RAiO

RA8889

应用程式介面

May 21, 2025

RAiO Technology Inc.

章节介绍

序言

在这份文件里，我们将为您介绍 8 种关於 RA8889 的功能。有 MCU 资料写入与显示资料输入格式、几何图形绘图、 DMA(Direct Memory Access)、 BTE(Block Transfer Engine)、 PIP(Picture in Picture)、文字和 PWM (Pulse Width Modulation)功能…等等，并提供关於这些功能的应用程式介面。

在一开始，您需要在 Userdef.h 这个档案里面，搭配您的环境，选择您所使用的 **MCU 通讯介面协定**、 **LCD Panel**、 **MCU 资料写入与显示资料输入格式**、**集通字库**与 **PLL 设定值**(SDRAM CLK 、 Core CLK 、 Pixel CLK)。详细的说明与设定，如 PLL Setting 、 SDRAM initialization 或 LCD initialization，请参考附录 1/2/3 的说明。

MCU 通讯介面协定: RA8889 提供了 Parallel 8080、Parallel 6800、SPI-3 wire、SPI-4 wire、IIC 等五种 MCU 通讯介面协定，可以参考 **RA8889_MCU_IF.c 与 RA8889_MCU_IF.h**。

SDRAM: RA8889 内建 128Mbits SDRAM。

MCU 资料写入与显示资料输入格式:

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.

集通字库: RA8889 可以经由外接集通字库，在显示上支援多种字体。

PLL 设定值: (SDRAM CLK、Core CLK、Pixel CLK):SDRAM CLK(最大 166MHz)、Core CLK(最大 133MHz 无内建字，120MHz 有内建字)、Pixel CLK(最大 80MHz)，可在 **userdef.h** 直接填入数值。

SDRAM CLK > Core CLK

在 16 位元色深情况下: Core CLK > 1.5*Pixel CLK

在 24 位元色深情况下: Core CLK > 2 * Pixel CLK

使用 API 功能需 Include 档案 ：userdef.h、RA8889.c、RA8889.h、RA8889_API.c、RA8889_API.h。

API Demo 需 Include 档案，绿色字体部分为 API Demo 使用的图资: API_Demo.c、API_Demo.h、8bit_8bpp_icon.h、8bit_16bpp_icon.h、8bit_24bpp_icon.h、8bit_test.h、16bit_16bpp_icon.h、16bit_24bpp1_icon.h、16bit_24bpp2_icon .h、16bit_test.h、pic8.h、pic16.h、pic24.h、pic168.h、pic1616.h、pic1624.h、pic16241.h。

附录四 RAIO 自建字库使用档案 ：ascii_table_8x12.h、ascii_table_16x24.h、ascii_table_32x48.h

注 ：MCU 通讯介面协定、MCU 资料写入与显示资料输入格式、集通字库、PLL 设定值皆在 userdef.h 设定

第一章: MCU 资料写入与显示资料输入格式

概要:

RA8889 提供六种模式给 MCU 对 SDRAM 做资料写入:

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)

本章节将会帮助使用者简单的了解如何使用这六种模式

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 资料格式

API :

使用 8 bits MCU 介面以及 8bpp color depth. MCU 写资料到 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
)
```

范例:

```
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 的图资，格式为 MCU 8 位元介面，色深 8 位元时所使用。
```

LCD 显示示意图:

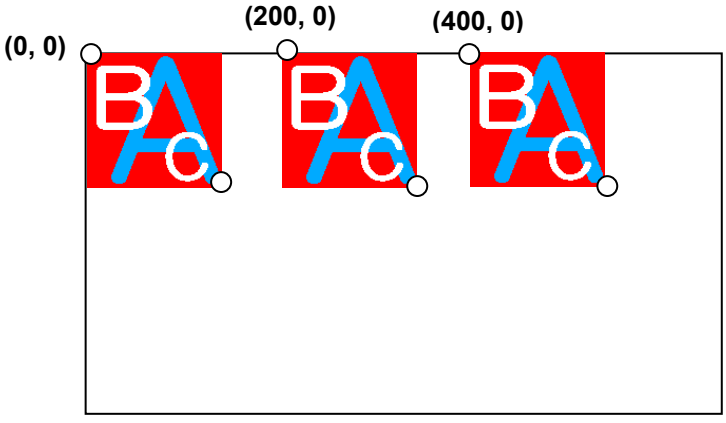


图 1.1：使用 8 bits MCU 介面以及 8bpp color depth. MCU 写资料到 SDRAM.

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 资料格式

API :

使用 8 bits MCU 介面以及 16bpp color depth. MCU 写资料到 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
)
```

范例:

```
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 的图资，格式为 MCU 8 位元介面，色深 16 位元时所使用。
```

LCD 显示示意图:

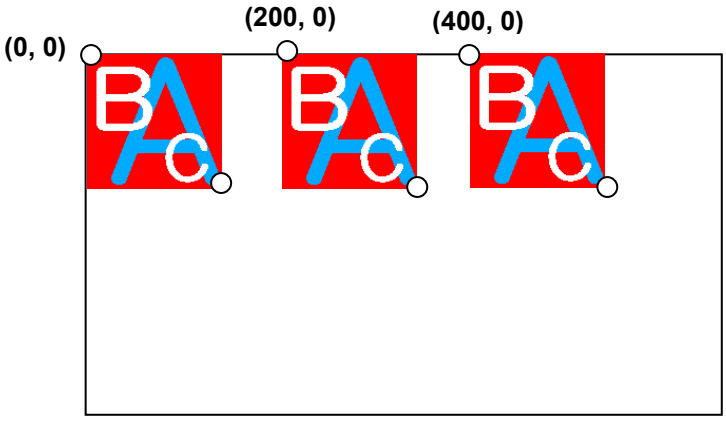


图 1.2 : 使用 8 bits MCU 介面以及 16bpp color depth. MCU 写资料到 SDRAM.

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 资料格式

API :

使用 8 bits MCU 介面以及 24bpp color depth. MCU 写资料到 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.
)
```

范例:

```
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 的图资，格式为 MCU 8 位元介面，色深 24 位元时所使用。
```

LCD 显示示意图:

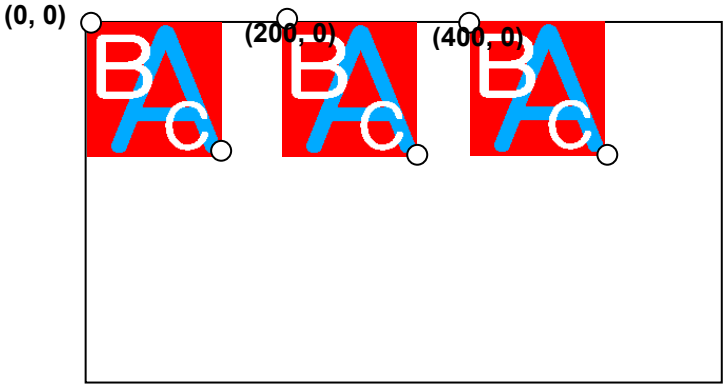


图 1.3 : 使用 8 bits MCU 介面以及 24bpp color depth. MCU 写资料到 SDRAM.

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 资料格式

API :

使用 16 bits MCU 介面以及 16bpp color depth. MCU 写资料到 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 //16bit data.
)
```

范例:

```
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 的图资，格式为 MCU 16 位元介面，色深 16 位元时所使用。
```

LCD 显示示意图:

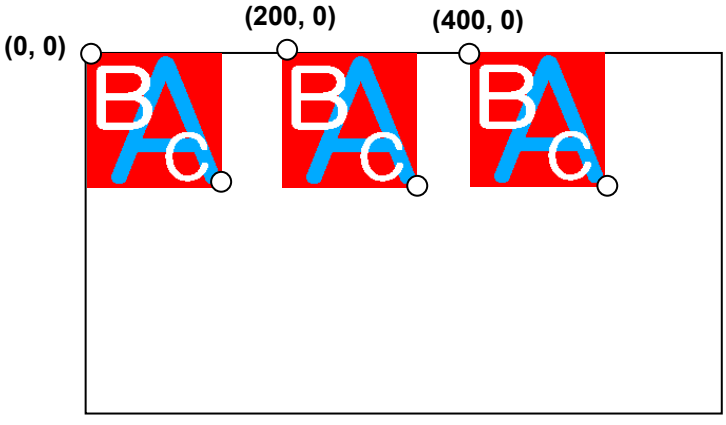


图 1.4 : 使用 16 bits MCU 介面以及 16bpp color depth. MCU 写资料到 SDRAM.

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 资料格式

API :

使用 16 bits MCU 介面以及 24bpp color depth mode1. MCU 写资料到 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 //16bit data.
)
```

范例:

```
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 的图资，格式为 MCU 16 位元介面，色深 24 位元的 mode1 下所使用。
```

LCD 显示示意图:

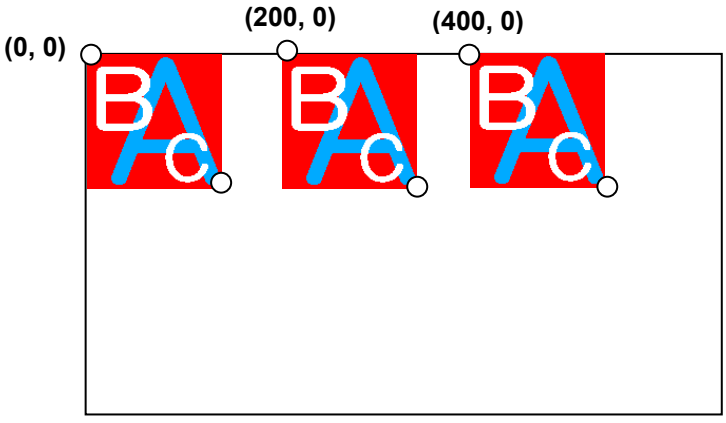


图 1.5 : 使用 16 bits MCU 介面以及 24bpp color depth mode1. MCU 写资料到 SDRAM

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 资料格式

API：

使用 16 bits MCU 介面以及 24bpp color depth mode2. MCU 写资料到 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 //16bit data.
)
```

范例:

```
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 的图资，格式为 MCU 16 位元介面，色深 24 位元的 mode2 下所使用。
```

LCD 显示示意图:

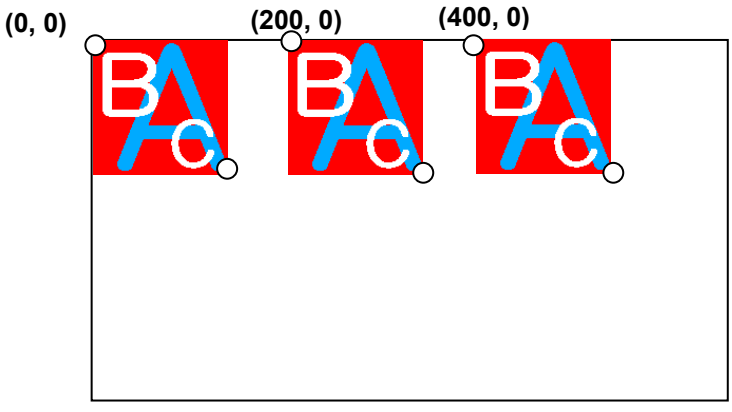


图 1.6：使用 16 bits MCU 介面以及 24bpp color depth mode2. MCU 写资料到 SDRAM.

