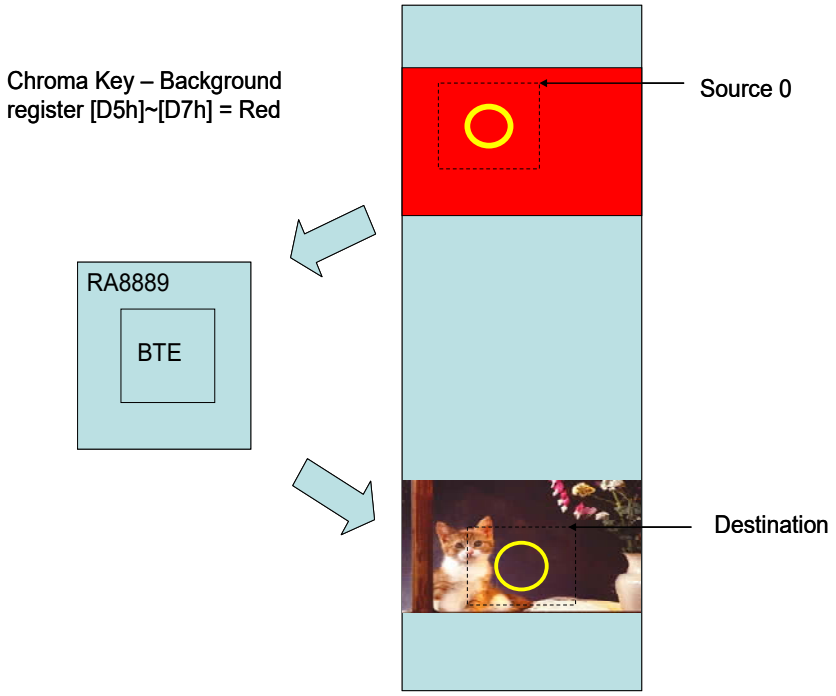


圖 4-32：硬體資料流程

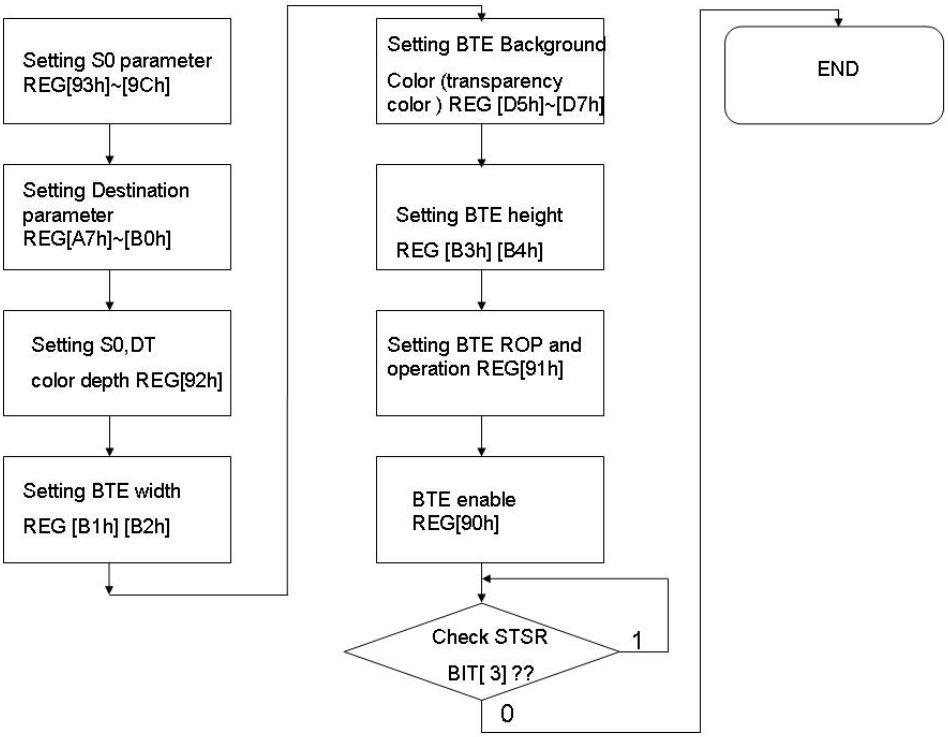


圖 4-33：流程圖

4.4.4 : BTE Memory Copy with Chroma key(w/o ROP)在 LCD 上的顯示說明:

以下為 BTE Memory Copy with Chroma key(w/o ROP) API 功能的解說與範例，先利用了 BTE Solid Fill 功能畫出一塊填滿紅色的方形，以及用畫圓功能畫出一個黃色的實心圓(如圖 4-34)，再透過 BTE Memory Copy with Chroma key(w/o ROP)複製一個將紅色部分濾掉的圖案(如圖 4-35)。

```
void BTE_Memory_Copy_Chroma_key
(
unsigned long S0_Addr //Start address of Source 0
,unsigned short S0_W //image width of Source 0 (recommend = canvas image width)
,unsigned short XS0 //coordinate X of Source 0
,unsigned short YS0 //coordinate Y of Source 0
,unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned long Background_color // transparent color
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
)
```

範例:

```
/*Source 0 : Start Address = 0, Image Width = canvas_image_width, coordinate = (0,0) .
Source 1 : Start Address = 0, Image Width = canvas_image_width, coordinate = (0,0)
Destination: Start Address = 0, Image Width = canvas_image_width, coordinate = (500,200) .
BTE Window Size = 200x200
Transparency color = 0xe0, 0xf800, 0xff0000 (Red)
In BTE Chroma Key (Transparency color) function Enable, BTE process compare source 0 data
and background color register data.*/

//8bpp color depth
BTE_Solid_Fill(0,canvas_image_width,0,0,0xe0,200,200);
Draw_Circle_Fill(0xfc,100,100,50);
BTE_Memory_Copy_Chroma_key(0,canvas_image_width,0,0,0,canvas_image_width,500,200,0xe0,
200,200);
Or
//16bpp color depth
BTE_Solid_Fill(0,canvas_image_width,0,0,0xf800,200,200);
Draw_Circle_Fill(0xf800,100,100,50);
BTE_Memory_Copy_Chroma_key(0,canvas_image_width,0,0,0,canvas_image_width,500,200,0xf80
0,200,200);
```

or

```
//24bpp color depth
BTE_Solid_Fill(0,canvas_image_width,0,0,0xFF0000,200,200);
Draw_Circle_Fill(0xffff00,100,100,50);
BTE_Memory_Copy_Chroma_key(0,canvas_image_width,0,0,0,canvas_image_width,500,200,0xff0000,200,200);
```

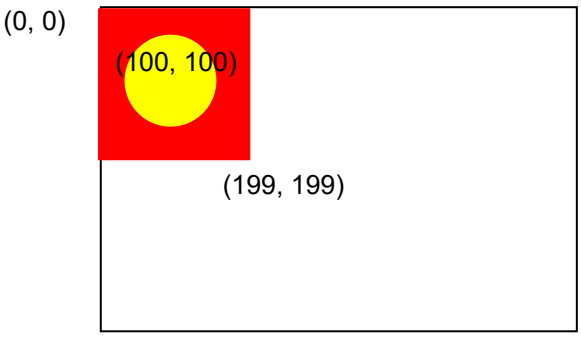


圖 4-34: 繪一黃色的實心圓與紅色實心矩形

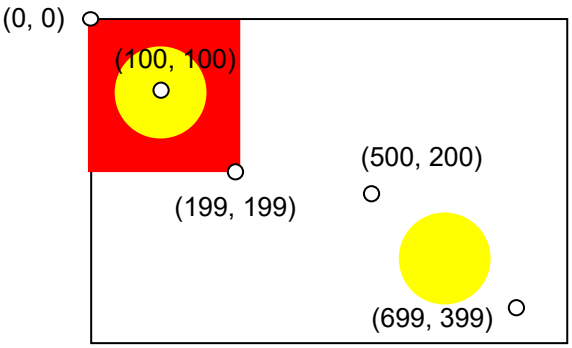


圖 4-35: 使用 **BTE Memory Copy with chroma key** 功能複製黃色的實心圓，並過濾掉紅色實心矩形的部分。

4.5.1: MCU Write With Color Expansion

MCU Write with Color Expansion 是一個很有用的功能，用來處理 MCU 的單色圖形資料轉換為彩色圖形資料，並寫入 SDRAM 中。此功能的來源資料為 MCU 提供的單色圖形資料 (Monochromes Bitmap)。而每一個位元根據內容被轉換為 BTE 前景色或背景色。若單色資料來源為"1" 則會被轉換為 BTE 前景色，若為"0" 則會轉換為 BTE 背景色。此功能可以大大降低將單色系統資料轉換為彩色系統資料的成本。顏色擴充功能會根據 MCU 的資料匯流排寬度，持續讀入 16 位元或 8 位元的資料做轉換，並且可以位元為單位，設定每一行的第一筆單色圖形資料的起始轉換位元，並且在每一行的最後一筆資料讀入後，超過範圍的位元也會被忽略而不寫入，而下一行則從下一筆資料開始執行同樣的操作。這樣以位元為單位的運算大大增加此功能的彈性。另外，每一筆資料的處理方向是從最高位元 (MBS) 至最低位元 (LSB)。

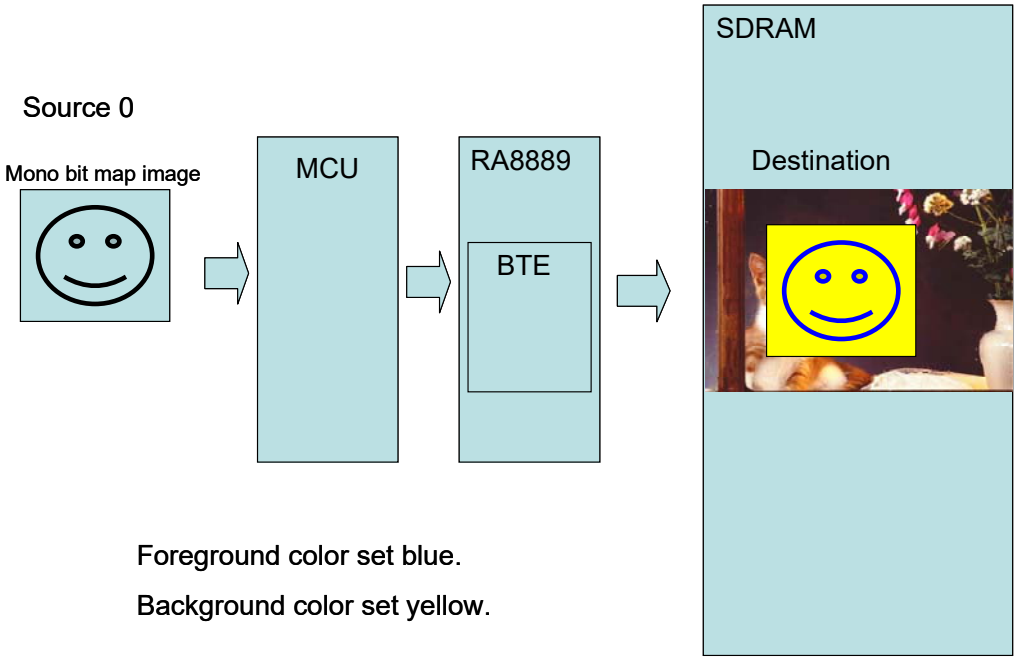


圖 4-36：硬體資料流程

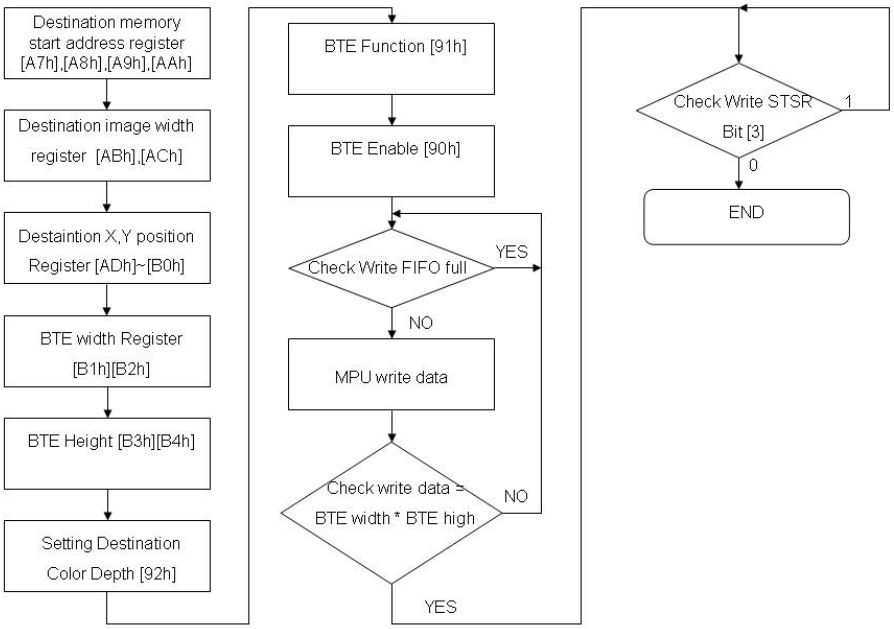


圖 4-37：流程圖

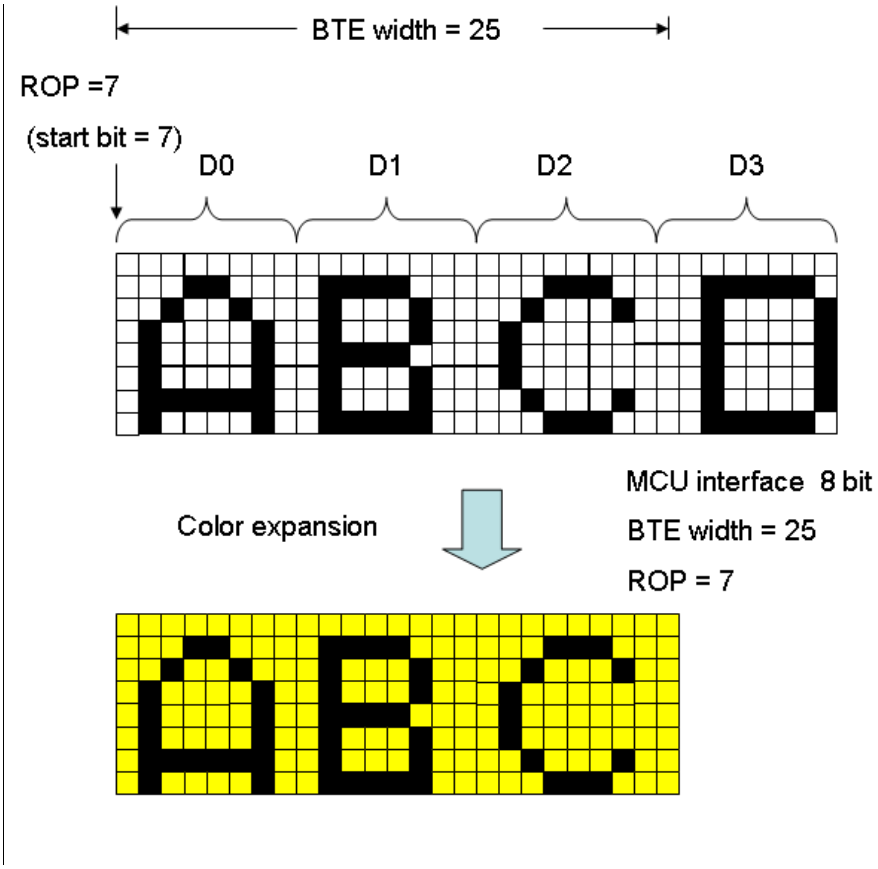


圖 4-38：起始位元範例一

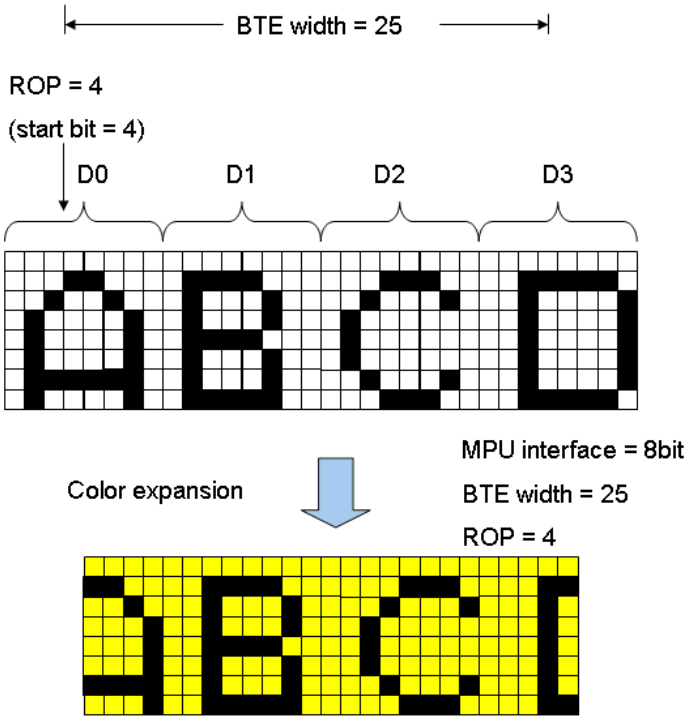


圖 4-39：起始位元範例二

Note:

1. Calculate sent data numbers per row = ((BTE Width size REG – (MCU interface bits – (start bit + 1))) / MCU interface bits) + ((start bit + 1) % (MCU interface))

2. Total data number = (sent data numbers per row) x BTE Vertical REG setting

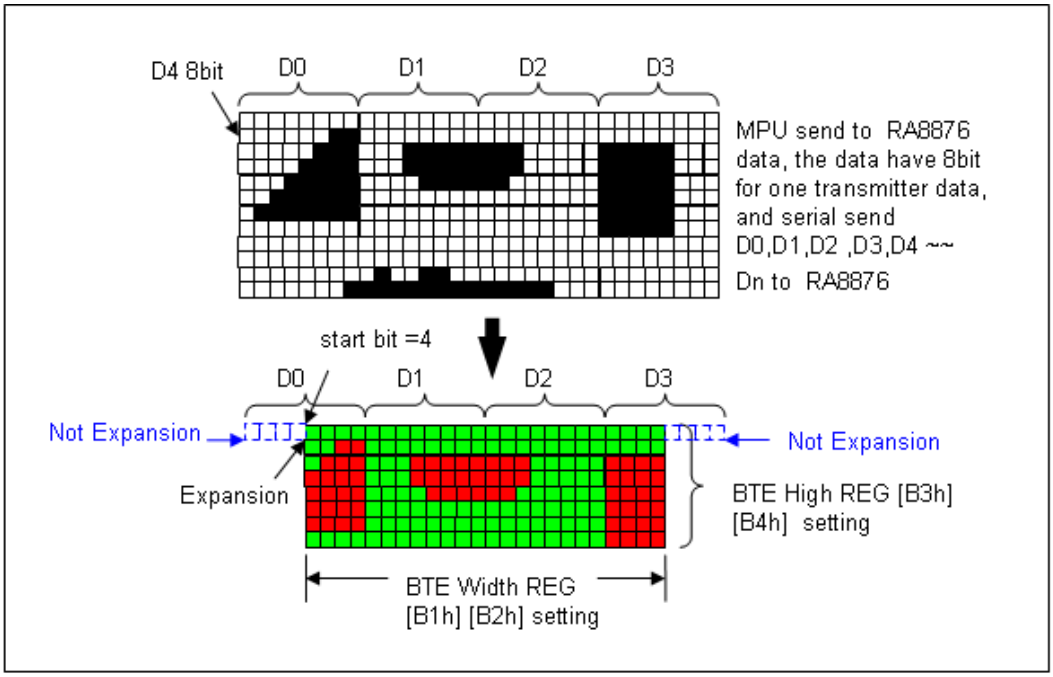


圖 4-40：Color Expansion 資料圖

4.5.2: BTE MCU Write with Color Expansion 在 LCD 上的顯示說明:

圖 4-41 為 128x128 的單色黑白圖像，假設前景色設定為綠色，背景色設定為藍色，使用了 BTE MCU Write with Color Expansion 功能後，就會轉出像圖 4-42 的圖案一樣。以下我們針對 MCU 8bit 與 16bit 各提供一組 API 以及說明與範例供使用者參考。

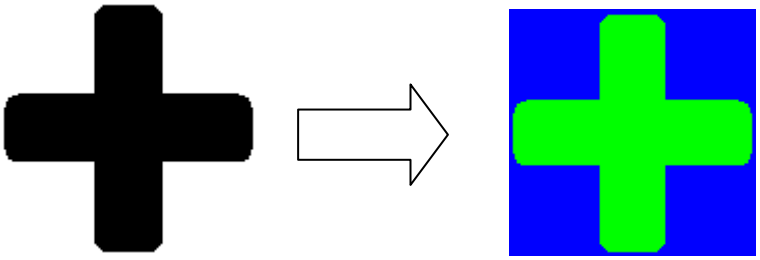


圖 4-41：
128x128 黑白單色圖資

圖 4-42：
BTE with color expansion

BTE MCU Write with Color Expansion API:

```
void BTE_MCU_Write_ColorExpansion_MCU_8bit
(
unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
,unsigned long Foreground_color
/*Foreground_color : The source (1bit map picture) map data 1 translate to Foreground color by color expansion*/
,unsigned long Background_color
/*Background_color : The source (1bit map picture) map data 0 translate to Foreground color by color expansion*/
,const unsigned char *data // 8-bit data
)
```

```
void BTE_MCU_Write_ColorExpansion_MCU_16bit
(
unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
,unsigned long Foreground_color
/*Foreground_color : The source (1bit map picture) map data 1 translate to Foreground color by color expansion*/
,unsigned long Background_color
/*Background_color : The source (1bit map picture) map data 0 translate to Background color by color expansion*/
,const unsigned short *data //16-bit data
)
```

範例:

```
/*Des_Addr : start address of Destination = 0
Des_W : image width of Destination (recommend = canvas image width)
XDes : coordinate X of Destination = 0
YDes : coordinate Y of Destination =0
X_W : Width of BTE Window =128
Y_H : Length of BTE Window =128
Foreground_color : The source (1bit map picture) map data 1 translate to Background color by color expansion = 0x03(8bpp)、0x001f(16bpp)、0x0000ff(24bpp) (Blue)
Background_color : The source (1bit map picture) map data 0 translate to Foreground color by color expansion = 0x1c(8bpp)、0x07e0(16bpp)、0x00ff00(24bpp) (Green)
When ColorDepth =8bpp */

//MCU_8bit_ColorDepth_8bpp                    //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_8bit(0,canvas_image_width,0,0,128,128,0x03,0x1c,gImage_1);
or
//MCU_8bit_ColorDepth_16bpp                   //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_8bit(0,canvas_image_width,0,0,128,128,0x001f,0x07e0,gImage_1);
or
//MCU_8bit_ColorDepth_24bpp                   //setting in UserDef.h
```

```
BTE_MCU_Write_ColorExpansion_MCU_8bit(0,canvas_image_width,0,0,128,128,0x0000ff,0x00ff00,glma
ge_1);
or
//MCU_16bit_ColorDepth_16bpp                    //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_16bit(0,canvas_image_width,0,0,128,128,0x001f,0x07e0,Test);
or
//MCU_16bit_ColorDepth_24bpp_Mode_1             //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_16bit(0,canvas_image_width,0,0,128,128,0x0000ff,0x00ff00,Tes
t);
or
//MCU_16bit_ColorDepth_24bpp_Mode_2             //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_16bit(0,canvas_image_width,0,0,128,128,0x0000ff,0x00ff00,Tes
t);
```

LCD 上的顯示畫面:

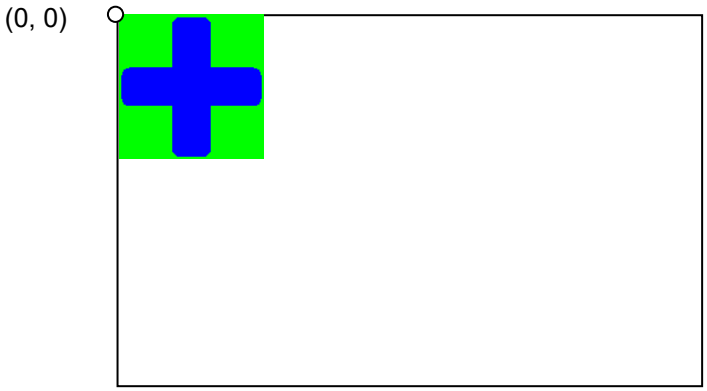


圖 4-43 : BTE MCU Write with Color Expansion

4.5.3: MCU Write with Color Expansion and Chroma key

除了將 BTE 的背景色忽略，用來當作通透色，此 BTE 功能與 BTE MCU Write with Color Expansion 功能幾乎是相同的。就是所有輸入單色資料值為"1" 的位元將會被轉換為 BTE 的前景色並且寫入目的位置，所有輸入單色資料值為"0" 的位元將不被轉換，而保持原來的目的資料顏色值。

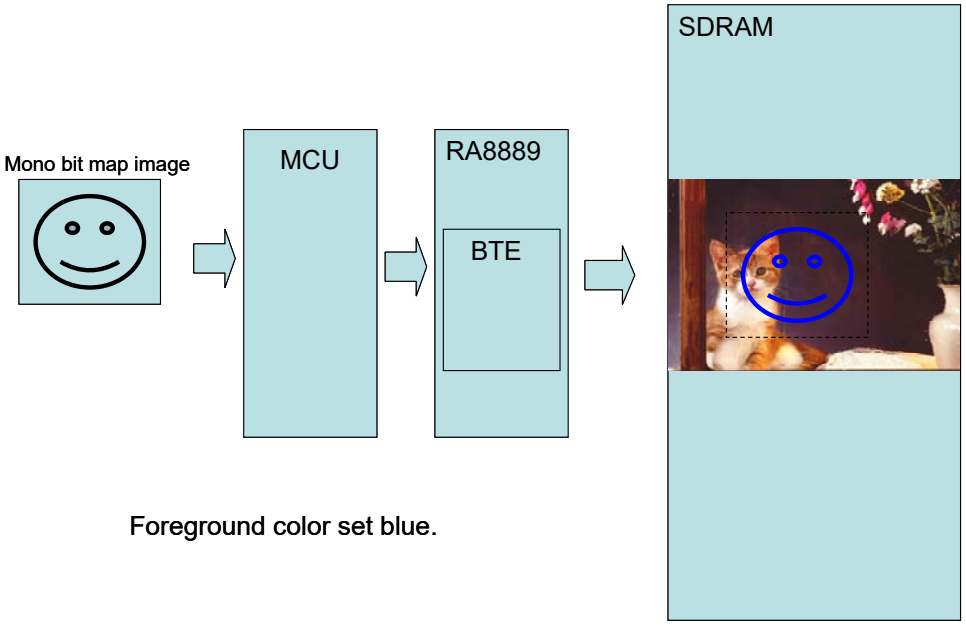


圖 4-44：硬體資料流程

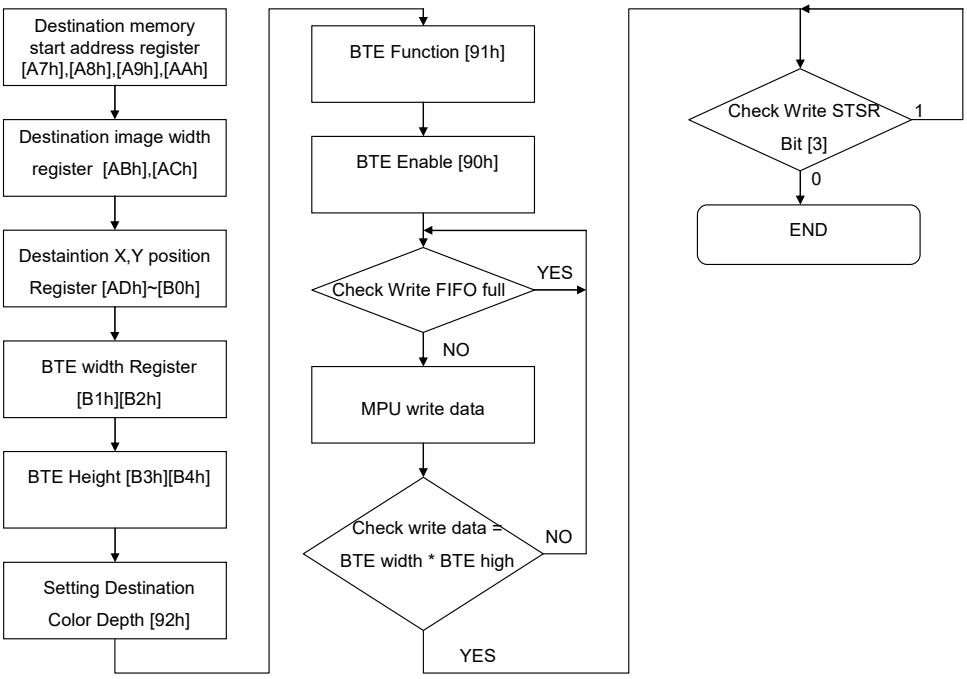


圖 4-45：流程圖

4.5.4: BTE MCU Write With Color Expansion and Chroma key 的 API 在 LCD 上的顯示說明:

圖 4-46 為範例所用到的 128x128 黑白的單色圖，透過 BTE MCU Write With Color Expansion and Chroma key 功能，可以轉換成其他顏色的圖案，圖 4-47 就是將圖 4-46 使用了此功能後將原本的圖轉換成綠色。以下我們也分別針對 MCU8bit 與 16bit 提供了兩組 API 以及範例說明。

圖 4-46 :
128x128 黑白單色圖資

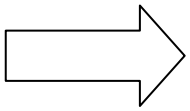


圖 4-47 :
BTE with color expansion and Chroma key

API:

```
void BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_8bit
(
unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
,unsigned long Foreground_color
/*Foreground_color : The source (1bit map picture) map data 1 translate to Foreground color by color expansion*/
,const unsigned char *data //8-bit data
)

void BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit
(
unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate X of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned short X_W //Width of BTE Window
,unsigned short Y_H //Length of BTE Window
```

```
,unsigned long Foreground_color
/*Foreground_color : The source (1bit map picture) map data 1 translate to Foreground color by color expansion*/
,const unsigned short *data //16-bit data
)
```

範例:

```
/*Des_Addr : start address of Destination = 0
Des_W : image width of Destination (recommend = canvas image width) =canvas_image_width
XDes : coordinate X of Destination = 0
YDes : coordinate Y of Destination =0
X_W : Width of BTE Window =128
Y_H : Length of BTE Window =128
Foreground_color : The source (1bit map picture) map data 1 translate to Foreground color by color expansion = 0xe0 (8bpp)、0xf800 (16bpp)、0xff0000 (24bpp) (Red)*/

//MCU_8bit_ColorDepth_8bpp                    //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_8bit(0,canvas_image_width,0,0,128,128,0xe0,glmage_1);
or
//MCU_8bit_ColorDepth_16bpp                   //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_8bit(0,canvas_image_width,0,0,128,128,0xf800,glmage_1);
or
//MCU_8bit_ColorDepth_24bpp                   //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_8bit(0,canvas_image_width,0,0,128,128,0xff0000,glmage_1);
or
//MCU_16bit_ColorDepth_16bpp                  //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit(0,canvas_image_width,0,0,128,128,0xf800,Test);
or
//MCU_16bit_ColorDepth_24bpp_Mode_1           //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit(0,canvas_image_width,0,0,128,128,0xff0000,Test);
or
//MCU_16bit_ColorDepth_24bpp_Mode_2           //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit(0,canvas_image_width,0,0,128,128,0xff0000,Test);
```

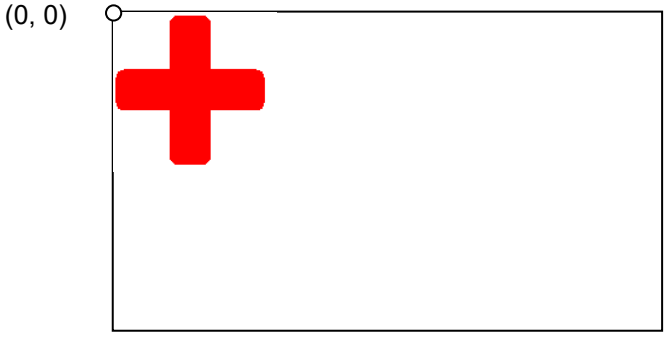


圖 4-48 : BTE MCU Write With Color Expansion and Chroma key (w/o ROP)

4.6 :BTE Alpha Blending

概要:

在許多顯示功能中，alpha blending 是一張圖像與其他層的顯示資料在畫面上做出半透明的效果。RA8889 也提供了硬體做 Alpha Blending 的功能，使用者不用花費很多的系統資源，就可以得到更絢麗的顯示效果。這份應用說明書將幫助我們的使用者，去瞭解並使用 RA8889 的 Alpha Blending 功能。

4.6.1: Memory Copy with Alpha Blending

“Memory Copy with opacity” 可以混合來源 0 資料與來源 1 資料然後再寫入目的記憶體。這個功能有兩個模式– **Picture 模式與 Pixel 模式**。Picture 模式可以被操作在 8 bpp/16bpp/32bpp 色深下並且對於全圖只具有一種混合透明度 (alpha level)，混合度被定義在 REG[B5h]。Pixel 模式只能被操作在來源 1 端是 8bpp/16bpp 模式，而各個 Pixel 具有其各自的混合度，在來源 1 為 16bpp 色深下像素的 bit [15:12] 是透明度 (alpha level)，剩餘的 bit 則為色彩資料；而來源 1 為 8bpp 色深情形下像素 bit [7:6] 是透明度 (alpha level)， Bit [5:0] 則是被使用在索引調色盤 (palette color) 的顏色。根據 32bpp 色深下的像素圖像，必須將 S1 顏色深度設置為 16bpp，並且必須將 S1 寬度設置為與原始圖像相同的寬度(寬度)。 在 32-bit 像素模式下，S1 圖像的 bit [31:24]代表其 alpha 值，bit [23：0]代表像素數據。**錯誤! 找不到參照來源。**顯示了有關如何通過 MPU 接口將αRGB 圖像數據寫入 SDRAM 的流程。

Picture mode 的目的地資料 = (來源 0* (1- alpha Level)l) + (來源 1 * alpha Level)

Pixel mode 32bpp - 下的目的地資料 = (來源 0* (1 - alpha Level)) + ((來源 1 [24:0] * alpha Level)