第二章 ： 几何图形绘图

概要:

在许多的人机介面应用中，时常需要去显示一些几何图形用来当作按键，感应器或者是其他的几何图形标志。当使用者想要透过 MCU 韧体来达到这个需求时，使用者会耗费很多的系统资源或心力，去做数学运算或者是图库编纂。

RA8889 提供了几何绘图引擎，使用者只要下一些指令，就可以轻易的在 TFT-LCD 上达到几何图形显示。使用者仅需选择几何图形的颜色与它是不是应该被填满。这份应用说明将会帮助使用者去了解，如何使用 RA8889 的几何图形绘图功能。

2.1: 椭圆/圆 输入

RA8889 提供绘制椭圆/圆形绘图的功能，让使用者以简易或者低速的 MCU 就可以在 TFT LCD 模组上迅速做画圆的工作。首先，先设定椭圆/圆形的中心点 REG[7Bh~7Eh]，其次，设定椭圆的长轴与短轴或圆的半径 REG[77h~7Ah]，设定椭圆/圆的颜色 REG[D2h~D4h]，设定画椭圆/圆 REG[76h]Bit5=0 和 Bit4=0，然後启动绘图 REG[76h]Bit7 = 1，RA8889 就会自动将椭圆/圆图形写入显示资料用的记忆体，此外，使用者可以经由设定 REG[76h]Bit6= 1，来画出实心椭圆/圆。

注意:椭圆/圆的中心必须要在 active windows 里面
写入程序请参照下图:

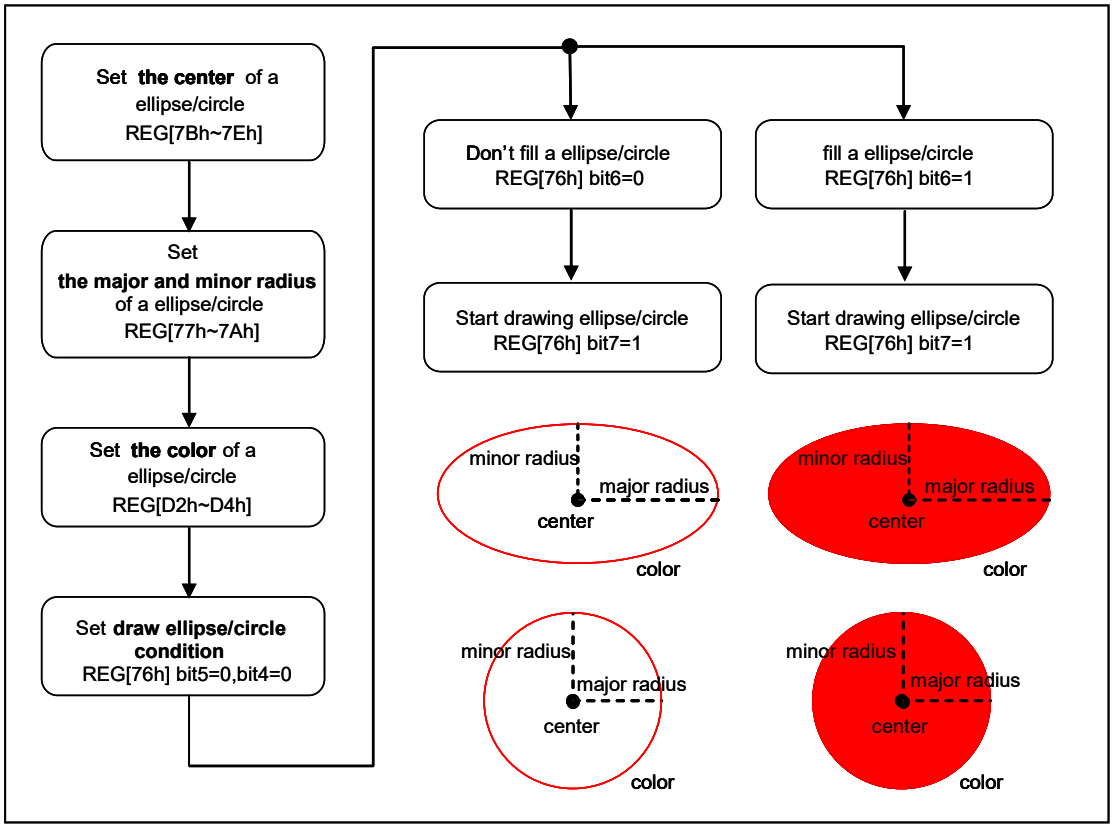


图 2.1 画圆形(填满)与椭圆形(填满)的程式流程图

画橢圓形或圓形的 **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
)
```

范例 1(画圆形):

```
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);
```

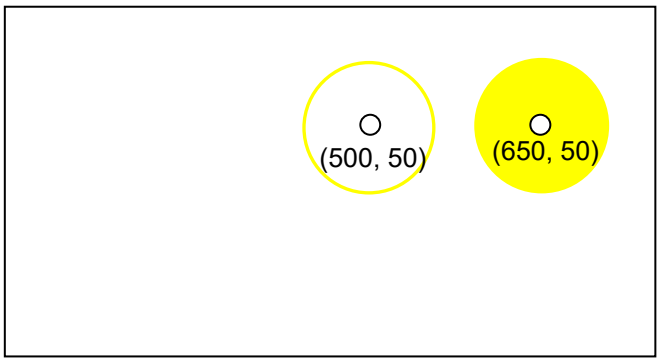


图 2.2： 画一个颜色为黄色的圆，圆心位置(500,50)，半径 = 50，
和画一个被黄色填满的圆型，圆心位置(650,50)，半径 = 50。

范例 2(画椭圆形):

```
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);
```

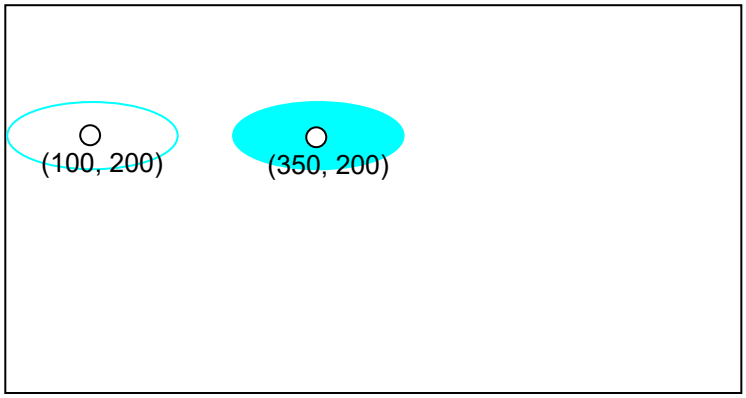


图 2.3：画一个青色的圆，圆心位置(100,200)、X 轴半径 = 100 、Y 轴半径 = 50，和画一个青色的实心圆，圆心位置(350,200)、X 轴半径 = 100 、Y 轴半径 = 50。

2.2: 曲线输入

RA8889 提供曲线绘图功能，让使用者以简易或低速的 MCU 就可以在 TFT 模组上画曲线。先设定曲线的中心点 REG[7Bh~7Dh]，曲线的长轴与短轴 REG[77h~7Ah]，曲线的颜色 REG[D2h~D4h]，曲线的相关参数为 REG[76h] Bit5=0 与 Bit4=1，REG[76h] Bit[1:0] 是椭圆的曲线部分，然後启动绘图设定 REG[76h] Bit7 = 1，RA8889 就会自动将曲线的图形写入显示资料的记忆体。此外，使用者可以经由设定 REG[76h]Bit6 = 1 来画出实心曲线。

注意曲线的中心必须要在 active windows 里面
写入程序请参照下图:

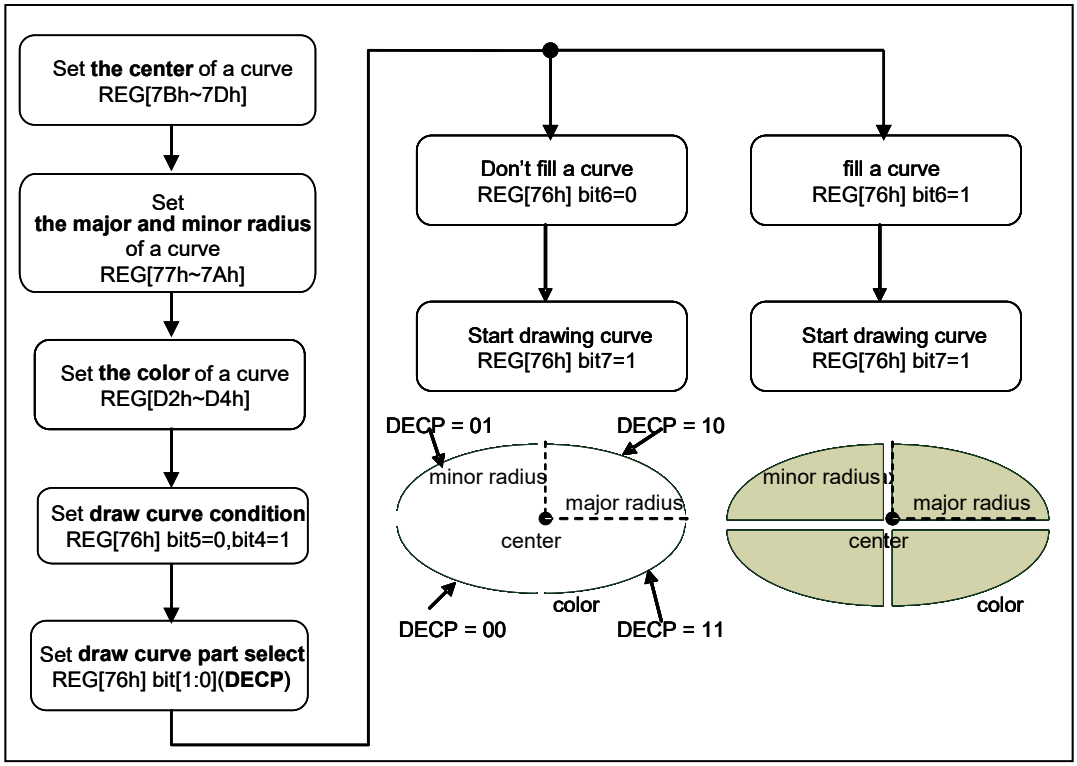


图 2-4 画圆弧与圆弧形填满的程式流程图

画圆弧与画圆弧填满功能的 **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
)
```

范例:

```
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);
```

本范例程式包含 8 个步骤:

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

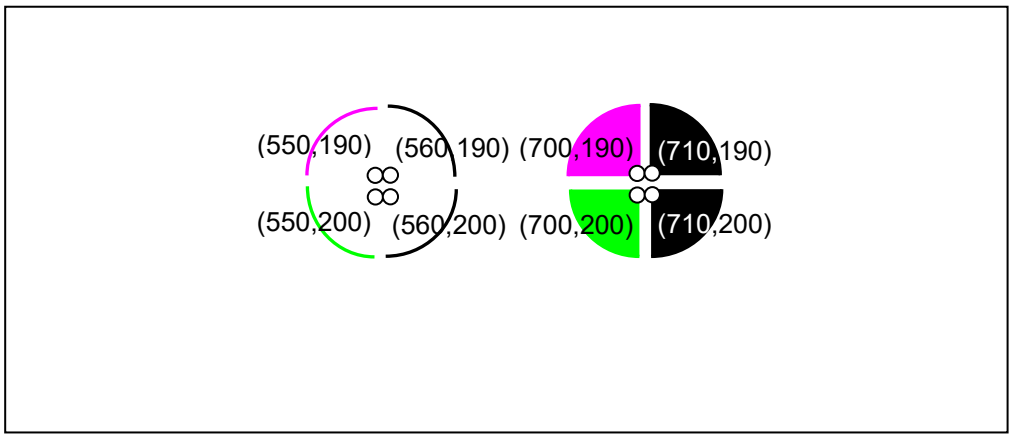


图 2.5：画圆弧与圆弧形填满的 LCD 显示示意图

2.3: 方形输入

RA8889 提供方形绘图功能，让使用者以简易或低速的 MCU 就可以在 TFT 模组上画方形。先设定方形的起始点 REG[68h~6Bh]与结束点 REG[6Ch~6Fh]，方形的颜色 REG[D2h~D4h]，然後设定画方形设定 REG[76h]Bit5=1，Bit4=0 和启动绘图 REG[76h]Bit7 = 1，RA8889 就会自动将方形的图形写入显示资料的记忆体，此外，使用者可以藉由设定 REG[76h]Bit6 = 1 来画出实心方形。

注意:方形的起始点与结束点必须要在 active windows 里面
写入程序请参照下图:

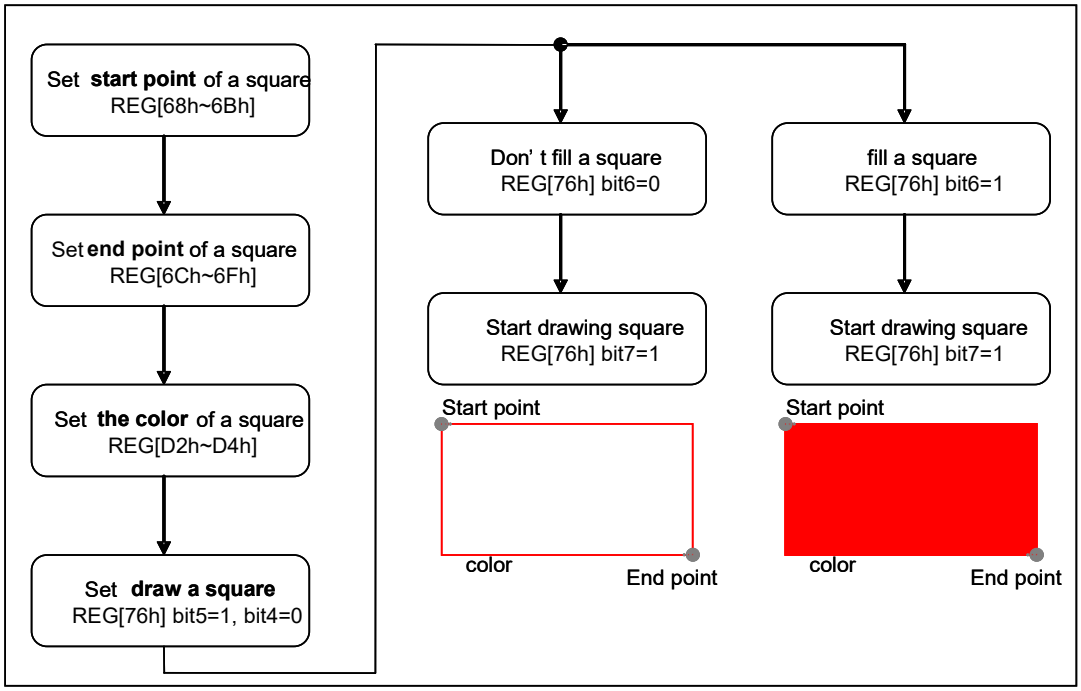


图 2.6：画方形与方形填满的程式流程图

画方形的 **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

)
```

范例 **1:**

```
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);
```

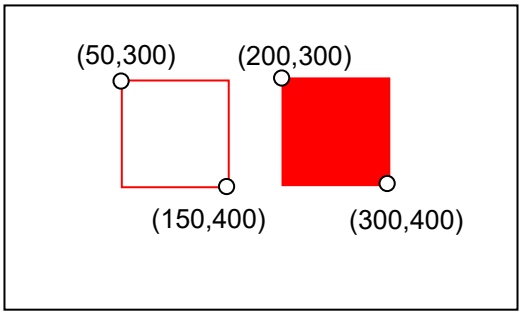


图 **2.7**：从点**(50,300)**到点**(150,400)**，画一个红色的正方形，以及从点**(200,300)**到点**(300,400)**，画一个红色被填满的方形。

2.4: 画直线功能

RA8889 支援画直线的绘图功能，让使用者以简易或低速的 MCU 就可以在 TFT 模组上画线，先设定起始点 REG[68h~6Bh]，结束点 REG[6Ch~6Fh]和线的颜色 REG[D2h~D4h]，然後设定画线参数 REG[67h]Bit1 = 0，和启动绘图 REG[67h]Bit7 = 1，RA8889 就会自动将线的图形写入显示资料的记忆体。

注意: :线的起始点与结束点必须要在 active windows 里面
写入程序请参照下图:

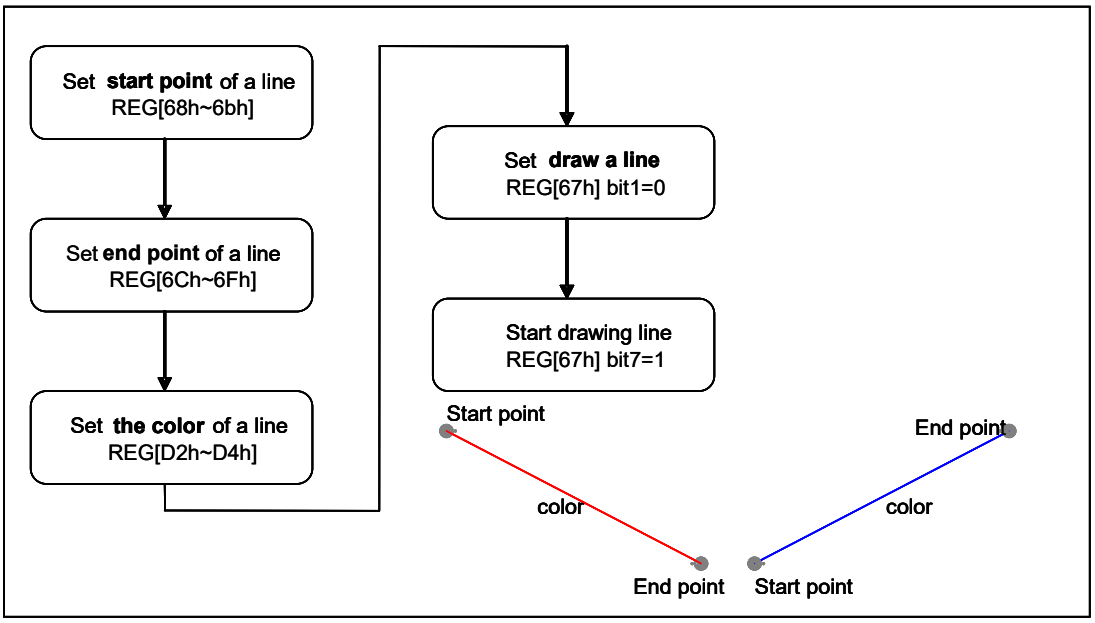


图 2.8: 画直线的程式流程图

画直线的 **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

)
```

范例:

```
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);
```

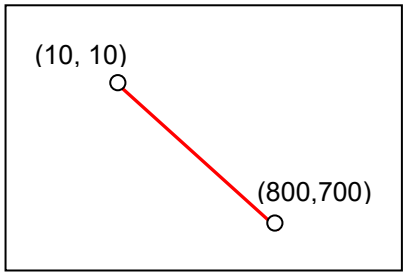


图 2.9：从点(10,10)到点(800,700)，画一条红色的直线。

2.5: 画三角形的功能

RA8889 支援三角形的绘图功能，让使用者以简易或低速的 MCU 就可以在 TFT 模组上画三角形。先设定三角的第 0 点 REG[68h~6Bh]、第 1 点 REG[6Ch~6Fh]、第 2 点 REG[70h~73h]和三角形的颜色 REG[D2h~D4h]，设定画三角形的参数 REG[67h] Bit1 = 1，然後启动绘图 REG[67h] Bit7 = 1，RA8889 就会自动将三角形的图形写入显示资料的记忆体。此外，使用者可以透过设定 REG[67h] Bit5 = 1 来画出实心三角形。

注意:三角形的第 0 点第 1 点与第 2 点与结束点必须要在 active windows 里面
写入程序请参照下图:

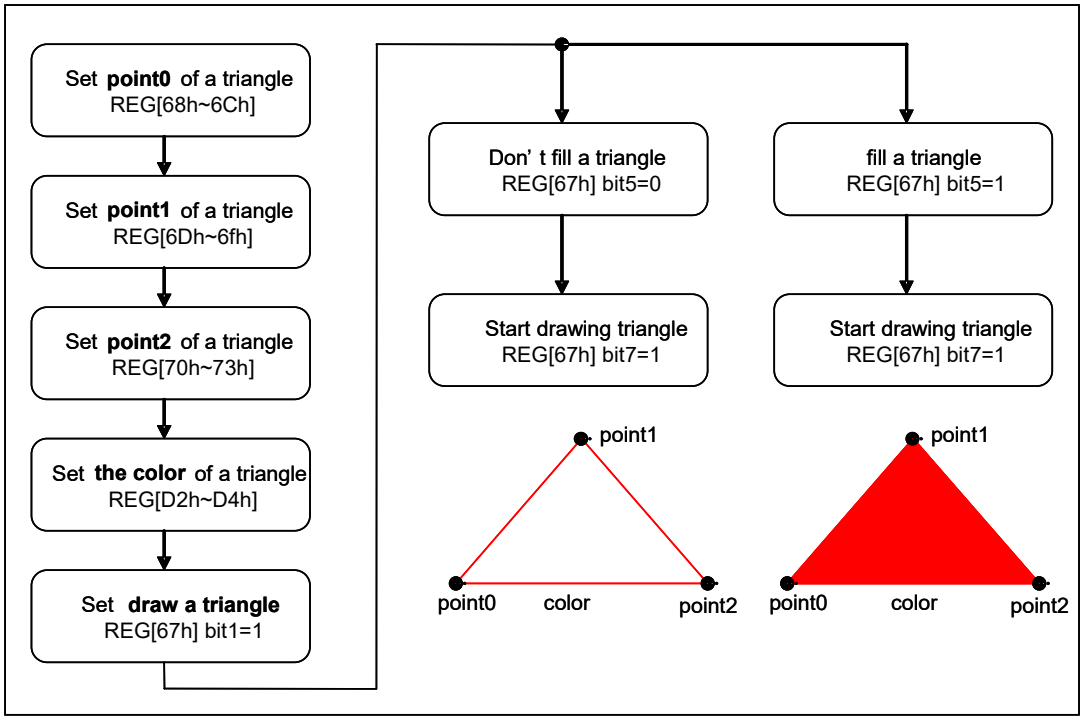


图 2.10：画三角形与三角形填满的程式流程图

画三角形的 **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
)
```

范例 **1:**

```
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);
```

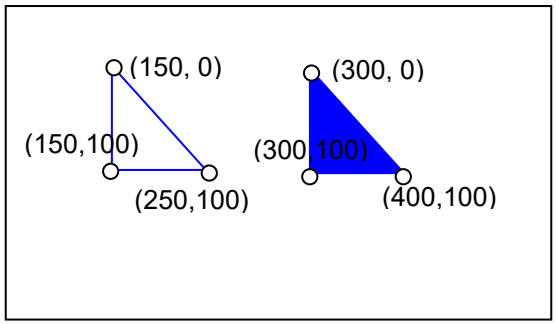


图 2.11：用(150,0)、(150,100)、(250,100)三点画一个蓝色的三角形，以及用(300,0) 、(300,100)、(400,100)三点画一个蓝色被填满的三角形。

2.6: 画圆角方形的功能

RA8889 支援圆角方形绘图功能，让使用者以简易或低速的 MCU 就可以在 TFT 模组上画方形。经由设定方形的起始点 REG[68h~6Bh]与结束点 REG[6Ch~6Fh]，圆角的长轴与短轴 REG[77h~7Ah]，圆角方形的颜色 REG[D2h~D4h]，然後设定画方形设定 REG[76h]Bit5=1，Bit4=1 和启动绘图 REG[76h]Bit7 = 1，RA8889 就会自动将圆角方形的图形写入显示资料的记忆体，此外，使用者可以藉由设定 REG[76h]Bit6 = 1 来画出实心的圆角方形。

提醒 1：(结束点 X － 起始点 X) 必须要大於(2*长轴 + 1) 且 (结束点 Y － 起始点 Y) 必须要大於(2*短轴 + 1)

提醒 2： 起始点与结束点需要在 active windows 内

写入程序请参照下图:

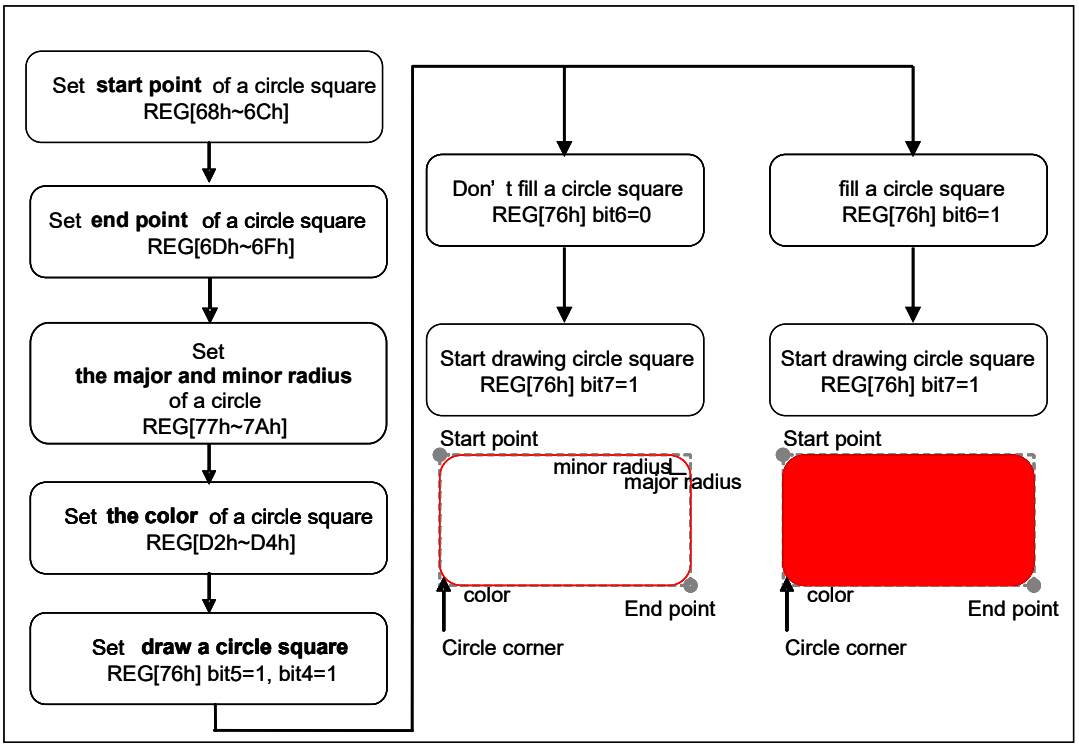


图 **2.12：** 方形与圆角方形填满的程式流程图画圆角

画圆角方形的 **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
)
```

范例 **1:**

```
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);
```

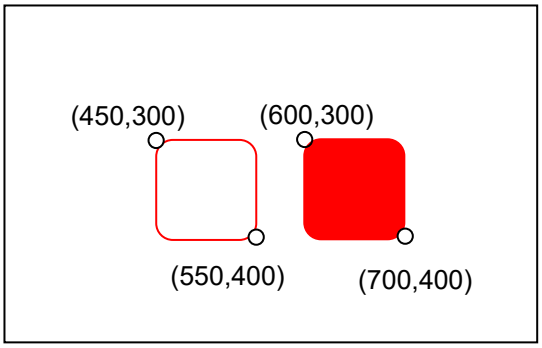


图 2.13：长轴为 20，短轴为 30，从点(450,300)到点(550,400)，画一个红色的圆角方形，
以及长轴为 20，短轴为 30，从点(600,300)到点(700,400)，画一个红色被填满的圆角方形

第三章 ： 串列式 Flash/ROM 控制单元

RA8889 建立了串列式 Flash/ROM 的介面，來支援下列的传输模式: 4-BUS 正常讀取(Normal Read)、5-BUS 快速讀取 (FAST Read)、双倍模式 0 (Dual mode 0)、双倍模式 1 (Dual mode 1)模式 0 (Mode 0) 、模式 3 (Mode 3)与 Quad mode。

串列式 Flash/ROM 记忆体功能可用在文字模式 (FONT Mode)、 DMA 模式(直接记忆存取模式)。文字模式意指外部串列式 Flash/ROM 记忆体被当成字体点阵图的來源。为了支援文字字体，RA8889 可与专业的字体供应商 — 上海集通公司的 FONT ROM 相容。DAM 模式意指串列式 Flash/ROM 可当作 DMA (Direct Memory Access) 的资料來源。使用者可以透过此模式，加快资料传送到显示记忆体(Display RAM) 的速度。

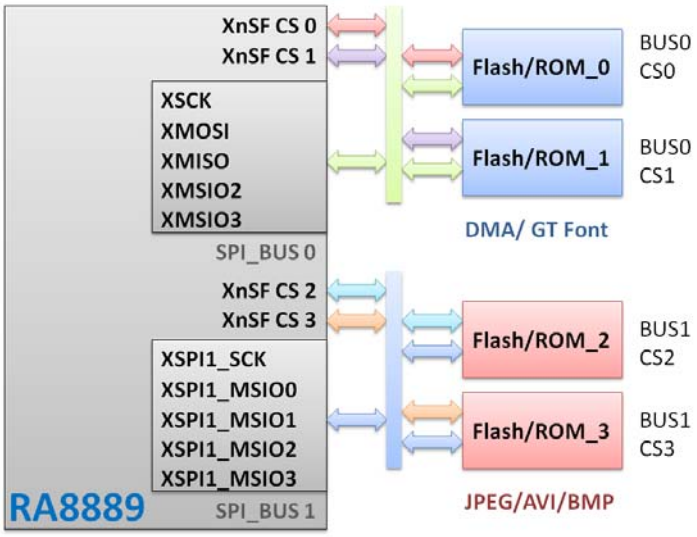


圖 3.1: RA8889 串列式 Flash/ROM 系统

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 In Block Mode

DMA block mode 是一个将图片从外部快闪记忆体搬(Serial Flash Memory)移到 RA8889 用的显示记忆体(SDRAM)，DMA 功能的使用的单位是像素(pixel)。关於 DMA 功能的流程图，请参照以下的叙述。