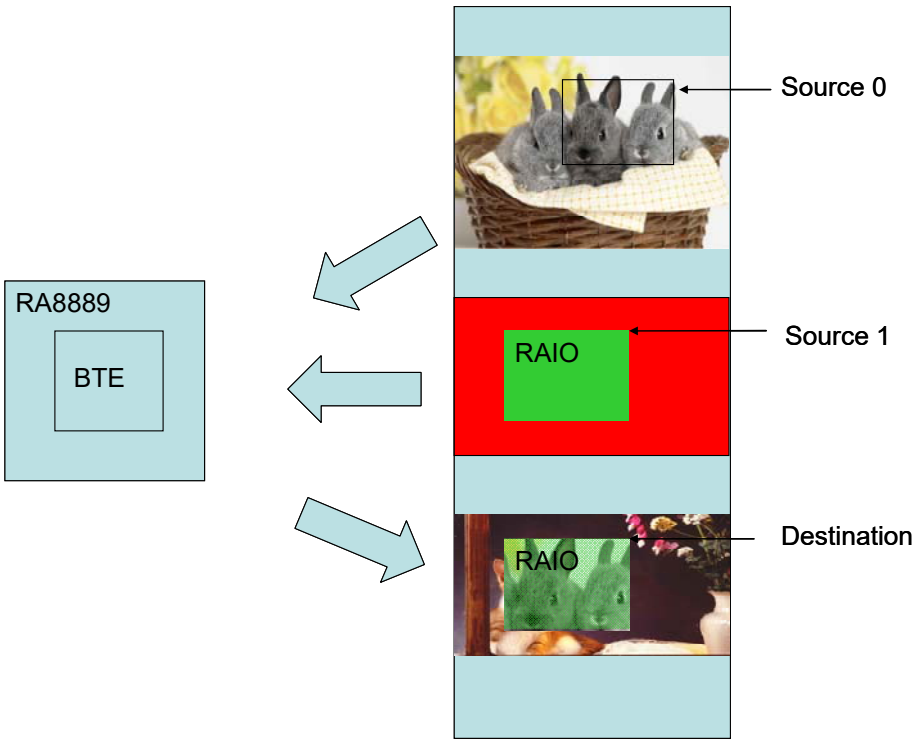


圖 4-49 : Picture Mode 硬體資料流程

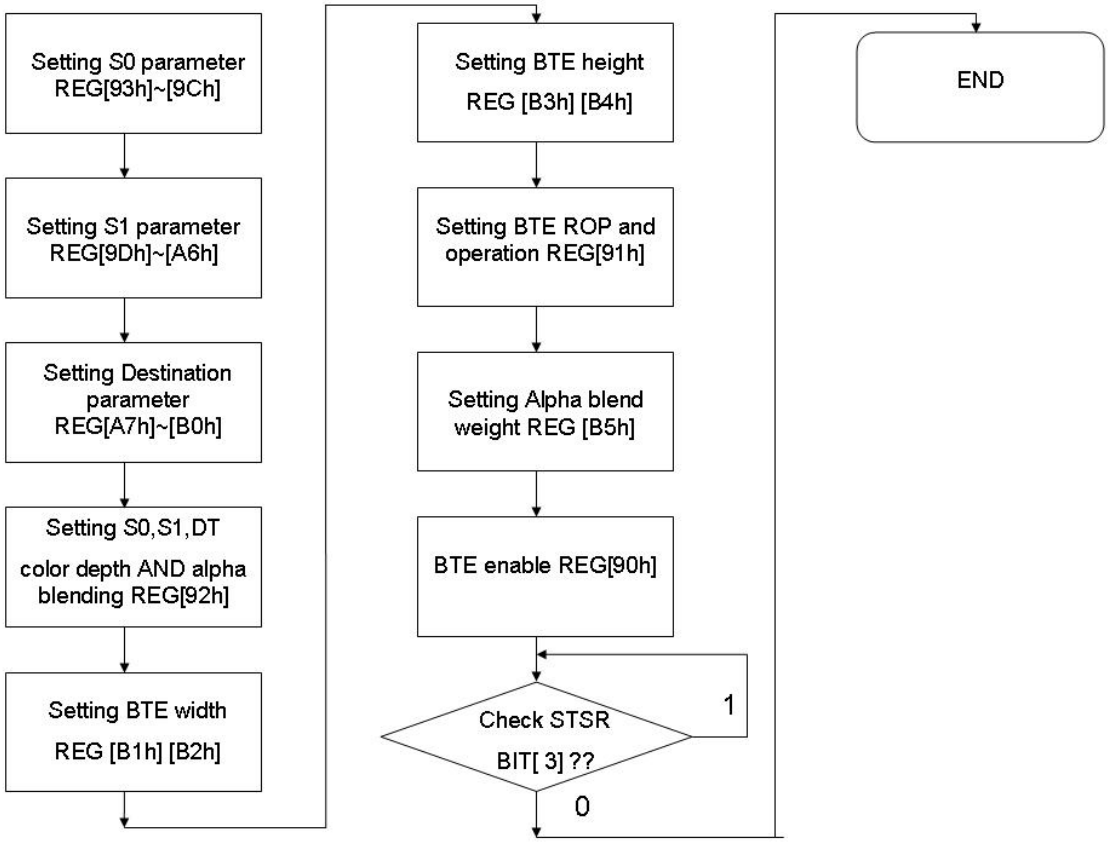


圖 4-50 : Picture Mode 流程圖

4.6.2: BTE Memory Copy with Alpha Blending in Picture mode 在 LCD 上的顯示說明:

“Memory Copy with Alpha blending”這個功能可以混合來源 0 與來源 1 的資料，然後顯示在目的地的顯示區塊。

這個功能有兩種模式 – Picture mode 與 Pixel mode，在這邊，我們使用 picture mode 做說明。

Picture mode 目的地資料 = (來源 0 * (1- alpha Level)l) + (來源 1 * alpha Level)

```
void BTE_Alpha_Blending
(
unsigned long S0_Addr //Start address of Source 0
,unsigned short S0_W //image width of Source 0 (recommend = canvas image width)
,unsigned short XS0 //coordinate X of Source 0
,unsigned short YS0 //coordinate Y of Source 0
,unsigned long S1_Addr //Start address of Source 1
,unsigned short S1_W //image width of Source 1 (recommend = canvas image width)
,unsigned short XS1 //coordinate X of Source 1
,unsigned short YS1 //coordinate Y of Source 1
,unsigned long Des_Addr //start address of Destination
,unsigned short Des_W //image width of Destination (recommend = canvas image width)
,unsigned short XDes //coordinate Y of Destination
,unsigned short YDes //coordinate Y of Destination
,unsigned short X_W //Width BTE Window
,unsigned short Y_H //Length BTE Window
,unsigned char alpha
//alpha : Alpha Blending effect 0 ~ 32, Destination data = (Source 0 * (1- alpha)) + (Source 1 * alpha)
)
```

範例:

```
/*Source 0 : Start Address = 0, Image Width = canvas_image_width, coordinate = (500,300) .
Source 1 : Start Address = 0, Image Width = canvas_image_width, coordinate = (800,0) .
Destination: Start Address = 0, Image Width = canvas_image_width, coordinate = (500,300) .
BTE Window Size = 200x200 , alpha = 16 .*/
SPI_NOR_initial_DMA (3,0,1,1,0);
+
//When Color Depth = 8bpp                                    /
DMA_24bit(2,0,0,800,480,800,15443088);
BTE_Solid_Fill(0,canvas_image_width,800 ,0,0x1c,200,200);
or
//When Color Depth = 16bpp
DMA_24bit(2,0,0,800,480,800,14675088);
BTE_Solid_Fill(0,canvas_image_width,800 ,0,0x07e0,200,200);
or
```

//When Color Depth = 24bpp

DMA_24bit(2,0,0,800,480,800,0);

BTE_Solid_Fill(0,canvas_image_width,800 ,0,0x00FF00,200,200);

+

BTE_Alpha_Blending_Picture_Mode(0,canvas_image_width,500,300,0,canvas_image_width,800,0,0,canvas_image_width,500,300,200,200,16);

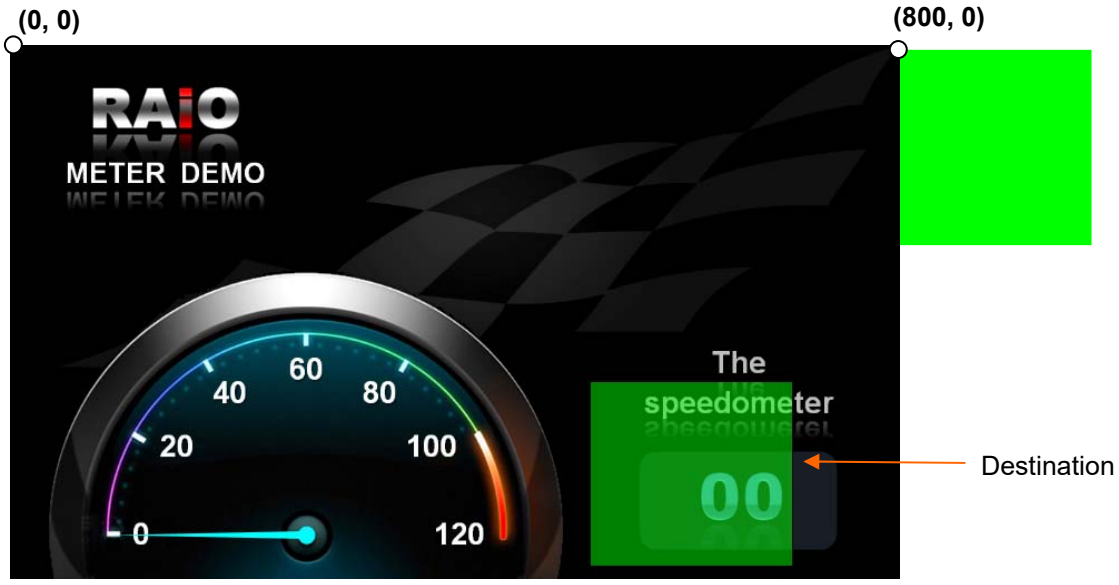


Figure 4-51：來源[(500,300) ~ (699,499)] ,來源 1[(800,0) ~ (999,199)] 執 alpha blending and showing on Destination[(500,300) ~ (699,499)], alpha = 16.

Destination:

Figure 4-52: Destination in different alpha/transparent level.

Picture mode - Destination data = (Source 0 * (1- alpha Level)l) + (Source 1 * alpha Level);

4.6.2: BTE Memory Copy with Alpha Blending in Pixel mode 在 LCD 上的顯示說明

```
void BTE_alpha_blending_32bit_Pixel_mode
(
unsigned int picture_Width //pic width
,unsigned int BTE_X //BTE window size of x
,unsigned int BTE_Y//BTE window size of y
,unsigned long S0X //source 0 coordinate of x
,unsigned long S0Y//source 0 coordinate of y
,unsigned long S0_Start_Addr //source 0 start addr
,unsigned long S0_canvas_width //recomamnd = canvas_image_width
,unsigned long desX//Destination coordinate of x
,unsigned long desY//Destination coordinate of y
,unsigned long DES_Start_Addr//Destination start addr
,unsigned long DES_canvas_width//recomamnd = canvas_image_width
,unsigned long pic_buffer_Layer//source 1 pic addr
)
```

範例:

Step 1:寫一張 PNG(32bit ARGB) 圖資到 SDRAM 中

```
SPI_NOR_initial_DMA (3,0,1,1,0);
SPI_NOR_DMA_png (14623888,nand_buff,0,80,80);
SPI_NOR_DMA_png (14649488,nand_buff+25600,0,80,80);
```

PNG Pic

Step 2:寫入一張 JPG 圖片到 SDRAM

SPI_NOR_initial_JPG_AVI (1,0,1,2,1);

JPG_NOR (1152000,42237,canvas_image_width,0,0);

JPG Pic

Step 3:來源 0 = JPG PIC，來源 1 = ARGB Data，執行 alpha blending pixel mode

BTE_alpha_blending_32bit_Pixel_mode(80,80,80,600,400,0,canvas_image_width,600,400,0,canvas_image_width,nand_buff);

BTE_alpha_blending_32bit_Pixel_mode(80,80,80,700,400,0,canvas_image_width,700,400,0,canvas_image_width,nand_buff+25600);

執行 Alpha blending pixel mode 的顯示畫面

第五章 ：Picture In Picture Function

5.1:PIP Window

RA8889 在主要顯示視窗裡，支援使用者可以使用兩個 PIP 視窗。PIP 視窗不支援通透功能，它僅提供使用者開啟或關閉將圖片資料覆蓋在主顯示畫面上。當 PIP1 與 PIP2 畫面重疊時，PIP1 會覆蓋在 PIP2 之上。PIP 視窗的位置與大小由 REG[2Ah]到 REG[3Bh]與 REG[11h]來設定，PIP1 與 PIP2 共用設定暫存器，而且根據 REG[10h] 的 Bit[4]，去選擇 REG[2Ah~3Bh]是 PIP1 或是 PIP2 視窗的參數。而相關的 PIP window 功能可經由若干功能位元做設置。PIP 視窗大小與起始位置水平方向是 4 個像素為一個單位，垂直則為一個像素。

5.2:PIP Windows Settings

一個 PIP 視窗的位置與大小是由 PIP 圖像起始位址、PIP 圖像寬度、PIP 顯示 X/Y 座標、PIP 圖像 X/Y 座標、PIP 視窗的色彩深度、PIP 視窗的寬度與高度等等的暫存器來界定。PIP1 與 PIP2 共用設定暫存器，然後根據 REG[10h]Bit[4]來選擇 REG[2Ah~3Bh]是 PIP1 或 PIP2 視窗的參數設定。

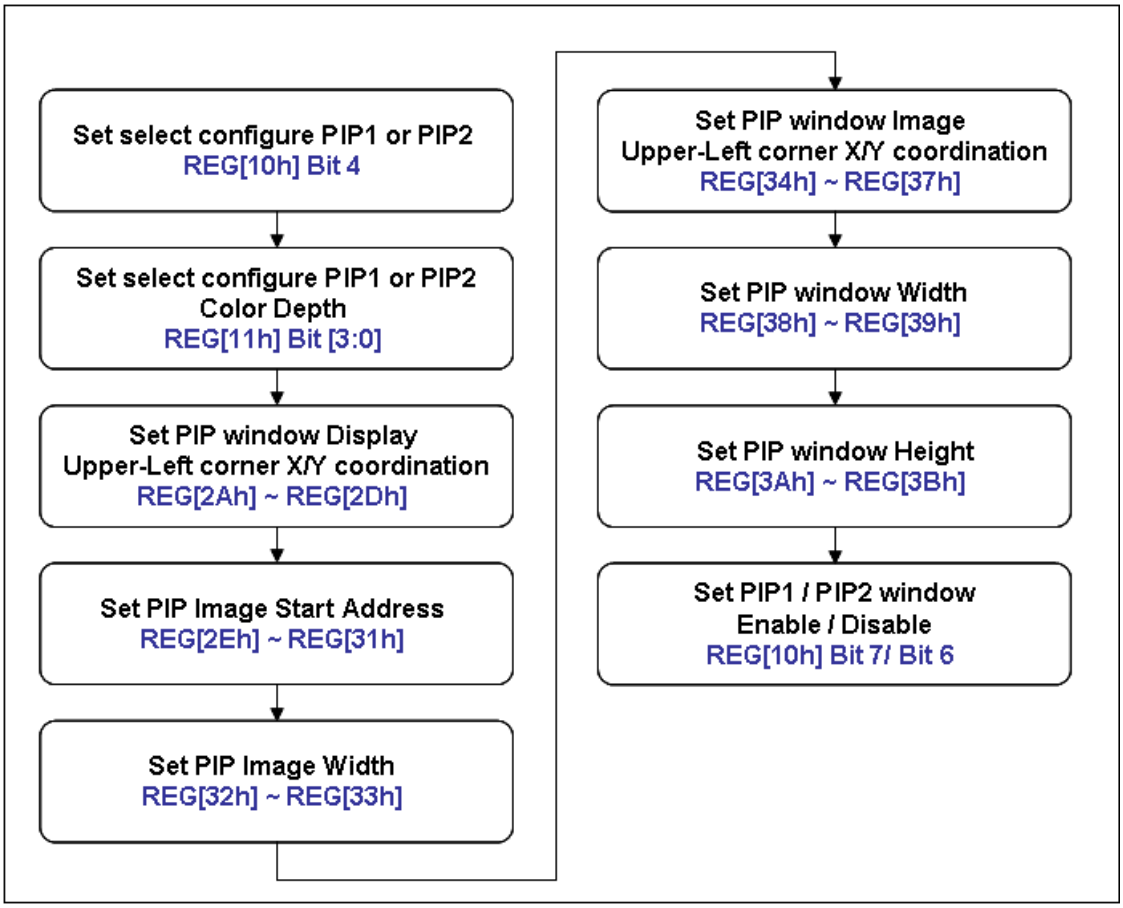


圖 5-1： 流程圖

5.3: PIP 功能圖解:

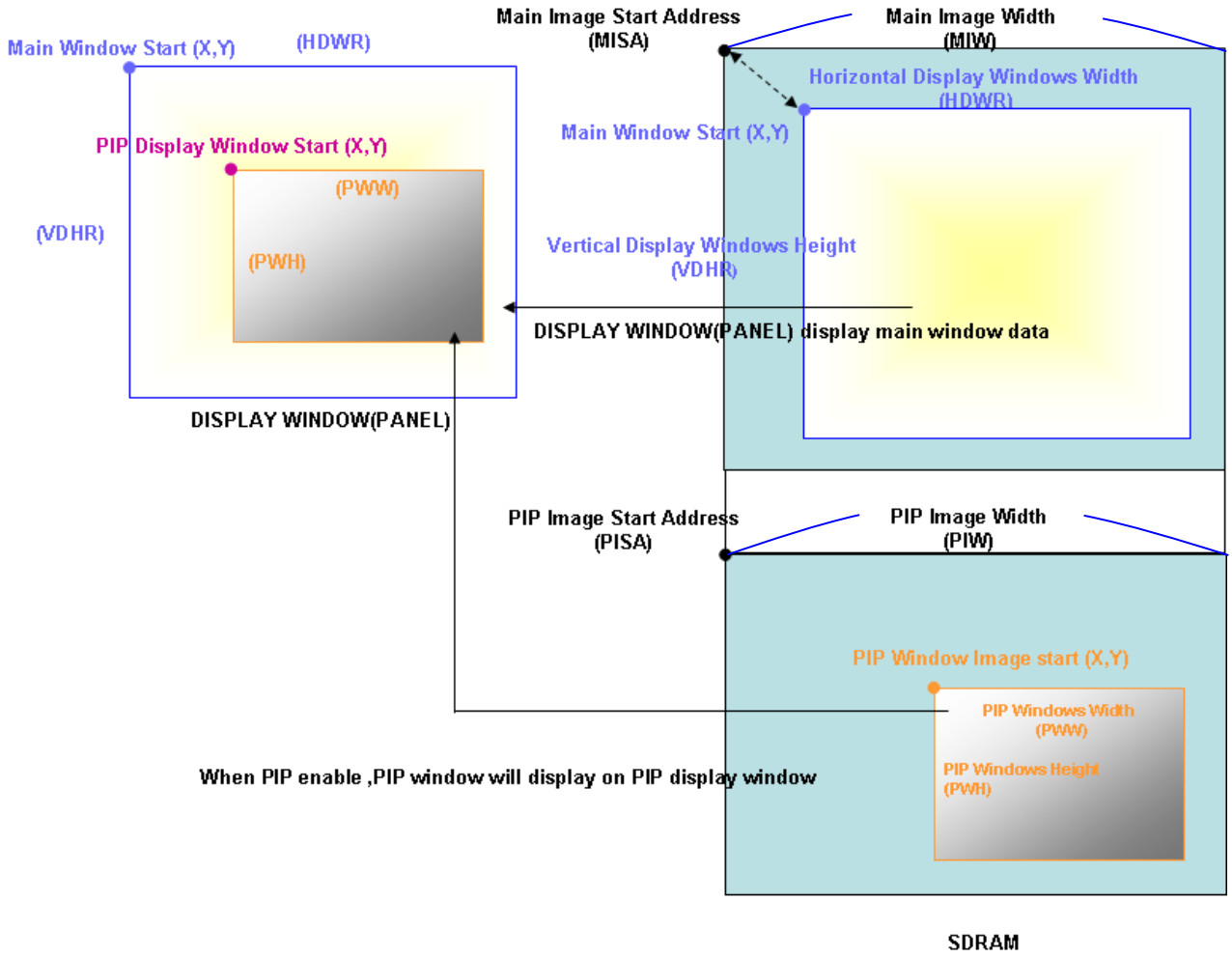


圖 5-2

5.4:PIP 功能在 LCD 上的顯示說明:

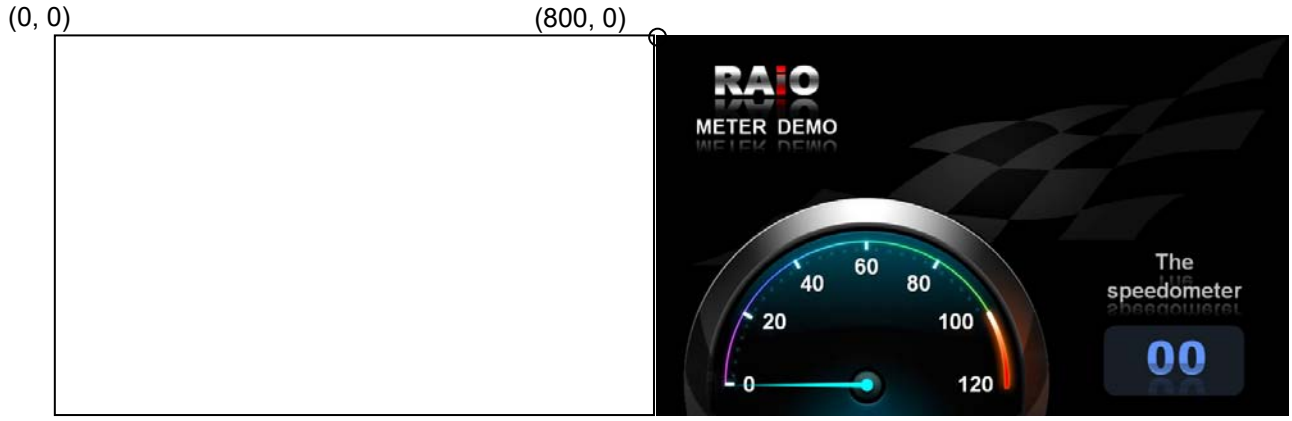
RA8889 支援使用者可以使用兩個 PIP 視窗。PIP 視窗不支援通透功能，它僅提供使用者開啟或關閉將圖片資料覆蓋在主顯示畫面上。當 PIP1 與 PIP2 畫面重疊時，PIP1 會覆蓋在 PIP2 之上。這個應用程式介面將會幫助使用者容易的去使用 PIP 這個功能。

```
void PIP
(
unsigned char On_Off // 0 : disable PIP, 1 : enable PIP, 2 : To maintain the original state
,unsigned char Select_PIP // 1 : use PIP1 , 2 : use PIP2
,unsigned long PAddr //start address of PIP
,unsigned short XP //coordinate X of PIP Window, It must be divided by 4.
,unsigned short YP //coordinate Y of PIP Window, It must be divided by 4.
,unsigned long ImageWidth //Image Width of PIP (recommend = canvas image width)
,unsigned short X_Dis //coordinate X of Display Window
,unsigned short Y_Dis //coordinate Y of Display Window
,unsigned short X_W //width of PIP and Display Window, It must be divided by 4.
,unsigned short Y_H //height of PIP and Display Window , It must be divided by 4.
)
```

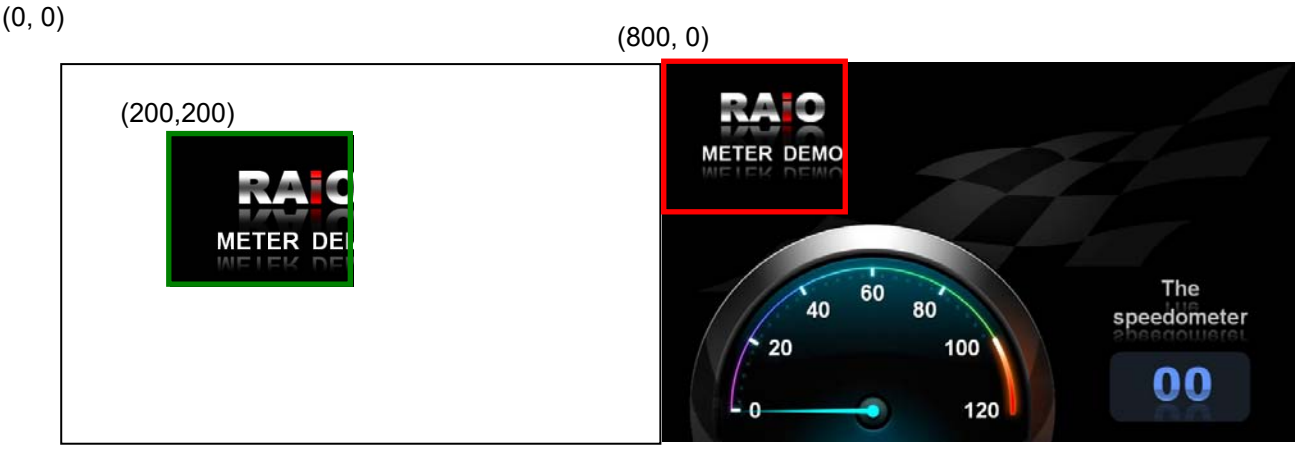
範例: (DMA 功能部分請參考第三章)

```
SPI_NOR_initial_DMA (3,0,1,1,0);
+
//When color depth = 8bpp
DMA_24bit(2,800,0,800,480,800,15443088);
Or
//When color depth = 16bpp
DMA_24bit(2,800,0,800,480,800,14675088);
Or
//When color depth = 24bpp
DMA_24bit(2,800,0,800,480,800,0);
+
PIP(1,2,0,800,0,canvas_image_width,200,200,200,200);
```

Step 1 :利用 DMA 功能寫入一張圖片到 SDRAM



Step 2 : 開啟 PIP 功能 ,PIP window = 200x200 , PIP (x,y) = (800,0),PIP destination(x,y) = (200,200)



紅色方框為 PIP 來源，綠色方框為 PIP 顯示視窗

第六章 ： 文字

RA8889 可支援兩種字庫來源，分為內建字庫與外部字庫，內建字庫支援 ISO/IEC 8859-1/2/4/5 等字型。外部字庫搭配上海集通公司 (Genitop Inc) 的部分串列字型 ROM，可支援多國字庫或字型標準的顯示，如 ASIC, GB12345, GB18030 , GB2312 Special, BIG5, UNI-jpn, JIS0208, Latin, Greek, Cyrillic, Arabic, UNICODE, Hebrew, Thai, ISO-8859 以及 GB2312 Extension 等等

RA8889 的文字輸入可分為兩種來源:

1. 內建字 .
2. 外部字型 ROM .

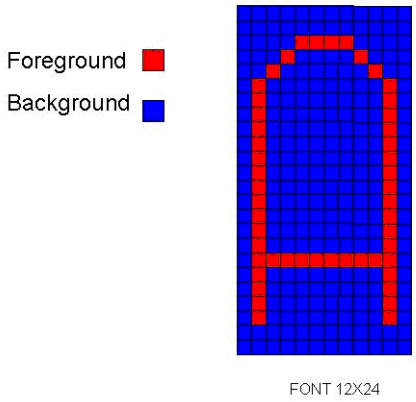


圖 6-1： 範例

文字有一些參數設定，在 REG[CCh]~REG[DEh]，當你需要改變任何的字型參數，你可以參考下面的流程圖。

字型的顏色設定在前景色與背景色的暫存器 REG[D2h]~[D7h]。

例如：我們寫 font1 與 font2 兩筆字串，各別有 64 筆字庫的資料。

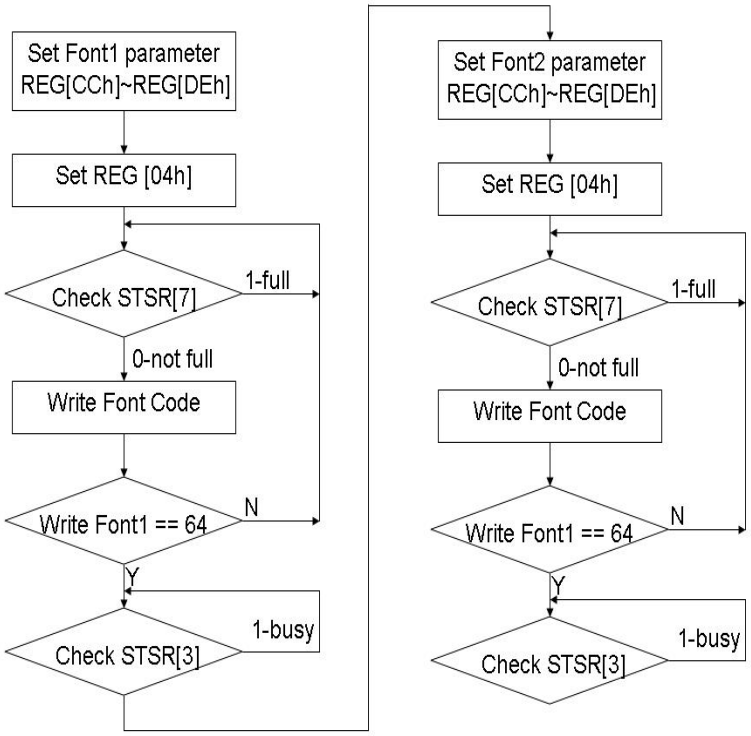


圖 6-2：文字顯示流程圖

內建字

RA8889 內建 12x24 點的 ASCII 字型 ROM，提供使用者更方便的方式，用特定編碼 (Code) 輸入文字。內建的字集支援 ISO/IEC 8859-1/2/4/5 編碼標準，此外，使用者可以透過 REG[D2h~D4h] 選擇文字前景顏色，以及透過 REG[D5h~D7h] 選擇背景顏色，文字寫入的程序請參考下圖:

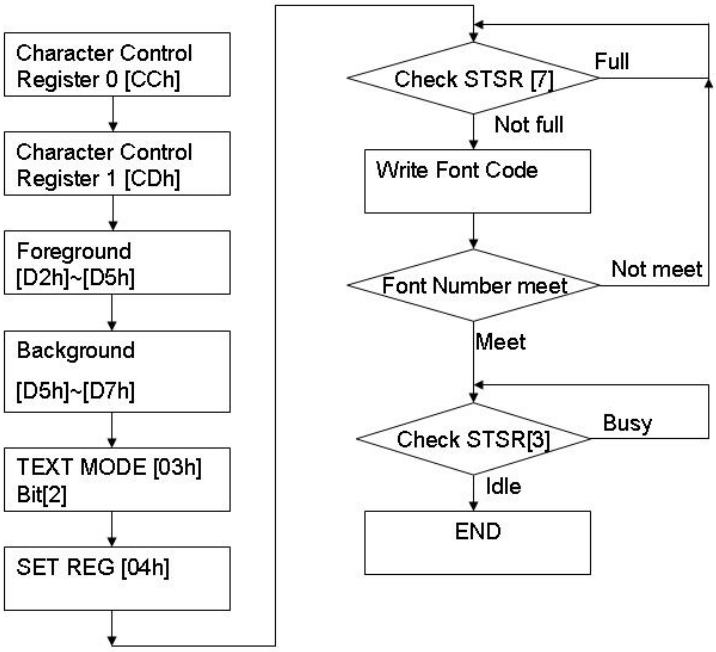


圖 **6-3：** 內部 **ASCII** 字庫使用流程

表 6-1 內含 ISO/IEC 8859-1 標準的字集。ISO 是國際標準化組織的簡稱，ISO/IEC 8859-1 又稱 ”Latin-1” 或「西歐語言」，是國際標準化組織內 ISO/IEC 8859 的第一個發展的 8 位元字集。以 ASCII 為基礎，包含了 0xA0 到 0xFF 的範圍內 192 個拉丁字母及符號。此字集編碼使用遍及西歐，包括阿爾巴尼亞語、巴斯克語、布列塔尼語、加泰羅尼亞語、丹麥語、荷蘭語、法羅語、弗裡斯語 (Frisian)、加利西亞語、德語、格陵蘭語、冰島語、愛爾蘭蓋爾語、義大利語、義大利語、拉丁語、盧森堡語、挪威語、葡萄牙語、里托羅曼斯語、蘇格蘭蓋爾語、西班牙語及瑞典語。

英語雖然沒有重音字母，但仍會標明為 ISO 8859-1 編碼，歐洲以外的部份語言，如南非荷蘭語、斯瓦希里 語、印尼語及馬來語、菲律賓他加洛語 (Tagalong) 也可使用 ISO8859-1 編碼。

表 6-1 : ASCII Block 1(ISO/IEC 8859-1)

L / H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		☺	☻	♥	♦	♣	♠	•	◘	○	◙	♂	♀	♪	♫	☼
1	►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	∟	↔	▲	▼
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
8	€		‚	ƒ	„	…	†	‡	ˆ	‰	Š	<	Œ		Ž	
9		‘	’	“	”	•	–	—	˜	™	š	>	œ		ž	Ÿ
A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	2	3	´	µ	¶	·	¸	1	0	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

表 6-2 內為 ISO/IEC 8859-2 的標準字集，又稱 Latin-2 或「中歐語言」，是國際標準化組織內 ISO/IEC 8859 的第二個 8 位字元集。此字元集主要支援以下文字：克羅埃西亞語、捷克語、匈牙利語、波蘭語、斯洛伐克語、斯洛維尼亞語、索布語。而阿爾巴尼亞語、英語、德語、拉丁語也可用此字元集顯示。芬蘭語中只有外來語才有 å 字元，若不考慮此字元，ISO/IEC8859-2 也可用於瑞士及芬蘭語。

表 6-2: ASCII Block 2 (ISO/IEC 8859-2)

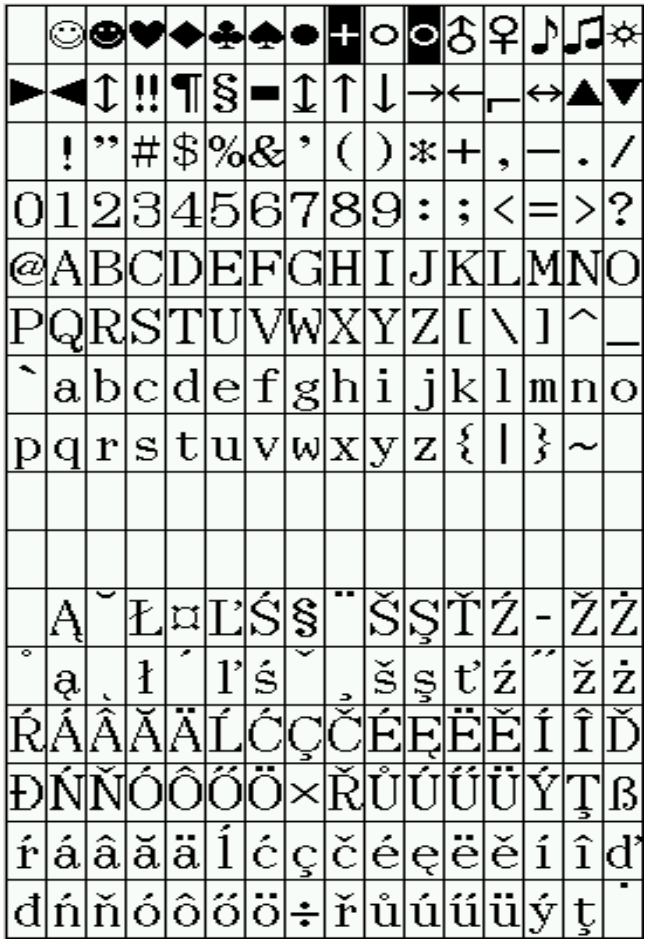


表 6-3 內為 ISO/IEC 8859-4 之標準字集，又稱 Latin-4 或「北歐語言」，是國際標準化組織內 ISO/IEC 8859 的第四個 8 位字元集，它設計來表示愛沙尼亞語、格陵蘭語、拉脫維雅語、立陶宛語及部分薩米語 (Sami) 文字，此字元集同時能支援以下文字：丹麥語、英語、芬蘭語、德語、拉丁語、挪威語、斯洛維尼亞語及瑞典語。

表 6-3: ASCII Block 3 (ISO/IEC 8859-4)

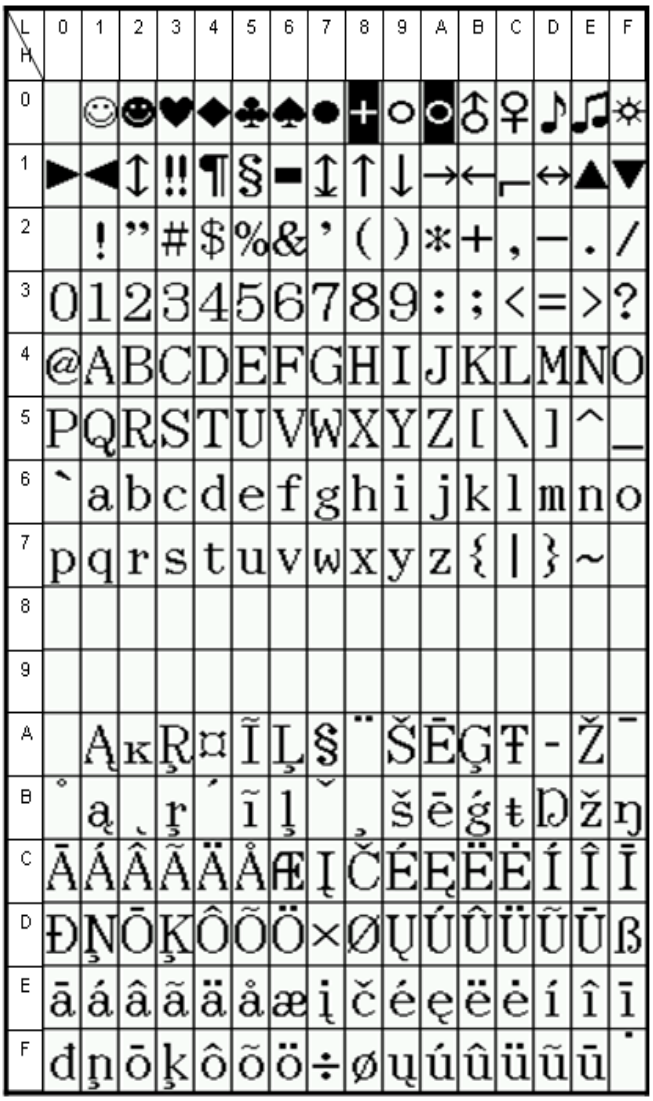


表 6-4 內為 ISO/IEC 8859-5 之標準字集，是國際標準化組織內 ISO/IEC 8859 的第五個 8 位字元集，它設計來表示保加利亞語、語俄、塞爾維亞語與馬其頓人語。

表 6-4 : ASCII Block 4 (ISO/IEC 8859-5)

外部字型 ROM

RA8889 的外部串列 ROM 介面針對不同應用提供了許多種字體。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援產品編號包含: GT21L16T1W、GT30L16U2W、GT30L24T3Y、GT30L24M1Z、GT30L32S4W、GT23L24F6Y、GT23L24S1W。根據不同的產品，它提供了不同的大小，其中包含 16x16、24x24、32x32 以及各種不同寬度的字體大小。'詳細的說明如下表列：

6.1.1:GT21L16T1W

- Reg[CEh][7:5]: 000b
- Character height: x16

Allowed character sets & width:

	GB12345 GB18030	BIG5	ASCII	UNI-jpn	JIS0208
Normal	V	V	V	V	V
Arial			V		
Roman			V		
Bold			V		

	Latin	Greek	Cyrillic	Arabic
Normal	V	V	V	
Arial	V	V	V	V
Roman				V
Bold				

*Arial & Roman is variable width.