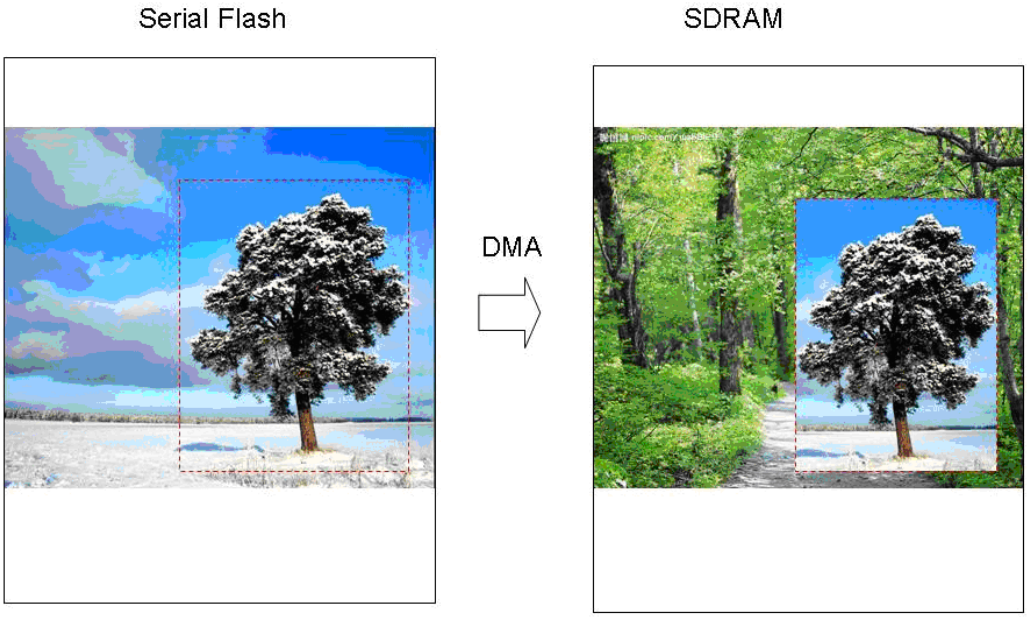


图 3.1 : DMA 功能

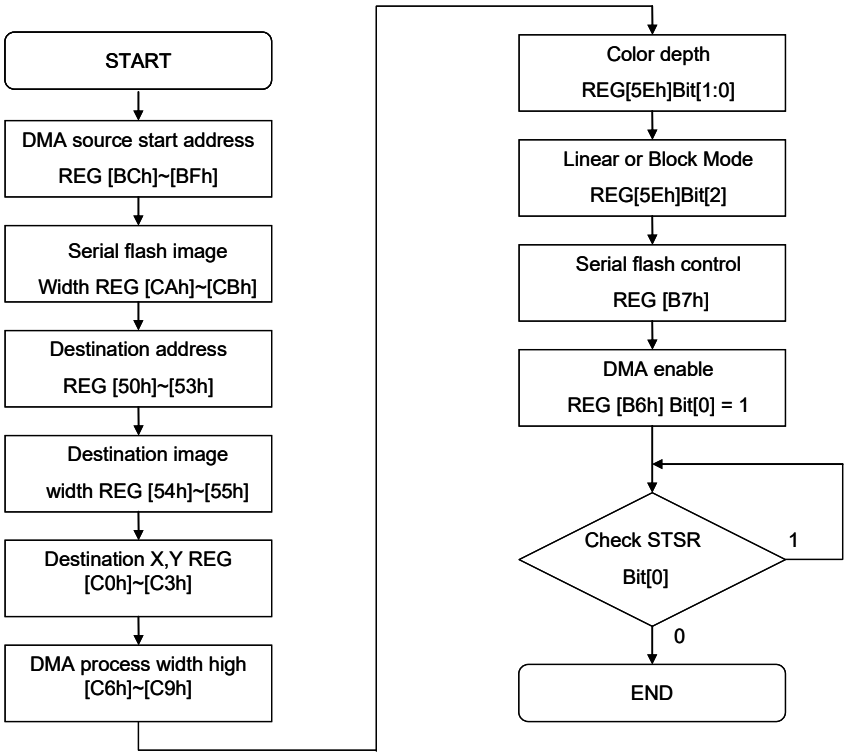


图 3.21 : 启动DMA功能的程式流程 － With Check Flag operation (STSR.0)

3.1.2: DMA 功能的 API 在 LCD 上的显示结果:

我们提供下列 7 组 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
)
```

范例 ：

```
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.

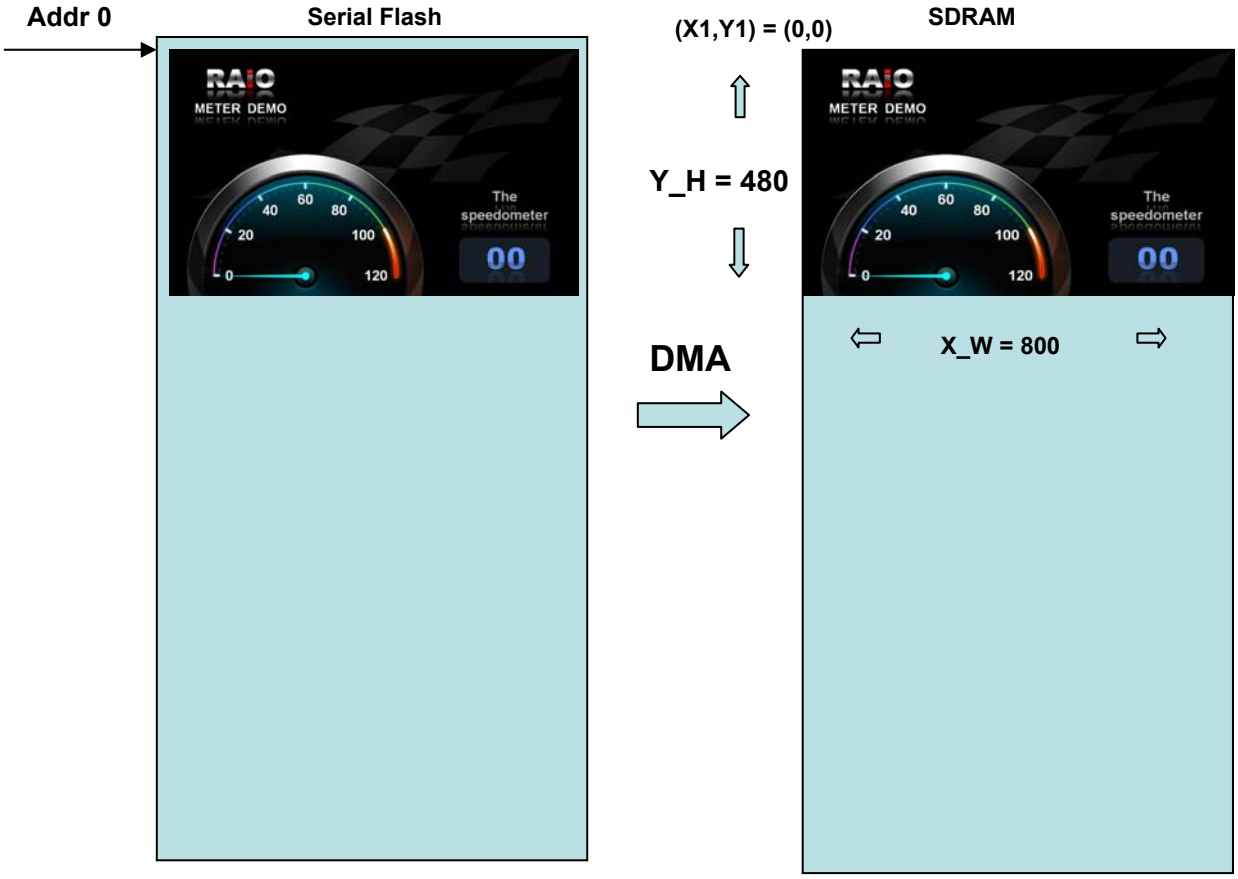


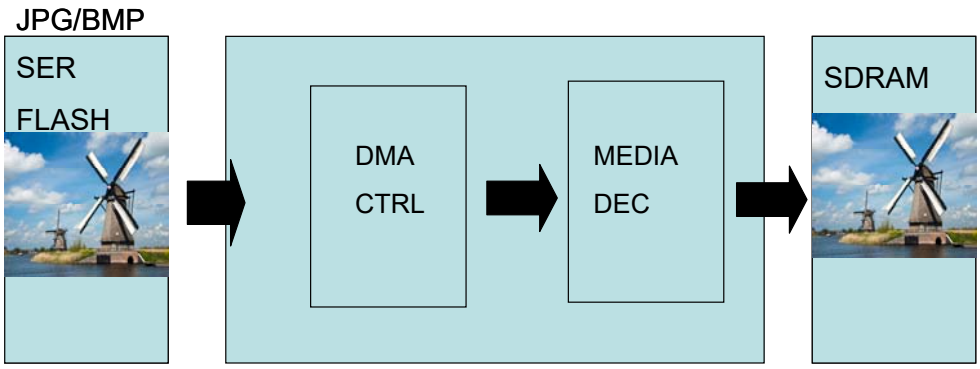
Figure 3.4: 使用 DMA 功能将图资从 Flash 中读出并写入 SDRAM

3.2.1： 媒体解码单元(Media Decode Unit)

RA8889 支援媒体解码单元，包含 JPEG（ISO/IEC 10918-1 Baseline profile，YUV444，YUV422，YUV420，YUV400，并不支援重启间隔格式），BMP（raw data）和 AVI（motion jpeg）格式。 RA8889 可以自动区分上述三种格式，并将其自动解析为相对的解码器。在视频功能中，RA8889 提供自动播放，暂停和停止功能。使用者应事先将图像或视频下载到串列快闪记忆体中，并通过设定 DMA、CANVAS 和 PIP 相对的暂存器让他们显示在 LCD 萤幕上。

图像写入的位址，请参考 CANVAS 相关暂存器。由于视频显示在 PIP1 或 PIP2 窗口中，因此使用者应在播放视频之前设定 PIP 相关暂存器。此外，RA8889 还提供中断和忙碌标志进行检查。
注意，
1.关于串列快闪存介面，请使用支援 Quad mode 串列快闪记忆体，建议核心时脉频率高于 100MHz。
2. AVI frame rate 可以是 30、29.97、25、24、23.97、20 和 15。
3. PIP 的色深应与主要视窗的色深一致。
4. AVI / JPG 的宽度和高度必须是 8 的倍数。
5.串列快闪记忆体的 DMA 长度应等于图像或视频的档案容量。

3.2.2: 硬体图像的解码流程



Reference CANVAS REG for write SDRAM data

Figure 3-5

3.2.3 ： 图像解码流程图

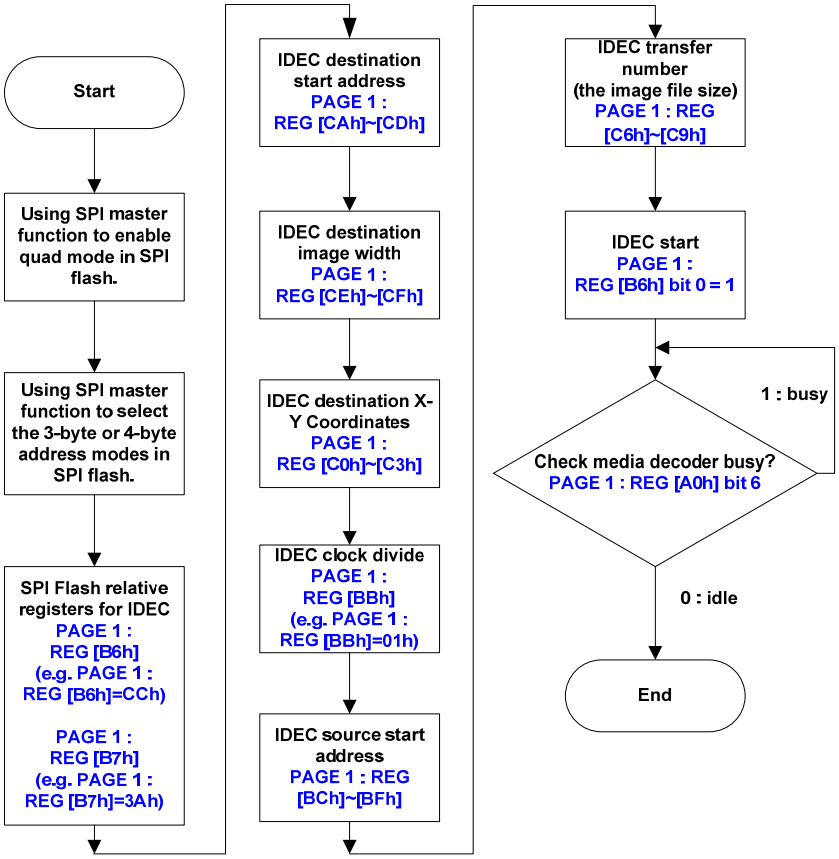
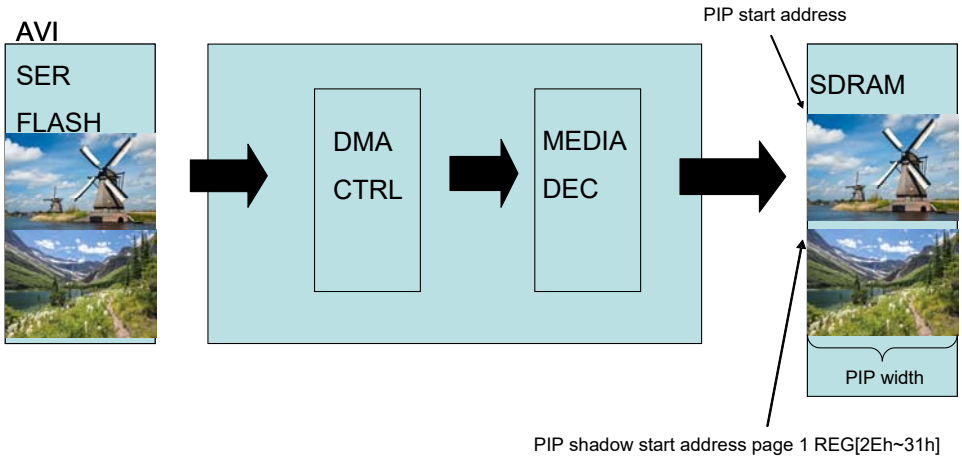


Figure 3-6

注 ：IEDC 意即支持媒体译码单元的影像格式(JPEG,BMP 及 AVI)

3.2.4 : AVI 的解码流程



Reference PIP REG for write SDRAM data

Figure 3-7

3.2.5: AVI 解码流程图

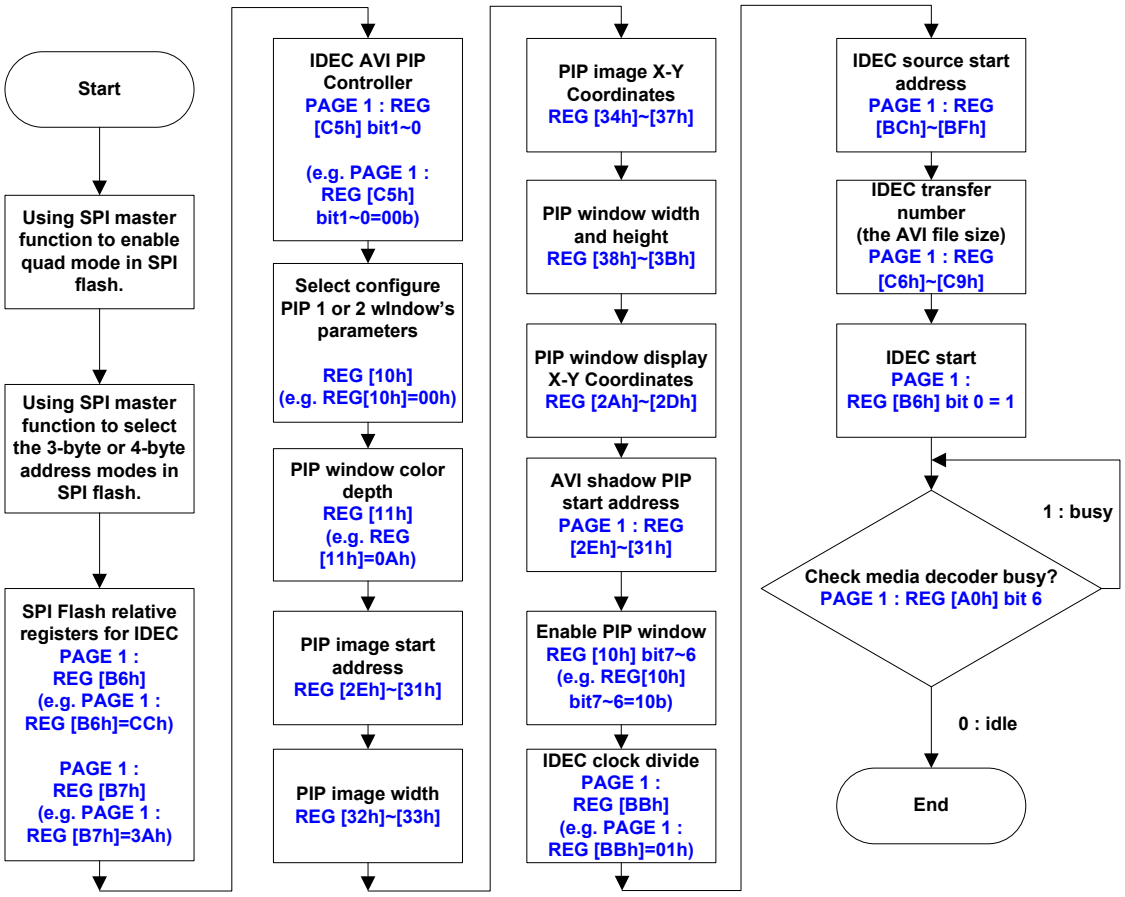


Figure 3-8

注 ：IEDC 意即支持媒体译码单元的影像格式(JPEG,BMP 及 AVI)

3.2.6 媒体译码(Media decode)功能 API 范例与 LCD 影响显示结果:

以下提供 7 个相关的 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：仅支援 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 范例:

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).