6.1.2:GT30L16U2W

- Reg[CEh][7:5]: 001b
- Character height: x16

Allowed character sets & width:

	UNICODE	ASCII	Latin	Greek	Cyrillic	Arabic	GB2312 Special
Normal	V	V	V	V	V		V
Arial		V	V	V	V	V	
Roman		V				V	
Bold							

*Arial & Roman is variable width.

6.1.3:GT30L24T3Y

- Reg[CEh][7:5]: 010b
- Character height: x16

Allowed character sets & width:

	GB2312	GB12345/ GB18030	BIG5	UNICODE	ASCII
Normal	V	V	V	V	V
Arial					V
Roman					
Bold					

*Arial & Roman is variable width.

- Character height: x24

Allowed character sets & width:

	GB2312	GB12345/ GB18030	BIG5	UNICODE	ASCII
Normal	V	V	V	V	
Arial					V
Roman					
Bold					

*Arial & Roman is variable width.

6.1.4:GT30L24M1Z

- Reg[CEh][7:5]: 011b
- Character height: x24

Allowed character sets & width:

	GB2312 Extension	GB12345/ GB18030	ASCII
Normal	V	V	V
Arial			V
Roman			V
Bold			

*Arial & Roman is variable width.

6.1.5:GT30L32S4W

- Reg[CEh][7:5]: 100b
- Character height: x16

Allowed character sets & width:

	GB2312	GB2312 Extension	ASCII
Normal	V	V	V
Arial			V
Roman			V
Bold			

*Arial & Roman is variable width.

- Character height: x24

Allowed character sets & width:

	GB2312	GB2312 Extension	ASCII
Normal	V	V	V
Arial			V
Roman			V
Bold			

*Arial & Roman is variable width.

- Character height: x32

Allowed character sets & width:

	GB2312	GB2312 Extension	ASCII
Normal	V	V	V
Arial			V
Roman			V
Bold			

*Arial & Roman is variable width.

6.1.6:GT20L24F6Y

- Reg[CEh][7:5]: 101b
- Character height: x16

Allowed character sets & width:

	ASCII	Latin	Greek	Cyrillic
Normal	V	V	V	V
Arial	V	V	V	V
Roman	V			
Bold	V			

	Arabic	Hebrew	Thai	ISO-8859
Normal		V	V	V
Arial	V			
Roman				
Bold				

*Arial & Roman is variable width.

- Character height: x24

Allowed character sets & width:

	ASCII	Latin	Greek	Cyrillic	Arabic
Normal		V	V	V	
Arial	V				V
Roman					
Bold					

*Arial & Roman is variable width.

6.1.7:GT21L24S1W

- Reg[CEh][7:5]: 110b
- Character height: x24

Allowed character sets & width:

	GB2312	GB2312 Extension	ASCII
Normal	V	V	V
Arial			V
Roman			
Bold			

*Arial & Roman is variable width.

6.2:在 LCD 顯示字

6.2.1:內建字 ：

RA8889內建12x24點的ASCII字形ROM提供使用者更方便的方式用特定編碼 (Code) 輸入文字。內建字庫支援了ISO//IEC 8859-1/2/4/5的標準字集。

```
void Select_Font_Height_WxN_HxN_ChromaKey_Alignment
(
unsigned char Font_Height
/*Font_Height:
16 : Font = 8x16 、16x16
24 : Font = 12x24、24x24
32 : Font = 16x32、32x32*/
,unsigned char XxN // Font size Width it could be set from x 1 to x 4
,unsigned char YxN // Font size Height it could be set from x 1 to x 4
,unsigned char ChromaKey
/*ChromaKey :
0 : Font Background color with no transparency
1 : Set Font Background color with transparency*/
,unsigned char Alignment //0 : no alignment , 1 : Set font alignment
)

void Print_Internal_Font_String
(
unsigned short x //coordinate x for print string
,unsigned short y //coordinate x for print string
,unsigned short X_W //active window width
,unsigned short Y_H //active window height
,unsigned long FontColor //FontColor : Set Font Color
,unsigned long BackGroundColor
/*BackGroundColor : Set Font BackGround Color.Font Color and BackGround Color dataformat :
ColorDepth_8bpp : R3G3B2
ColorDepth_16bpp : R5G6B5
ColorDepth_24bpp : R8G8B8*/
,char tmp2[] //tmp2 : Font String which you want print on LCD
)
```

範例 ：

```
/*Font_Height = 24 : Font   = 12x24、24x24  、XxN = 4 :Font Width x4、YxN = 4 :Font Height x 4、
ChromaKey = 0 : Font Background color with no transparency、
Alignment = 0 : no alignment
x : coordinate x for print string = 0
y : coordinate y for print string = 0
X_W : active window width = 1000
Y_H : active window height = 1000
FontColor : Set Font Color = 0xe0(8bpp)、0xf800(16bpp)、0xff0000(24bpp)
BackGroundColor : Set Font BackGround Color = 0x1c(8bpp)、0x07e0(16bpp)、0x00ff00(24bpp)*/

Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,4,4,0,0);
+
//When Color Depth = 8bpp
Print_Internal_Font_String(0,0,1000, 1000,0xe0,0x1c, "adfsdfdgfhhgfh");
Or
//When Color Depth = 16bpp
Print_Internal_Font_String(0,0, 1000, 1000,0xf800,0x07e0, "adfsdfdgfhhgfh");
or
//When Color Depth = 24bpp
Print_Internal_Font_String(0,0, 1000, 1000,0xff0000,0x00ff00, "adfsdfdgfhhgfh");
```

實際的顯示畫面 ：

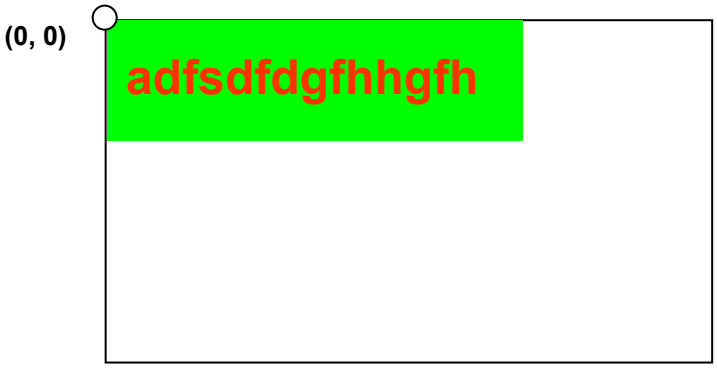


圖 6-4 內建字範例

6.2.2:透過外部字庫寫入 Big5 字串

RA8889 的外部串列 ROM 介面針對不同應用提供了許多種字體。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援產品編號包含: GT21L16T1W、GT30L16U2W、GT30L24T3Y、GT30L24M1Z、 GT30L32S4W、GT23L24F6Y、GT23L24S1W。根據不同的產品，它提供了不同的大小，其中包含 16x16、24x24、32x32 以及各種不同寬度的字體大小。

這個應用程式介面是使用了 RA8889 外部串列字庫來列印出 Big5 的字串。

```
void Select_Font_Height_WxN_HxN_ChromaKey_Alignment
(
unsigned char Font_Height
/*Font_Height:
16 : Font = 8x16 、16x16
24 : Font = 12x24、24x24
32 : Font = 16x32、32x32*/
,unsigned char XxN // Font size Width it could be set from x 1 to x 4
,unsigned char YxN // Font size Height it could be set from x 1 to x 4
,unsigned char ChromaKey
/*ChromaKey :
0 : Font Background color with no transparency
1 : Set Font Background color with transparency*/
,unsigned char Alignment //0 : no alignment , 1 : Set font alignment
)

void Print_BIG5String
(
unsigned char Clk //SPI CLK = System Clock / 2*(Clk+1)
,unsigned char BUS// 0 : Bus0, 1:Bus1
,unsigned char SCS //0 : use CS0 , 1 : use CS1 , 2 : use CS2, 3 :use CS3
,unsigned short x //coordinate x for print string
,unsigned short y //coordinate y for print string
,unsigned short X_W //active window width
,unsigned short Y_H //active window height
,unsigned long FontColor //Set Font Color
,unsigned long BackGroundColor //Set Font BackGround Color
/*Font Color and BackGround Color dataformat :
ColorDepth_8bpp : R3G3B2
ColorDepth_16bpp : R5G6B5
```

```
ColorDepth_24bpp : R8G8B8*/
,char *tmp2 //tmp2 : BIG5 Font String which you want print on LCD
)
```

範例:

```
/*Font_Height = 24 : Font   = 12x24、24x24  、XxN = 4 :Font Width x4、YxN = 4 :Font Height x 4、
ChromaKey = 0 : Font Background color with no transparency、
Alignment = 0 : no alignment
x : coordinate x for print string = 0
y : coordinate y for print string = 0
X_W : active window width = 800
Y_H : active window height = 480
FontColor : Set Font Color = 0xe0(8bpp)、0xf800(16bpp)、0xff0000(24bpp)
BackGroundColor : Set Font BackGround Color = 0x1c(8bpp)、0x07e0(16bpp)、0x00ff00(24bpp)
Print 瑞佑科技 123456*/
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,4,4,0,1);
+
Print_BIG5String(3,0,1,0,0,800 ,480 ,0xe0,0x1c,"瑞佑科技 123456"); When Color Depth = 8bpp
or
Print_BIG5String(3,0,1,0,0,800 ,480 ,0xf800,0x07e0,"瑞佑科技 123456"); When Color Depth = 16bpp
or
Print_BIG5String(3,0,1,0,0,800 ,480 ,0xff0000,0x00ff00,"瑞佑科技 123456"); When Color Depth = 24bpp
```

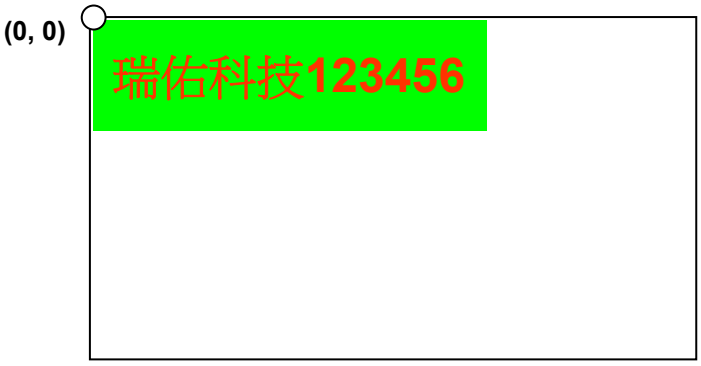


圖 6-5 透過外部字庫寫入 Big5 字串

6.2.3:透過外部字庫寫入 GB2312 字串:

RA8889 的外部串列 ROM 介面針對不同應用提供了許多種字體。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援產品編號包含: GT21L16T1W、GT30L16U2W、GT30L24T3Y、GT30L24M1Z、 GT30L32S4W、GT23L24F6Y、GT23L24S1W。根據不同的產品，它提供了不同的大小，其中包含 16x16、24x24、32x32 以及各種不同寬度的字體大小。

這個應用程式介面是使用了 RA8889 外部串列字庫來列印出 GB2312 的字串。

```
void Select_Font_Height_WxN_HxN_ChromaKey_Alignment
(
unsigned char Font_Height
/*Font_Height:
16 : Font = 8x16  、16x16
24 : Font = 12x24、24x24
32 : Font = 16x32、32x32*/
,unsigned char XxN // Font size Width it could be set from x 1 to x 4
,unsigned char YxN // Font size Height it could be set from x 1 to x 4
,unsigned char ChromaKey
/*ChromaKey :
0 : Font Background color with no transparency
1 : Set Font Background color with transparency*/
,unsigned char Alignment // 0 : no alignment, 1 : Set font alignment
)

void Print_GB2312String
(
unsigned char Clk //Clk : SPI CLK = System Clock / 2*(Clk+1)
,unsigned char BUS// 0 : Bus0, 1:Bus1
,unsigned char SCS //0 : use CS0 , 1 : use CS1 , 2 : use CS2, 3 :use CS3
,unsigned short x //coordinate x for print string
,unsigned short y //coordinate y for print string
,unsigned short X_W //active window width
,unsigned short Y_H //active window height
,unsigned long FontColor //Set Font Color
,unsigned long BackGroundColor //Set Font BackGround Color
/*Font Color and BackGround Color dataformat :
ColorDepth_8bpp : R3G3B2
ColorDepth_16bpp : R5G6B5
```

```
ColorDepth_24bpp : R8G8B8*/
,char tmp2[] //tmp2 : GB2312 Font String which you want print on LCD
)
```

範例 ：

```
/*Font_Height = 24 : Font   = 12x24、24x24  、XxN = 4 :Font Width x4、YxN = 4 :Font Height x 4、
ChromaKey = 0 : Font Background color with no transparency、
Alignment = 0 : no alignment
x : coordinate x for print string = 0
y : coordinate y for print string = 0
X_W : active window width = 800
Y_H : active window height = 480
FontColor : Set Font Color = 0xe0(8bpp)、0xf800(16bpp)、0xff0000(24bpp)
BackGroundColor : Set Font BackGround Color = 0x1c(8bpp)、0x07e0(16bpp)、0x00ff00(24bpp)
Print 瑞佑科技 123456*/
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,4,4,0,1);
+
Print_GB2312String(3,0,1,0,0, 800, 480,0xe0,0x03, "瑞佑科技 123456"); //When Color Depth = 8bpp
or
Print_GB2312String(3,0,1,0,0, 800, 480,0xf800,0x001f, "瑞佑科技 123456"); //When Color Depth = 16bpp
or
Print_GB2312String(3,0,1,0,0, 800, 480,0xff0000,0x0000ff, "瑞佑科技 123456"); //When Color Depth =
24bpp
```

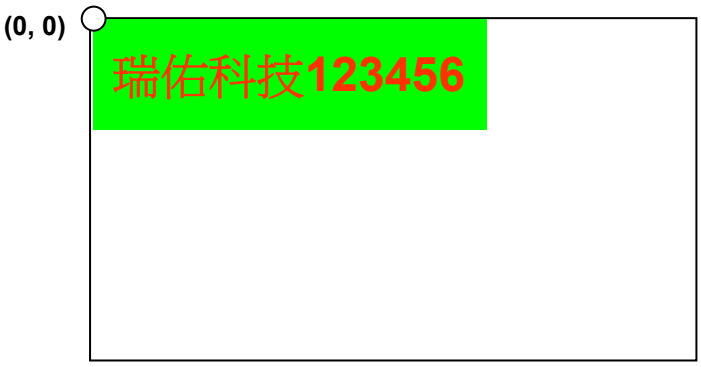


圖 6-6: 透過外部字庫寫入 GB2312 字串

6.3.4: 透過外部字庫寫入 GB12345 字串

RA8889 的外部串列 ROM 介面針對不同應用提供了許多種字體。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援產品編號包含: GT21L16T1W、GT30L16U2W、GT30L24T3Y、GT30L24M1Z、 GT30L32S4W、GT23L24F6Y、GT23L24S1W。根據不同的產品，它提供了不同的大小，其中包含 16x16、24x24、32x32 以及各種不同寬度的字體大小。