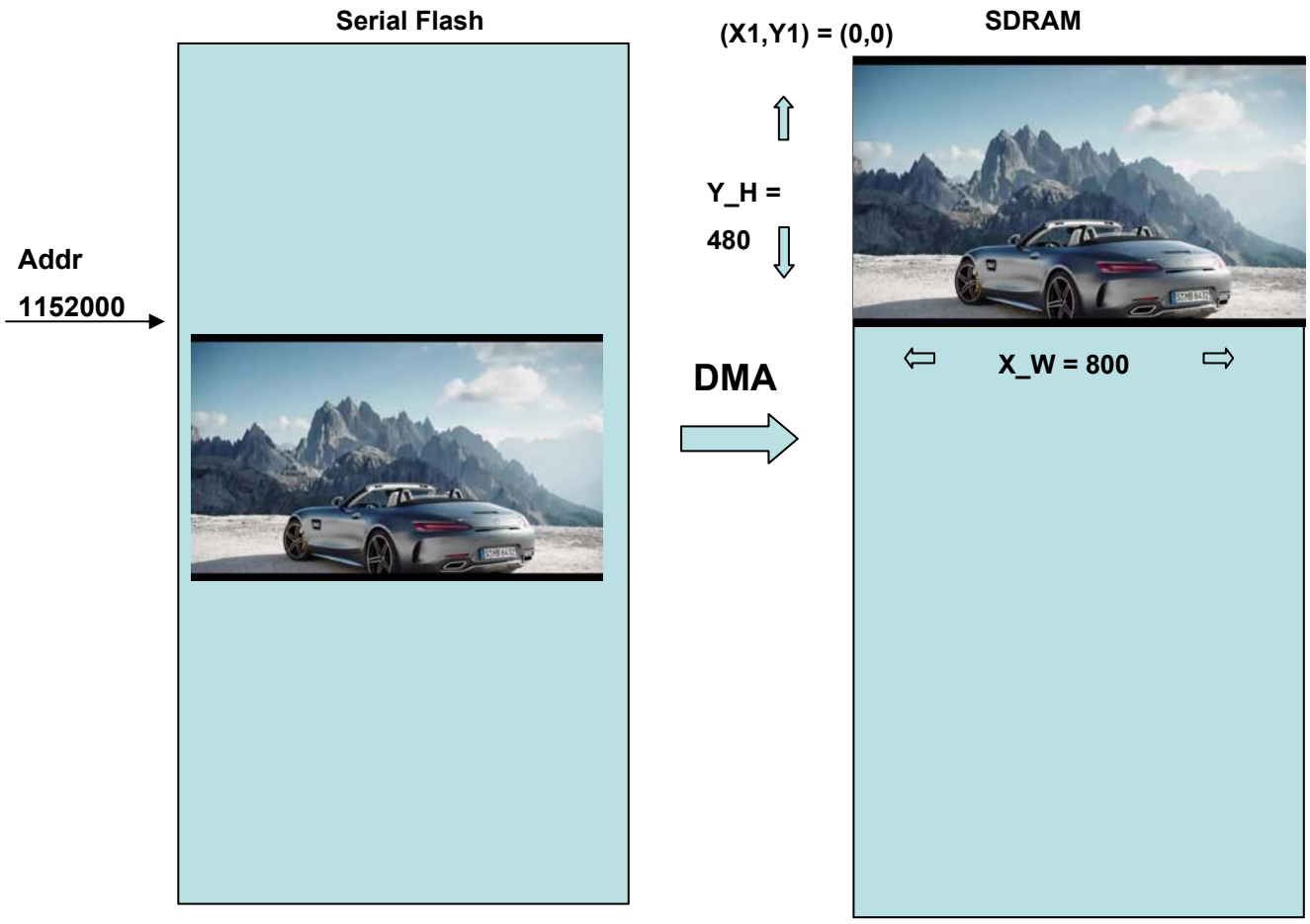


Figure 3.9: 使用 Media decode 功能将存在外接串列 Flash 的 JPG 图资写入到 SDRAM 中

AVI 范例:

```
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

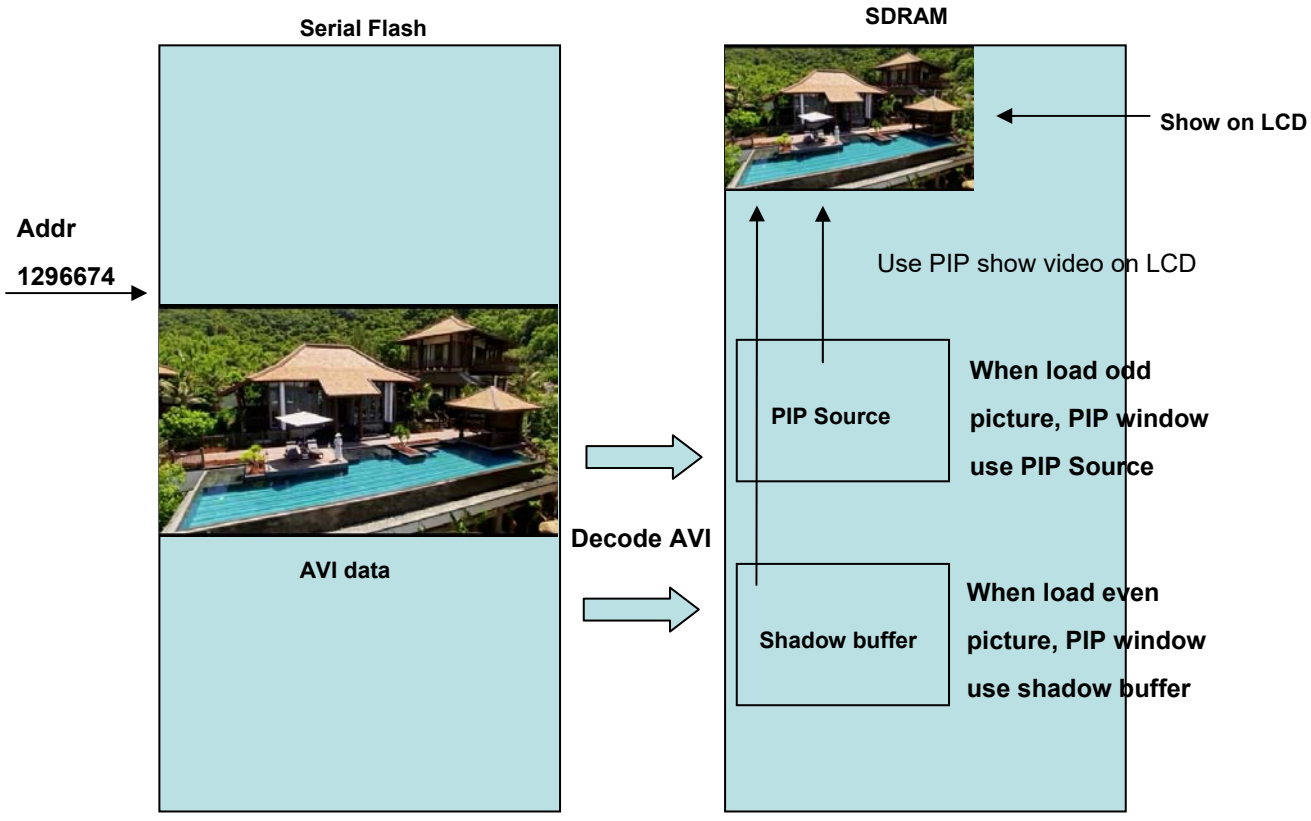


Figure 3.10: 使用 **Media decode** 功能将存在外接串列 **Flash** 的 **AVI** 档，写入到 **SDRAM** 中，并显示在 **LCD** 上

第四章 ：Block Transfer Engine

概要:

RA8889 内建了增加区域运算效能的区域运算引擎(BTE)，当有一个区块资料需要被搬移或是和某些特定资料做一些逻辑运算，RA8889 可以经由 BTE 硬体加速来简化 MCU 的程式，这边我们将要来讨论 BTE 引擎的操作与功能。在使用 BTE 功能之前，使用者必须选择相对应的 BTE 功能，RA88895 支援 13 种 BTE 功能。关於功能描述，请参照表 4-1。每个 BTE 功能针对不同的应用最多支援 16 种光栅运算(ROP)。它们可以提供给光栅运算来源与光栅运算目的地不同的逻辑结合。通过结合 BTE 功能与光栅运算，使用者可以达到许多非常有用的应用操作。请参考章节後面的详细描述。这份应用说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.** "如果我们的客户需要其他 BTE 功能的解说，请洽询瑞佑科技业务部或代理商。

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: 来源 0 的资料, S1: 来源 1 的资料, D: 目的地的资料
2. 以 pattern fill 功能来说,目的地的资料是由来源来表示。

Example:

当 ROP 功能设定为 Ch (1100b), 目的地的资料 = 来源 0 的资料
当 ROP 功能设定为 Eh (1110b), 目的地的资料 = 来源 0 + 来源 1
当 ROP 功能设定为 2h (0010b), 目的地的资料 = ~来源 0・来源 1
当 ROP 功能设定为 Ah (1010b), 目的地的资料 = 来源 1 资料

BTE Access Memory Method

伴随着这些设定，BTE 的来源/目的地的资料被当作一个显示的区块，下面的例子解释了将来源 0/来源 1/目的地定义成区块的方法。

BTE Chroma Key (Transparency Color) Compare

当启用 BTE 通透色时，BTE 程序会比对来源 0 的资料和背景色寄存器的资料。资料相等的部分在目的地并不会有所改变，资料不相等的部分才会由来源 0 写入到目的地。

当来源色彩深度 = 256 色时，

来源 0 红色部分只比对 REG[D5h]Bit[7:5]

来源 0 绿色部分只比对 REG [D6h] Bit [7:5]

来源 0 蓝色部分只比对 REG [D7h] Bit [7:6]

当来源色彩深度 = 65k 色时，

来源 0 红色部分只比对 REG[D5h]Bit[7:3]

来源 0 绿色部分只比对 REG [D6h] Bit [7:2]

来源 0 蓝色部分只比对 REG [D7h] Bit [7:3]

当来源色彩深度 = 16.7M 色时，

来源 0 红色部分只比对 REG[D5h]Bit[7:0]

来源 0 绿色部分只比对 REG [D6h] Bit [7:0]

来源 0 蓝色部分只比对 REG [D7h] Bit [7:0]

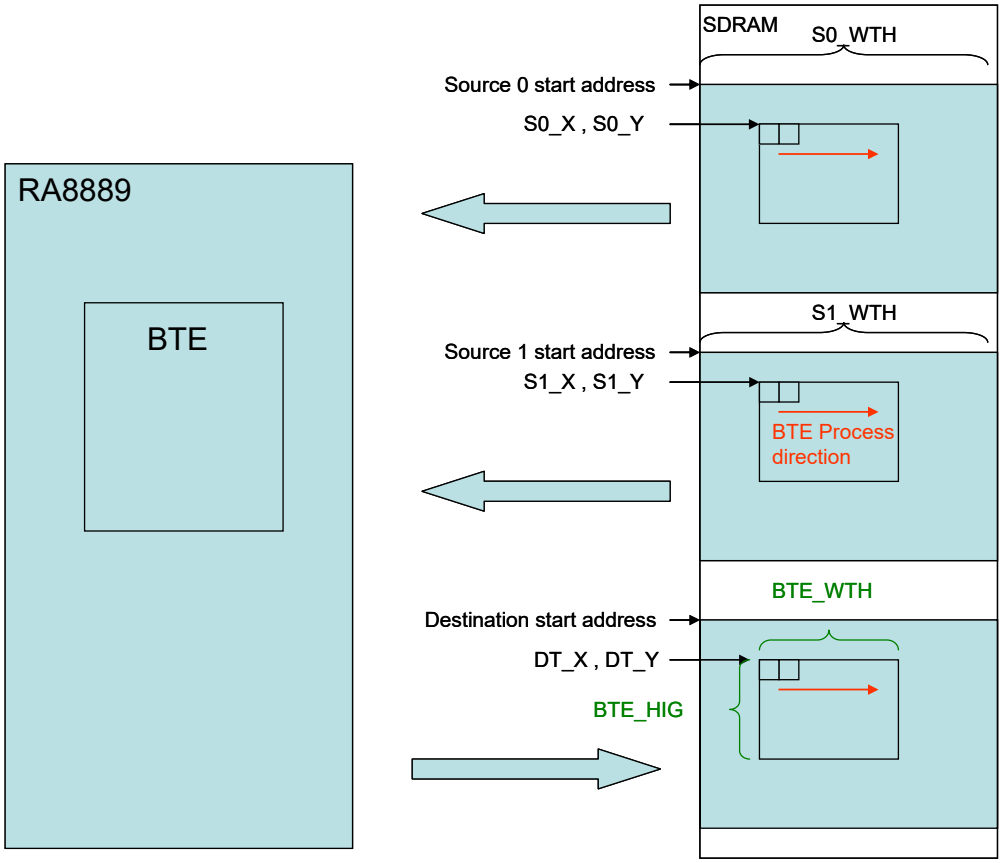


图4-1：Memory Access of BTE Function

4.1.1: Solid Fill

Solid Fill BTE功能提供使用者可将特定的区域 (BTE 來源区域)这个功能用在将。以特定颜色来填满SDRAM它的颜色是经由，里的一个大区块填满为同样的一个颜色BTE 。前景色来设定

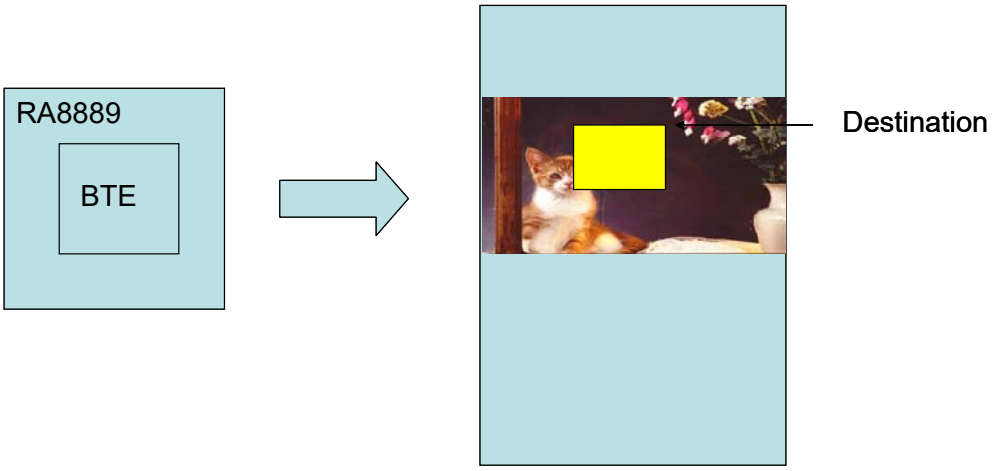


图4-2 : Hardware Data Flow

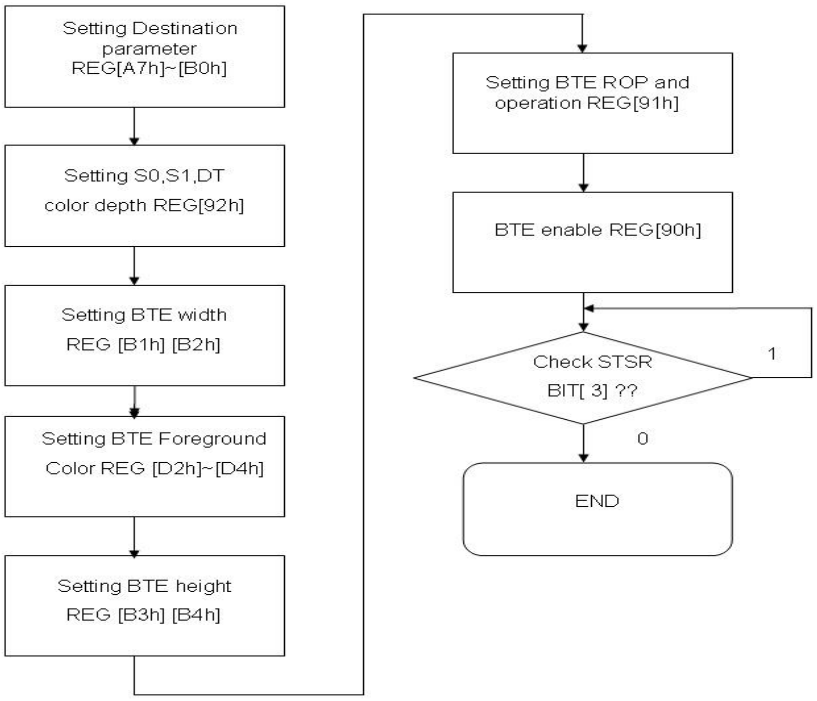


图4-3 : Flow Chart

4.1.2. BTE 的 Solid Fill 功能在 LCD 上的显示结果:

以下为 API 程式与范例说明，图 4-4 为使用 Solid Fill 功能填满一个红色的方形区块。