這個應用程式介面是使用了 RA8889 外部串列字庫來列印出 GB12345 的字串。

```
void Select_Font_Height_WxN_HxN_ChromaKey_Alignment
(
unsigned char Font_Height
/*Font_Height:
16 : Font = 8x16  、16x16
24 : Font = 12x24、24x24
32 : Font = 16x32、32x32*/
,unsigned char XxN // Font size Width it could be set from x 1 to x 4
,unsigned char YxN // Font size Height it could be set from x 1 to x 4
,unsigned char ChromaKey
/*ChromaKey :
0 : Font Background color with no transparency
1 : Set Font Background color with transparency*/
,unsigned char Alignment // 0 : no alignment, 1 : Set font alignment
)

void Print_GB12345String
(
unsigned char Clk //Clk : SPI CLK = System Clock / 2*(Clk+1)
,unsigned char BUS// 0 : Bus0, 1:Bus1
,unsigned char SCS //0 : use CS0 , 1 : use CS1 , 2 : use CS2, 3 :use CS3
,unsigned short x //coordinate x for print string
,unsigned short y ////coordinate y for print string
,unsigned short X_W //active window width
,unsigned short Y_H //active window height
,unsigned long FontColor //Set Font Color
,unsigned long BackGroundColor //Set Font BackGround Color
/*Font Color and BackGround Color dataformat :
ColorDepth_8bpp : R3G3B2
ColorDepth_16bpp : R5G6B5
```

```
ColorDepth_24bpp : R8G8B8*/
,char *tmp2 //tmp2 : GB12345 Font String which you want print on LCD
)
```

範例 ：

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,4,4,0,1);
+
Print_GB12345String(3,0,1,0,0, 800, 480,0xe0,0x03,"瑞佑科技123456"); When Color Depth = 8bpp
or
Print_GB12345String(3,0,1,0,0, 800, 480,0xf800,0x001f, "瑞佑科技  123456"); When Color Depth = 16bpp
or
Print_GB12345String(3,0,1,0,0, 800, 480,0xff0000,0x0000ff, "瑞佑科技 123456"); When Color Depth = 24bpp
```

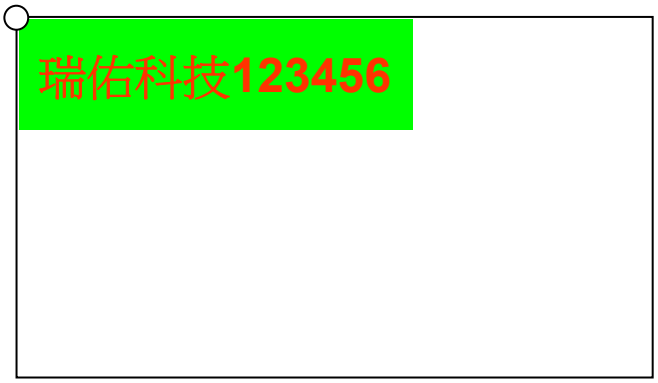


圖 6-7: 透過外部字庫寫入 GB12345 字串

6.3.5: 透過外部字庫寫入 Unicode 字串

RA8889 的外部串列 ROM 介面針對不同應用提供了許多種字體。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援產品編號包含: GT21L16T1W、GT30L16U2W、GT30L24T3Y、GT30L24M1Z、 GT30L32S4W、GT23L24F6Y、GT23L24S1W。根據不同的產品，它提供了不同的大小，其中包含 16x16、24x24、32x32 以及各種不同寬度的字體大小。

這個應用程式介面是使用了 RA8889 外部串列字庫來列印出 Unicode 的字串。

```
void Select_Font_Height_WxN_HxN_ChromaKey_Alignment
(
unsigned char Font_Height
/*Font_Height:
16 : Font = 8x16  、16x16
24 : Font = 12x24、24x24
32 : Font = 16x32、32x32 */
,unsigned char XxN //XxN :Font Width x 1~4
,unsigned char YxN //YxN :Font Height x 1~4
,unsigned char ChromaKey
/*ChromaKey :
0 : Font Background color not transparency
1 : Set Font Background color transparency*/
,unsigned char Alignment // 0 : no alignment, 1 : Set font alignment
)

void Print_UnicodeString
(
unsigned char Clk //SPI CLK = System Clock / 2*(Clk+1)
,unsigned char BUS// 0 : Bus0, 1:Bus1
,unsigned char SCS //0 : use CS0 , 1 : use CS1 , 2 : use CS2, 3 :use CS3
,unsigned short x //Print font start coordinate of X
,unsigned short y //Print font start coordinate of Y
,unsigned short X_W //active window width
,unsigned short Y_H //active window height
,unsigned long FontColor //Set Font Color
,unsigned long BackGroundColor //Set Font BackGround Color
/*Font Color and BackGround Color dataformat :
ColorDepth_8bpp : R3G3B2
ColorDepth_16bpp : R5G6B5
```

```
ColorDepth_24bpp : R8G8B8*/
,unsigned short *tmp2 /*tmp2 : Unicode Font String which you want print on LCD (L"string" in keil c is Unicode string)*/
)
```

範例 ：

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,4,4,0,1);
+
//When Color Depth = 8bpp
Print_UnicodeString(3,0,1,0,0,800 ,480 ,0xe0,0x1c,L"新竹縣竹北市台元一街8號6樓之5");
Or
//When Color Depth = 16bpp
Print_UnicodeString(3,0,1,0,0,800 ,480 ,0xf800,0x07e0,L"新竹縣竹北市台元一街8號6樓之5");
Or
//When Color Depth = 24bpp
Print_UnicodeString(3,0,1,0,0,800 ,480 ,0xff0000,0x00ff00,L"新竹縣竹北市台元一街8號6樓之5");
```

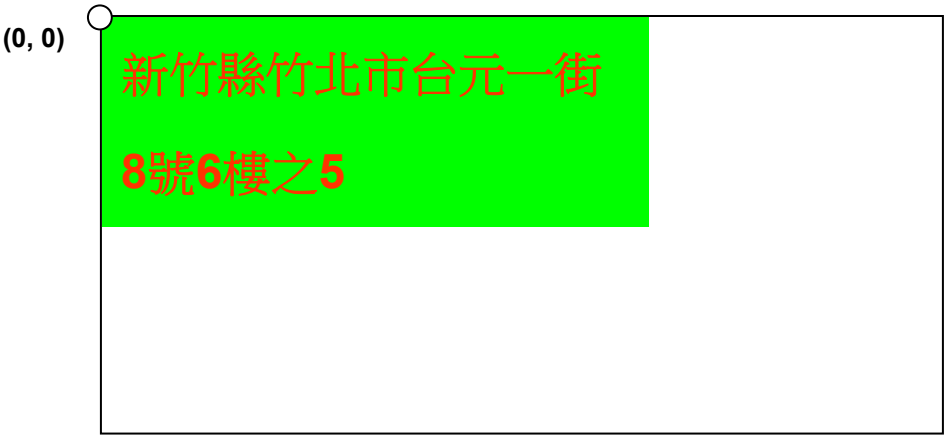


圖 6-8: 透過外部字庫寫入 Unicode 字串

第七章 ：PWM (Pulse Width Modulation)

RA8889有2 組 16定時器。位定時器0和1 具 有 ）脈衝寬度調製PWM定時器。功能（0。其用於以大電流的設備，具有一個死區生成器 定時器0和1有一個共用的8位元的預分頻器(prescalar)每個定時器。還有一個時脈除從而產生，頻器4個不同的除 ）頻信號1，1/2，1/4和1/8。（ 最後透過timer count buffer (TCNTBn)用來調整PWM輸出的週期時間以及利用timer compare buffer (TCMPBn)）的值來進行脈衝寬度調製PWM（，產生一個穩定的脈波。詳細公式與圖解如下列說明:

PWM CLK = (Core CLK / Prescalar) /2$^{\text{clock divided}}$

PWM output period = (Count Buffer + 1) x PWM CLK time

PWM output high level time = (Compare Buffer + 1) x PWM CLK time

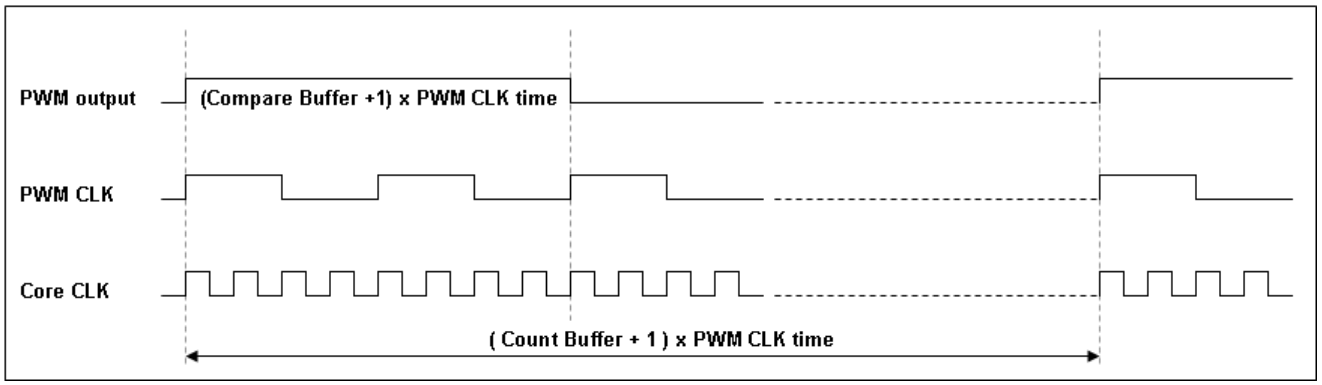


圖 7-1 : PWM 時序圖

PWM 功能 API:

```
void PWM0
(
unsigned char on_off //on_off = 1 ,enable PWM, on_off = 0 , disable PWM.
, unsigned char Clock_Divided // divided PWM clock, only 0~3(1,1/2,1/4,1/8)
, unsigned char Prescalar //Prescaler : only 1~256
, unsigned short Count_Buffer //Count_Buffer : set PWM output period time
,unsigned short Compare_Buffer //Compare_Buffer : set PWM output high level time(Duty cycle)
/*Such as the following formula :
PWM CLK = (Core CLK / Prescalar ) /2^ divided clock
PWM output period = (Count Buffer + 1) x PWM CLK time
PWM output high level time = (Compare Buffer + 1) x PWM CLK time */
)

void PWM1
(
unsigned char on_off //on_off = 1 ,enable PWM, on_off = 0 , disable PWM.
, unsigned char Clock_Divided // divided PWM clock, only 0~3(1,1/2,1/4,1/8)
, unsigned char Prescalar //Prescaler : only 1~256
, unsigned short Count_Buffer //Count_Buffer : set PWM output period time
,unsigned short Compare_Buffer //Compare_Buffer : set PWM output high level time(Duty cycle)
/*Such as the following formula :
PWM CLK = (Core CLK / Prescalar ) /2^ divided clock
PWM output period = (Count Buffer + 1) x PWM CLK time
PWM output high level time = (Compare Buffer + 1) x PWM CLK time */
)
```

範例:

當 Core CLK = 60MHz

```
PWM0(1,3,100,1,0);
PWM1(1,3,100,4,3);
```

PWM0 輸出:

/*On_off = 1, PWM Enable.

Clock Divided = 3, Prescalar = 100, Count_Buffer = 1, Compare_Buffer = 0.

PWM CLK = (Core CLK / Prescaler) /$2^{clock\ divided}$ = (60M / 100) / 2^3 = 75KHz

PWM output period = (Count Buffer + 1) x PWM CLK time = (1+1) x (1 / 75K) ≒ 26.67us

PWM output high level time = (Compare Buffer + 1) x PWM CLK time = (0+1) x (1 / 75K) ≒ 13.33us*/

示波器波形:

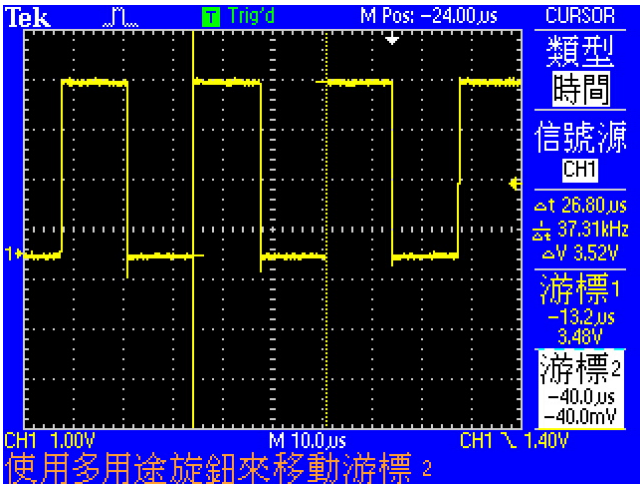


圖 7-2：PWM0 輸出波形週期

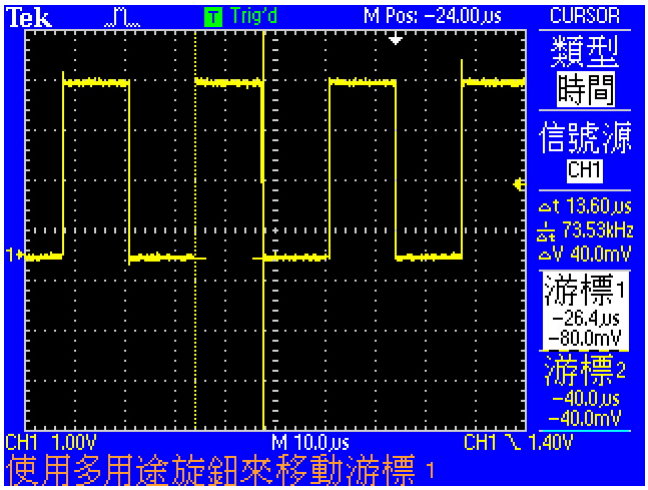


圖 7-3：PWM0 輸出高電位的時間

PWM1 輸出:

/*On_off = 1, PWM Enable.

Clock Divided = 3, Prescalar = 100, Count_Buffer = 4, Compare_Buffer = 3.

PWM CLK = (Core CLK / Prescaler) /$2^{clock\ divided}$ = (60M / 100) / 2^3 = 75KHz

PWM output period = (Count Buffer + 1) x PWM CLK time = (4+1) x (1 / 75K) ≒ 66.67us

PWM output high level time = (Compare Buffer + 1) x PWM CLK time = (3+1) x (1 / 75K) ≒ 53.33us*/

示波器波形:

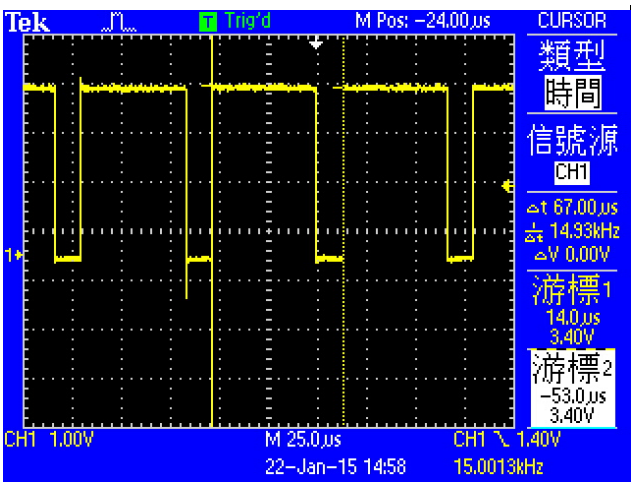


圖 7-4：PWM1 輸出波形週期

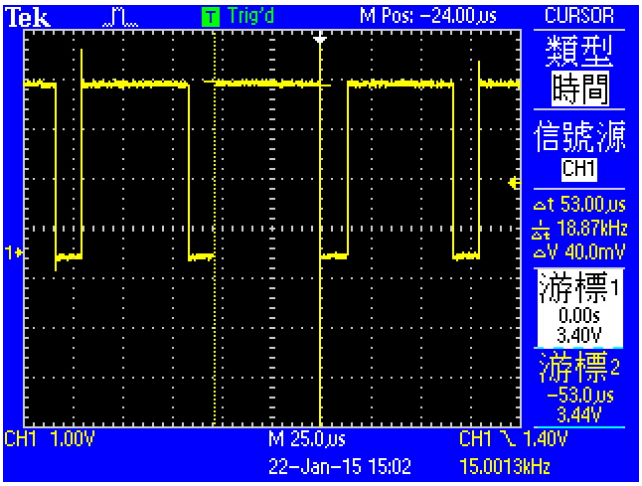


圖 7-5：PWM1 輸出高電位的時間

第八章 ： 鍵盤掃描

鍵盤掃描會掃描並讀取鍵盤狀態，而鍵盤矩陣會由硬體來切換掃描線。這個功能可以提供鍵盤應用，圖 8-1 顯示的是基本的鍵盤應用電路。RA8889 為因應外部電路需求，已經在 KIN[4:0] 內建提升電阻。

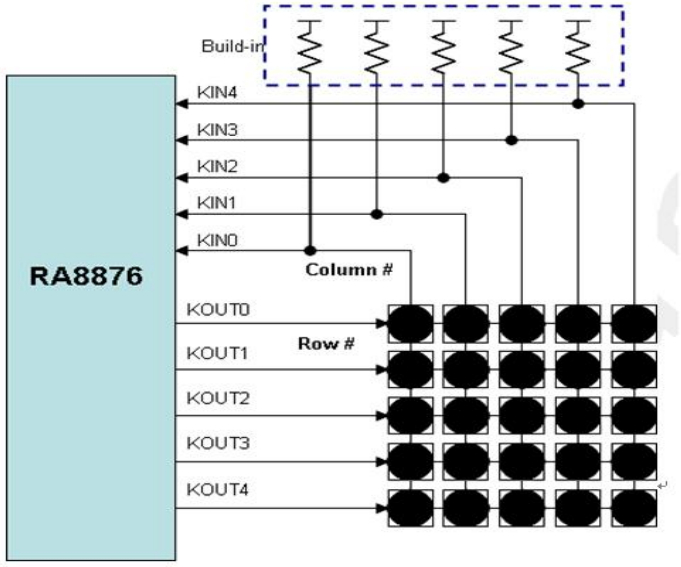


圖 8-1 : Key-Pad Application

8.1 鍵盤掃描操作方式

RA8889 鍵盤掃描控制器的特點如下:

1. 最多支援 5x5 鍵盤矩陣
2. Key-Scan 具有可程式化的掃描頻率與取樣時間。
3. 可調整的長按鍵時間
4. 支持多鍵同時按
   注意: 可同時按 2 個按鍵或是要按 3 個按鍵 (但是 3 按鍵組成不能是 90°排列)
5. 可使用按鍵來喚醒系統

KSCR 是鍵盤掃描的狀態暫存器，這個暫存器被使用在了解鍵盤掃描的操作狀態，如取樣時間、取樣頻率、致能長按鍵。而若是有按鍵按下，使用者也可以透過中斷得知。在 KSCR2 bit1~0 紀錄目前按下的按鍵數目。然後使用者可以透過讀取 KSDR 得到按鍵碼。

註：“Normal key” 是在以取樣時間為基礎上有被認知為合格的按下按鍵行為。“Long Key” 則是在長按鍵取樣週期下有被認知為合格的按下按鍵行為。先產生 "Normal Key" 才會產生 “Long Key”，有時在某些應用上需要分開使用。

表8-1 是在 ”Nnomal Key” 下鍵碼與鍵盤矩陣的對應，按下的按鍵鍵碼會被存在 KSDR0~2。如果是長時間按下按鍵，則表現出的會是 ”Long Key”，而相關鍵碼在表8-2。

表8-1: Key Code Mapping 表 (Normal Key)

	Kin0	Kin1	Kin2	Kin3	Kin4
Kout0	00h	01h	02h	03h	04h
Kout1	10h	11h	12h	13h	14h
Kout2	20h	21h	22h	23h	24h
Kout3	30h	31h	32h	33h	34h
Kout4	40h	41h	42h	43h	44h

表8-2: Key Code Mapping 表 (Long Key)

	Kin0	Kin1	Kin2	Kin3	Kin4
Kout0	80h	81h	82h	83h	84h
Kout1	90h	91h	92h	93h	94h
Kout2	A0h	A1h	A2h	A3h	A4h
Kout3	B0h	B1h	B2h	B3h	B4h
Kout4	C0h	C1h	C2h	C3h	C4h

當按下多鍵時，最多有三個按鍵會被存在 KSDR0，KSDR1 與 KSDR2 三個暫存器中。注意鍵碼儲存的方式與按鍵位置有關或者說與鍵碼有關，而與按鍵順序無關，請參下列例子:

在相同時間按下鍵碼 0x34, 0x00 and 0x22，在 KSDR0~2 儲存方式如下:

KSDR0 = 0x00
KSDR1 = 0x22
KSDR2 = 0x34

以上所提的鍵盤掃描設定介紹如下:

表 8-3 : Key-Scan Relative Registers

Reg.	Bit_Num	Description	Reference
KSCR1	Bit 6	Long Key Enable bit	REG[FBh]
	Bit [5:4]	Key-Scan sampling times setting	
	Bit [2:0]	Key-Scan scan frequency setting	
KSCR2	Bit [7]	Key-Scan Wakeup Function Enable Bit	REG[FCh]
	Bit [3:2]	long key timing adjustment	
	Bit [1:0]	The number of key hit	
KSDR0 KSDR1 KSDR2	Bit [7:0]	Key code for pressed key	REG[FDh ~ FFh]
CCR	Bit 5	Key-Scan enable bit	REG[01h]
INTR	Bit 4	Key-Scan interrupt enable	REG[0Bh]
INTC2	Bit 4	Key-Scan Interrupt Status bit	REG[0Ch]

致能鍵盤掃描功能(Key-Scan)，使用者可以使用下列方法檢查按鍵狀態:

1) Software check method: 檢查 Key-scan 的狀態值(status)，來得知是否被按下。

2) Hardware check method: 由中斷來得知是否有按鍵被按下。

若是設定中斷致能 (INTEN bit[3]) 為 1，那麼有鍵盤有被按下時就會產生中斷。而當中斷產生時，Key-scan 中斷狀態旗標 (bit[3] of INTF) 將永遠為 1，無論使用何種方法，使用者在讀取鍵碼後必須清除中斷狀態旗標，否則以後就不會再產生中斷。

此外，RA8889 在省電模式下支援“Key-stroke wakeup ”，經由設定完成後，任何按鍵觸發都可以將 RA8889 由睡眠模式中喚醒。為了檢知喚醒事件，MPU 可以透過軟體程式去輪詢 RA8889 的中斷是否產生。

以上應用的暫存器程式設定流程圖如下:

1. 軟體方法

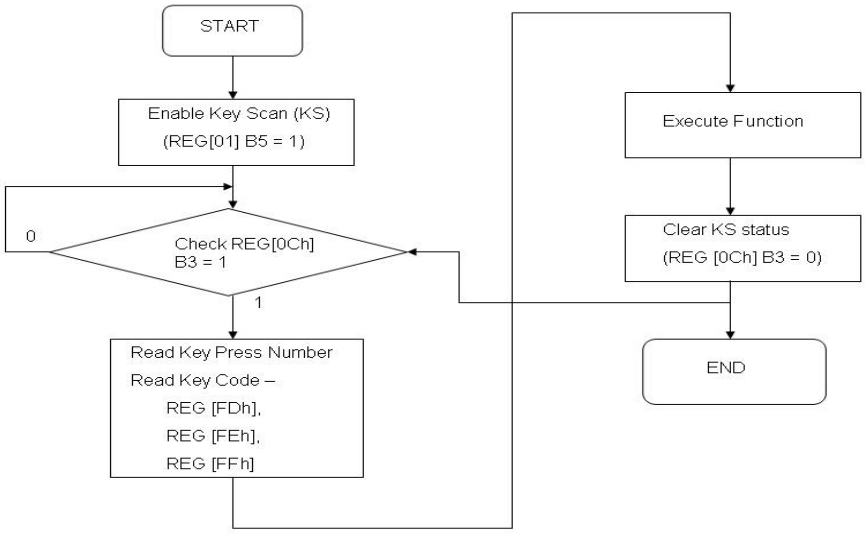


圖8-2 : Key-Scan Flowchart for Software Polling

2. 硬體方法

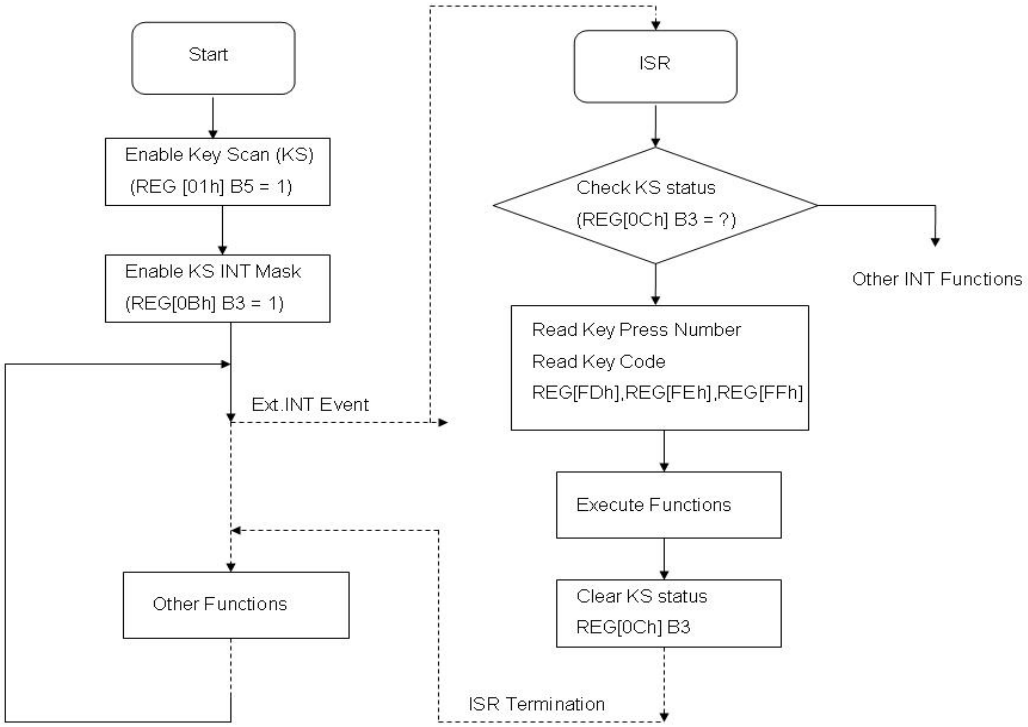


圖8-3 : Key-Scan for Hardware Interrupt

KEY SCAN 功能 API:

void API_Print_key_code(unsigned short x,unsigned short y,unsigned long FontColor,unsigned long BackGroundColor)

執行

```
API_Print_key_code(0,0,0x00,0xff);          //8bbp color depth
```

Or

```
API_Print_key_code(0,0,0x0000,0xffff);      //16bbp color depth
```

Or

```
API_Print_key_code(0,0,0x000000,0xffffff);  //24bbp color depth
```

條件:

x:顯示 Key Code 的 X 座標 = 0

y:顯示 Key Code 的 Y 座標 = 0

FontColor:顯示 Keycode 的顏色 = 黑色

BackGroundColor:顯示 Keycode 的背景色 = 白色

Example:

如按下下列三個按鍵

	Kin0	Kin1	Kin2	Kin3	Kin4
Kout0	00h	01h	02h	03h	04h
Kout1	10h	11h	12h	13h	14h
Kout2	20h	21h	22h	23h	24h
Kout3	30h	31h	32h	33h	34h
Kout4	40h	41h	42h	43h	44h

LCD 顯示畫面:

001123

附錄 1：PLL initialization

$$xCLK = \frac{\left(Fin / 2^{(xPLLDIVM)}\right) \times (xPLLDIVN + 1)}{2^{xPLLDIVK}}$$

The input OSC frequency (F_{IN}) must greater & PLLDIVM has following restriction:

$$10MHz \le Fin \le 15MHz \quad \& \quad 10MHz \le \frac{Fin}{2^{PLLDIVM}} \le 40MHz$$

The internal multiplied clock frequency $Fvco = \frac{Fin}{2^{PLLDIVM}} \times (PLLDIVN + 1)$ must be equal to or greater than 250 MHz and small than 500 MHz. i.e, $250MHz \le Fvco \le 500MHz$

```
void RA8889_PLL(unsigned short DRAM_clock, unsigned short CORE_clock, unsigned short SCAN_clock)
{
/*
[A]
DRAM_clock maximum 166 MHz
CORE_clock maximum 133 MHz (without internal font function)
SCAN_clock maximum 100 MHz

[B]
(1) 10MHz <= OSC_FREQ <= 15MHz
(2) 10MHz <= (OSC_FREQ/2^PLLDIVM) <= 40MHz
(3) 250MHz <= [OSC_FREQ/(2^PLLDIVM)]x(PLLDIVN+1) <= 500MHz
(4) In addition, please pay special attention to:
      [DRAM_clock] >= [CORE_clock],
      [CORE_clock] >= 2x[SCAN_clock].
PLLDIVM:0
PLLDIVN:1~63
PLLDIVK:CPLL & MPLL = 1/2/4/8.SPLL = 1/2/4/8/16/32/64/128.
ex:
 OSC_FREQ = 10MHz
 Set X_DIVK=2
 Set X_DIVM=0
 => (X_DIVN+1)=(XPLLx4)/10
*/



        // Set DRAM clock
```

```
if((DRAM_clock>=125)&&(DRAM_clock<=166))
{
	LCD_RegisterWrite(0x07,0x02);			//PLL Divided by 2
	LCD_RegisterWrite(0x08,(DRAM_clock*2/OSC_FREQ)-1);
}
else if((DRAM_clock>=63)&&(DRAM_clock<=124))
{
	LCD_RegisterWrite(0x07,0x04);			//PLL Divided by 4
	LCD_RegisterWrite(0x08,(DRAM_clock*4/OSC_FREQ)-1);
}
else if((DRAM_clock>=32)&&(DRAM_clock<=62))
{
	LCD_RegisterWrite(0x07,0x06);			//PLL Divided by 8
	LCD_RegisterWrite(0x08,(DRAM_clock*8/OSC_FREQ)-1);
}
else //if(DRAM_clock<32)
{
	LCD_RegisterWrite(0x07,0x06);			//PLL Divided by 8
	LCD_RegisterWrite(0x08,(32*8/OSC_FREQ)-1); //
}


// Set Core clock
if((CORE_clock>=125)&&(CORE_clock<=133))
{
	LCD_RegisterWrite(0x09,0x02);			//PLL Divided by 2
	LCD_RegisterWrite(0x0A,(CORE_clock*2/OSC_FREQ)-1);
}
else if((CORE_clock>=63)&&(CORE_clock<=124))
{
	LCD_RegisterWrite(0x09,0x04);			//PLL Divided by 4
	LCD_RegisterWrite(0x0A,(CORE_clock*4/OSC_FREQ)-1);
}
else if((CORE_clock>=32)&&(CORE_clock<=62))
{
	LCD_RegisterWrite(0x09,0x06);			//PLL Divided by 8
	LCD_RegisterWrite(0x0A,(CORE_clock*8/OSC_FREQ)-1);
}
```

```
else //if(CORE_clock<32)
{
	LCD_RegisterWrite(0x09,0x06);			//PLL Divided by 8
	LCD_RegisterWrite(0x0A,(32*8/OSC_FREQ)-1);//
}


// Set pixel clock
if((SCAN_clock>=63)&&(SCAN_clock<=100))
{
	LCD_RegisterWrite(0x05,0x04);			//PLL Divided by 4
	LCD_RegisterWrite(0x06,(SCAN_clock*4/OSC_FREQ)-1);
}
else if((SCAN_clock>=32)&&(SCAN_clock<=62))
{
	LCD_RegisterWrite(0x05,0x06);			//PLL Divided by 8
	LCD_RegisterWrite(0x06,(SCAN_clock*8/OSC_FREQ)-1);
}
else if((SCAN_clock>=16)&&(SCAN_clock<=31))
{
	LCD_RegisterWrite(0x05,0x16);			//PLL Divided by 16
	LCD_RegisterWrite(0x06,(SCAN_clock*16/OSC_FREQ)-1);
}
else if((SCAN_clock>=8)&&(SCAN_clock<=15))
{
	LCD_RegisterWrite(0x05,0x26);			//PLL Divided by 32
	LCD_RegisterWrite(0x06,(SCAN_clock*32/OSC_FREQ)-1);
}
else if((SCAN_clock>0)&&(SCAN_clock<=7))
{
	LCD_RegisterWrite(0x05,0x36);			//PLL Divided by 64
	LCD_RegisterWrite(0x06,(SCAN_clock*64/OSC_FREQ)-1);
}
else // if out of range, set 32MHz for debug.
{
	LCD_RegisterWrite(0x05,0x06);
	LCD_RegisterWrite(0x06,(32*8/OSC_FREQ)-1);
}
```

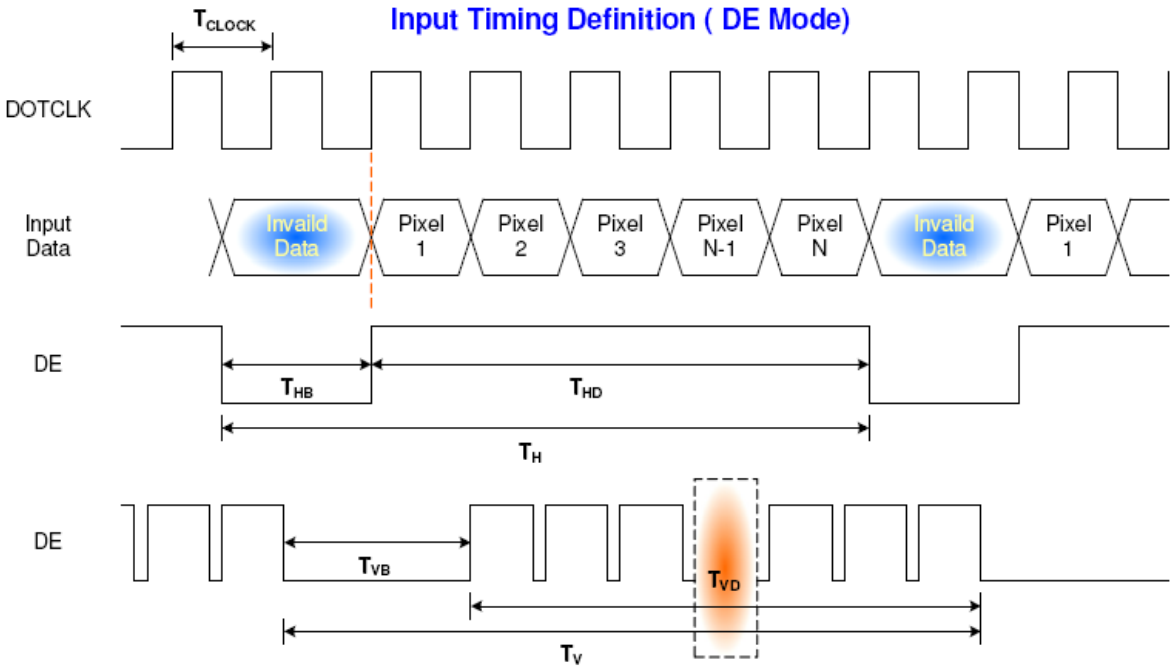
```
        Enable_PLL();
        delay_ms(100);

}
```

附錄 2 LCD initialization:

B116XW03 (Resolution 1366X768):

Parameter		Symbol	Min.	Typ.	Max.	Unit
Frame Rate		-		60	-	Hz
Clock frequency		1/ T_{Clock}	65	69.3	80	MHz
Vertical Section	Period	T_V	776	793	1000	T_{Line}
	Active	T_{VD}		768		
	Blanking	T_{VB}	8	25	180	
Horizontal Section	Period	T_H	1396	1456	2000	T_{Clock}
	Active	T_{HD}		1366		
	Blanking	T_{HB}	30	90	634	

B116XW03 Timing



Clock Frequency = SCLK = 70MHz

T_{HD} = Horizontal display width = 1366

T_{HB} = Horizontal non-display period(Back porch) + HSYNC Start Position(Front porch) =34 + 56 = 90

T_{VD} = Vertical Display Height(Line) = 768

T_{VB} = Vertical Non-Display Period(Back porch) + VSYNC Start Position(Front porch) = 13 + 12 =25

```
void Initial_AUO1366x768(void)
{
```

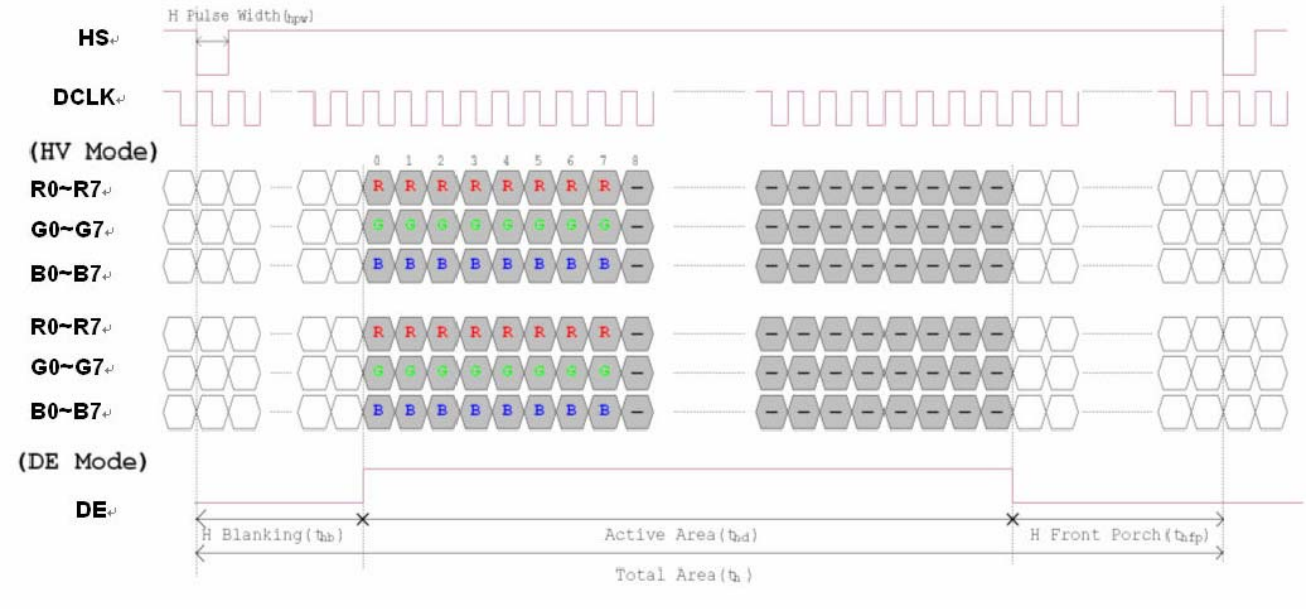
```
DE_PCLK_Rising();          //PDAT, DE, HSYNC etc. Panel fetches PDAT at the rising edge of PCLK
PDATA_Set_RGB();           //Parallel XPDAT[23:0] Output Sequence is set as RGB
DE_High_Active();          //Set XDE Polarity worked as High active.

//Horizontal display width(pixels) = (HDWR + 1) * 8 + HDWFTR Maximum value cannot over 2048
//Horizontal display width(pixels) = (0xa9+1) * 8 + 6 = 1366
LCD_CmdWrite(0x14);              //HDWR//Horizontal Display Width Setting Bit[6:0]
LCD_DataWrite(0xA9);             //Horizontal display width(pixels) = (HDWR + 1)*8
Delay_us(100);
LCD_CmdWrite(0x15);              //Horizontal Display Width Fine Tuning (HDWFT) [3:0]
LCD_DataWrite(0x06);
Delay_us(1);


//Horizontal non-display period(Back porch) = (HNDR + 1) * 8 + HNDFTR = (2+1)*8 + 6 = 34
LCD_CmdWrite(0x16);              //HNDR//Horizontal Non-Display Period Bit[4:0]
LCD_DataWrite(0x02);             //Horizontal Non-Display Period (pixels) = (HNDR + 1)*8
Delay_us(100);
LCD_CmdWrite(0x17);              //HNDR//Horizontal Non-Display Period Bit[4:0]
LCD_DataWrite(0x0a);             //Horizontal Non-Display Period Fine Tuning(HNDFT) [3:0]
Delay_us(100);

//HSYNC Start Position(Front porch) = (HSTR + 1)*8    = (0x06 + 1)*8 = 56
LCD_CmdWrite(0x18);              //HSTR//HSYNC Start Position[4:0]
LCD_DataWrite(0x06);             //HSYNC Start Position(PCLK) = (HSTR + 1)*8
Delay_us(100);

//HSYNC Pulse Width(pixels) = (HPW + 1)x8 = (0 + 1)*8 = 8
LCD_CmdWrite(0x19);              //HSYNC Pulse Width(HPW) [4:0]
LCD_DataWrite(0x00);             //HSYNC Pulse Width(pixels) = (HPW + 1)x8
Delay_us(100);

//Vertical Display Height(Line) = VDHR + 1 = (512 + 255 ) +1 =768
LCD_CmdWrite(0x1A);              //VDHR0 //Vertical Display Height Bit[7:0]
LCD_DataWrite(0xff);             //Vertical Display Height(Line) = VDHR + 1    =  255
Delay_us(10);
LCD_CmdWrite(0x1B);               //VDHR1 //Vertical Display Height Bit[10:8] = 512
LCD_DataWrite(0x02);              //Vertical Display Height(Line) = VDHR + 1
```

```
Delay_us(100);

//Vertical Non-Display Period(Back porch) = (VNDR + 1) =  (0 + 0x0c) + 1 =13
LCD_CmdWrite(0x1c);                 //VNDR0 //Vertical Non-Display Period Bit[7:0]
LCD_DataWrite(0x0c);                //Vertical Non-Display Period(Line) = (VNDR + 1)
Delay_us(100);
LCD_CmdWrite(0x1d);                 //VNDR1 //Vertical Non-Display Period Bit[9:8]
LCD_DataWrite(0x00);                //Vertical Non-Display Period(Line) = (VNDR + 1)
Delay_us(100);

// VSYNC Start Position(Front porch) = (VSTR + 1) = ( 0x0b + 1 )=12
LCD_CmdWrite(0x1e);                  //VSTR0 //VSYNC Start Position[7:0]
LCD_DataWrite(0x0c);                 //VSYNC Start Position(Line) = (VSTR + 1)
Delay_us(100);

////VSYNC Pulse Width(Line) = (VPWR + 1) = (0 + 1) = 1
LCD_CmdWrite(0x1f);                  //The pulse width of VSYNC in lines.
LCD_DataWrite(0x00);                 //VSYNC Pulse Width(Line) = (VPWR + 1)
Delay_us(100);
}
```

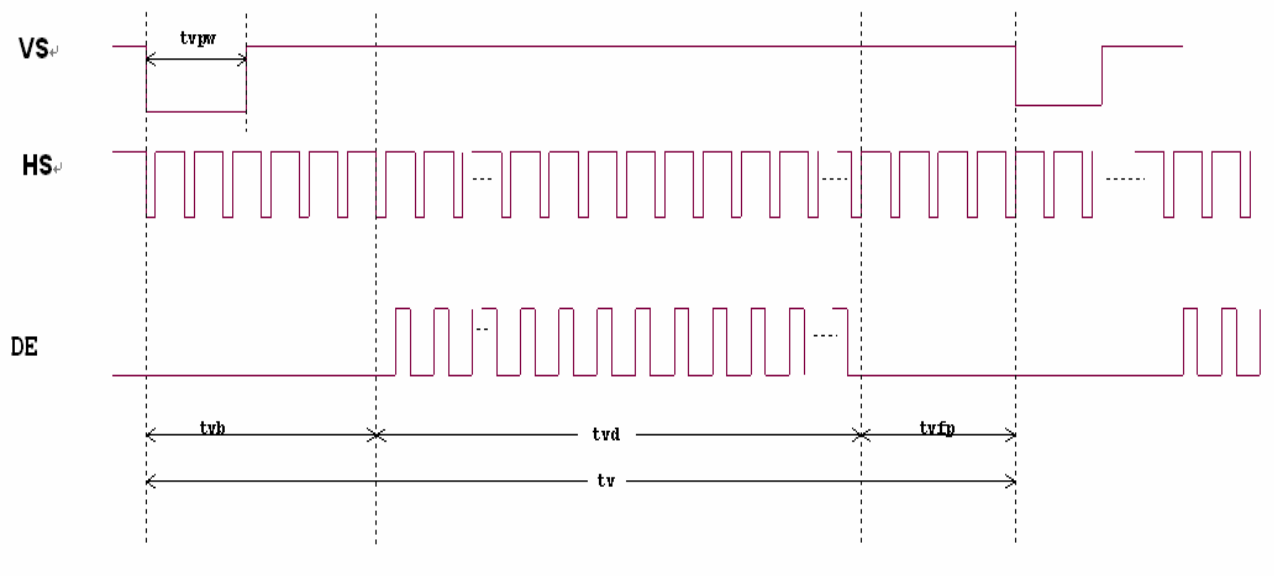
AT070TN92 Timing :

Item	symbol	value			Unit	Remark
		Min.	Typ.	Max.		
Horizontal Display Area	thd		800		DCLK	
DCLK Frequency(PCLK)	fclk	26.4	33.3	46.8	MHz	
One Horizontal Line	th	862	1056	1200	DCLK	
HS pulse width	thpw	1	-	40	DCLK	
HS Blanking(Back Porch)	thb	46	46	46	DCLK	
HS Front Porch	thfp	16	210	354	DCLK	

Item	symbol	value			Unit	Remark
		Min.	Typ.	Max.		
Vertical Display Area	tvd		480		TH	
VS period time	tv	510	525	650	TH	
VS pulse width	tvpw	1	-	20	TH	
VS Blanking(Back Proch)	tvb	23	23	23	TH	
VS Front Porch	tvfp	7	22	147	TH	



Horizontal Input Timing Diagram



Vertical Input Timing Diagram

```
DE_PCLK_Falling();
PDATA_Set_RGB();
LCD_CmdWrite(0x13);          // de --> low active
LCD_DataWrite(0x00);
delay_us(100);
//Horizontal display width(pixels) = (HDWR + 1) * 8 + HDWFTR Maximum value cannot over 2048
//Horizontal display width(pixels) = (0x63+1) * 8 + 0 = 800
```

```
LCD_CmdWrite(0x14);        //HDWR//Horizontal Display Width Setting Bit[6:0]
LCD_DataWrite(0x63);       //Horizontal display width(pixels) = (HDWR + 1)*8
delay_us(100);
LCD_CmdWrite(0x15);       //Horizontal Display Width Fine Tuning (HDWFT) [3:0]
LCD_DataWrite(0x00);
delay_us(100);

//Horizontal non-display period(Back porch) = (HNDR + 1) * 8 + HNDFTR = (4+1)*8 + 6 = 46
LCD_CmdWrite(0x16);        //HNDR//Horizontal Non-Display Period Bit[4:0]
LCD_DataWrite(0x04);       //Horizontal Non-Display Period (pixels) = (HNDR + 1)*8
delay_us(100);
LCD_CmdWrite(0x17);        //Horizontal Non-Display Period Fine Tuning(HNDFT) [3:0]
LCD_DataWrite(0x06);
delay_us(100);

//HSYNC Start Position(Front porch) = (HSTR + 1)*8    = (0x19 + 1)*8 = 208
LCD_CmdWrite(0x18);        //HSTR//HSYNC Start Position[4:0]
LCD_DataWrite(0x19);       //HSYNC Start Position(PCLK) = (HSTR + 1)*8
delay_us(100);

//HSYNC Pulse Width(pixels) = (HPW + 1)x8 = (0 + 1)*8 = 8
LCD_CmdWrite(0x19);        //HPWR//HSYNC Polarity ,The period width of HSYNC.
LCD_DataWrite(0x00);       //HSYNC Width [4:0]     HSYNC Pulse width(PCLK) = (HPWR + 1)*8
delay_us(100);

//Vertical Display Height(Line) = VDHR + 1 = (256 + 223 ) +1 = 480
LCD_CmdWrite(0x1A);         //VDHR0 //Vertical Display Height Bit [7:0]
LCD_DataWrite(0xdf);         //Vertical pixels = VDHR + 1
LCD_CmdWrite(0x1B);         //VDHR1 //Vertical Display Height Bit[10:8] = 256
LCD_DataWrite(0x01);        //Vertical Display Height(Line) = VDHR + 1
delay_us(100);

//Vertical Non-Display Period(Back porch) = (VNDR + 1) =    (0 + 0x15) + 1 =22
LCD_CmdWrite(0x1C);         //VNDR0 //Vertical Non-Display Period Bit [7:0]
LCD_DataWrite(0x15);        //Vertical Non-Display area = (VNDR + 1)
delay_us(100);
LCD_CmdWrite(0x1D);         //VNDR1 //Vertical Non-Display Period Bit [8]
LCD_DataWrite(0x00);        //Vertical Non-Display area = (VNDR + 1)
```

```
delay_us(100);

// VSYNC Start Position(Front porch) = (VSTR + 1) = ( 0x16 + 1 )=23
LCD_CmdWrite(0x1E);         //VSTR0 //VSYNC Start Position[7:0]
LCD_DataWrite(0x16);        //VSYNC Start Position(PCLK) = (VSTR + 1)
delay_us(100);

//VSYNC Pulse Width(Line) = (VPWR + 1) = (0 + 1) = 1
LCD_CmdWrite(0x1f);         //VPWR //VSYNC Polarity ,VSYNC Pulse Width[6:0]
LCD_DataWrite(0x01);        //VSYNC Pulse Width(PCLK) = (VPWR + 1)
delay_us(100);
```

附錄 3：RAIO 自建字庫

RAIO 自製自建字庫，提供三種大小的 ASCII 字體，分別為 8x12、16x24、32x48，本章節 API 支援 MCU 8bit color depth 8bpp、MCU 8bit color depth 16bpp、MCU 8bit color depth 16bpp、MCU 16bit color depth 16bpp、MCU 16bit color depth 24bpp mode2 幾種模式，但並不支援 MCU 16bit color depth 24bpp mode 1。

API:

```
// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void putPixel
(
unsigned short x //x of coordinate
,unsigned short y //y of coordinate
,unsigned long color
/*color : 8bpp:R3G3B2
        16bpp:R5G6B5
        24bpp:R8G8B8 */
)


// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void lcdPutChar8x12
(
unsigned short x // x of coordinate
,unsigned short y // y of coordinate
,unsigned long fgcolor //fgcolor : foreground color(font color)
,unsigned long bgcolor //bgcolor : background color
/*8bpp:R3G3B2
16bpp:R5G6B5
24bpp:R8G8B8*/
, unsigned char bg_transparent
/*bg_transparent = 0, background color with no transparent
bg_transparent = 1, background color with transparent*/
,unsigned char code //code : font char
)
```

```
// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void lcdPutString8x12
(
unsigned short x //x of coordinate
,unsigned short y //y of coordinate
, unsigned long fgcolor //fgcolor : foreground color(font color)
,unsigned long bgcolor //bgcolor : background color
/*8bpp:R3G3B2
16bpp:R5G6B5
24bpp:R8G8B8*/
, unsigned char bg_transparent
/*bg_transparent = 0, background color with no transparent
bg_transparent = 1, background color with transparent*/
,char *ptr //*ptr: font string
)


// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void lcdPutChar16x24
(
unsigned short x //x of coordinate
,unsigned short y //y of coordinate
,unsigned long fgcolor //fgcolor : foreground color(font color)
,unsigned long bgcolor //bgcolor : background color
/*8bpp:R3G3B2
16bpp:R5G6B5
24bpp:R8G8B8*/
, unsigned char bg_transparent
/*bg_transparent = 0, background color with no transparent
bg_transparent = 1, background color with transparent*/
,unsigned char code //code : font char
)
```

```
// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void lcdPutString16x24
(
unsigned short x //x of coordinate
,unsigned short y //y of coordinate
, unsigned long fgcolor //fgcolor : foreground color(font color)
, unsigned long bgcolor //bgcolor : background color
/*8bpp:R3G3B2
16bpp:R5G6B5
24bpp:R8G8B8/*
, unsigned char bg_transparent
/*bg_transparent = 0, background color with no transparent
bg_transparent = 1, background color with transparent*/
,char *ptr //*ptr : font string
)



// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void lcdPutChar32x48
(
unsigned short x //x of coordinate
,unsigned short y //y of coordinate
,unsigned long fgcolor //fgcolor : foreground color(font color)
,unsigned long bgcolor //bgcolor : background color
/*8bpp:R3G3B2
16bpp:R5G6B5
24bpp:R8G8B8*/
, unsigned char bg_transparent
/*bg_transparent = 0, background color with no transparent
bg_transparent = 1, background color with transparent*/
,unsigned char code //code : font char
)
```

```
// Note. this API does not support the case that MCU=16bit, 24bpp and mode1
void lcdPutString32x48
(
unsigned short x //x of coordinate
,unsigned short y //y of coordinate
, unsigned long fgcolor //fgcolor : foreground color(font color)
, unsigned long bgcolor //bgcolor : background color
/*8bpp:R3G3B2
16bpp:R5G6B5
24bpp:R8G8B8*/
, unsigned char bg_transparent,
/*bg_transparent = 0, background color with no transparent
bg_transparent = 1, background color with transparent*/
char *ptr //*ptr: font string
)
```

範例:

When color depth = 8bpp

```
lcdPutString8x12(0,0,0xe0,0x00,0,"sdfs6+55");
lcdPutString16x24(0,100,0x1c,0x00, 0,"sijsojfe565");
lcdPutString32x48(0,200,0x03,0x00,1,"sdjlfw5464ewr");
```

When color depth = 16bpp

```
lcdPutString8x12(0,0,0xf800,0x0000,0,"sdfs6+55");
lcdPutString16x24(0,100,0x07e0,0x0000, 0,"sijsojfe565");
lcdPutString32x48(0,200,0x001f,0x0000,1,"sdjlfw5464ewr");
```

When color depth = 24bpp

```
lcdPutString8x12(0,0,0xff0000,0x000000,0,"sdfs6+55");
lcdPutString16x24(0,100,0x00ff00,0x000000, 0,"sijsojfe565");
lcdPutString32x48(0,200,0x0000ff,0x000000,1,"sdjlfw5464ewr");
```

圖 8-1： 自建字畫面示意圖