```
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
)
```

范例:

```
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
```

条件:

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

程式执行结果:

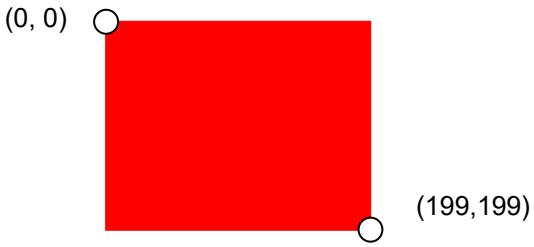


图 4-4：使用Solid Fill功能将[(0, 0) 到(199,199)]用红色填满

4.2.1:Pattern Fill with ROP

“Pattern Fill with ROP” 功能可设定一个在 SDRAM 中的特定方形记忆区块，并填入重覆的特定图形样板。图形样板是 8x8/16x16 的像素图形，存放在 SDRAM 中的非显示区的特定位置，图形样板并且可以配合 16 种光栅运算和目的地资料做邏辑运算。此操作可以用來加速某些需要在特定区域重覆贴入某一种图形，像是背景图案等应用。

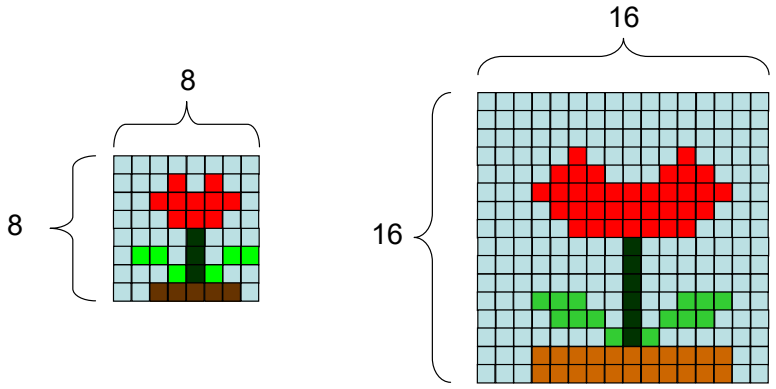


图4-5 : Pattern Format

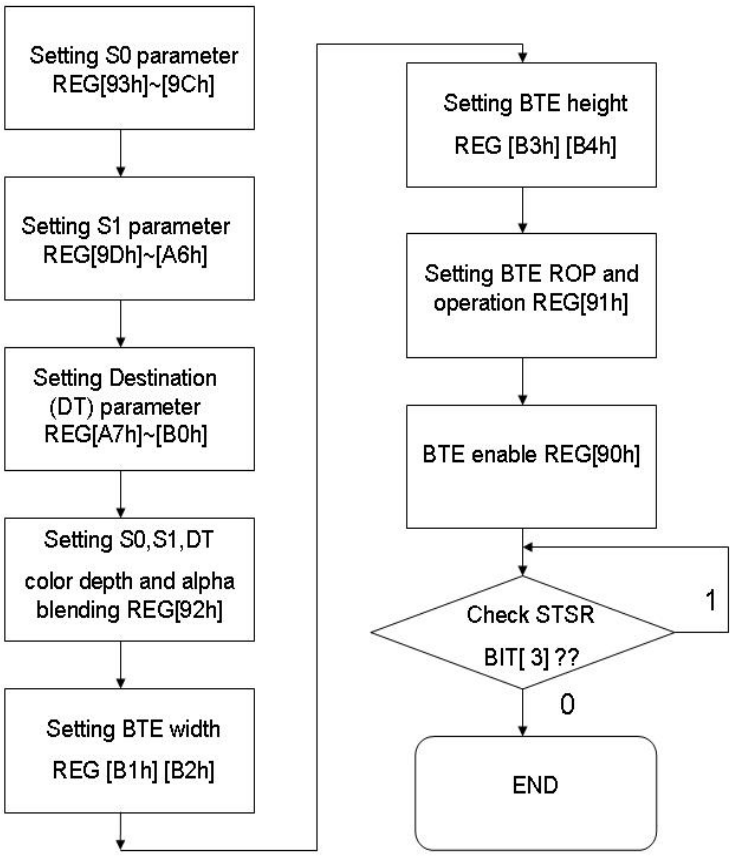


图 4-6 : 流程图

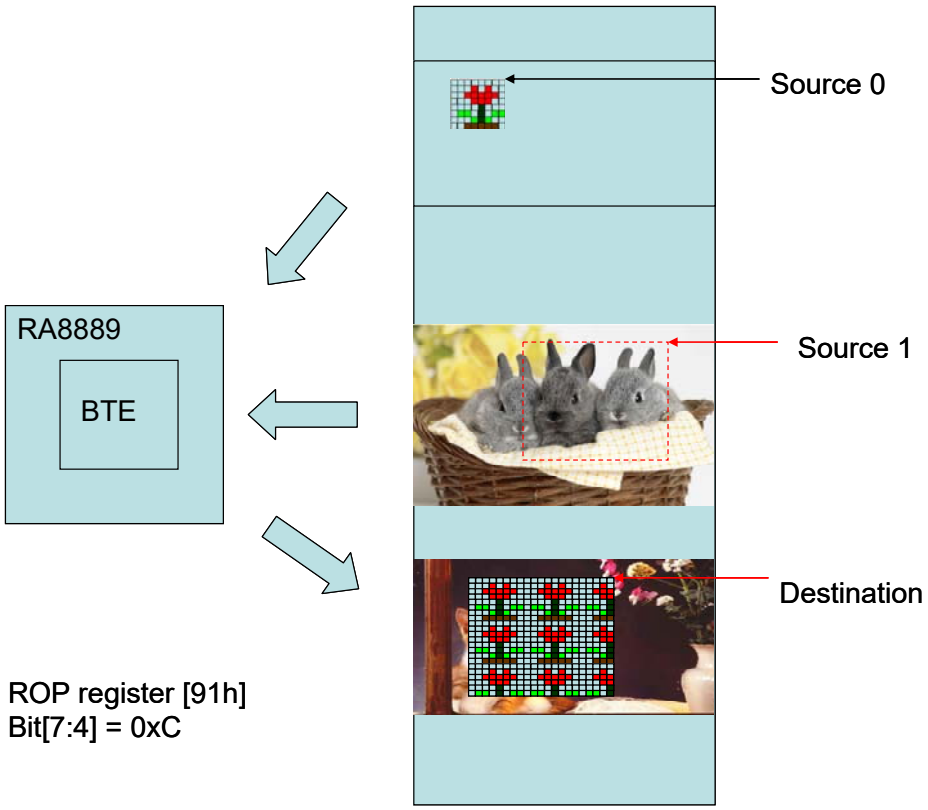

Source 0
Source 1
Destination
RA8889
BTE
ROP register [91h]
Bit[7:4] = 0xC

图 4-72：硬体资料流程

4.2.2. BTE 的 Pattern Fill 功能在 LCD 上的执行显示结果:

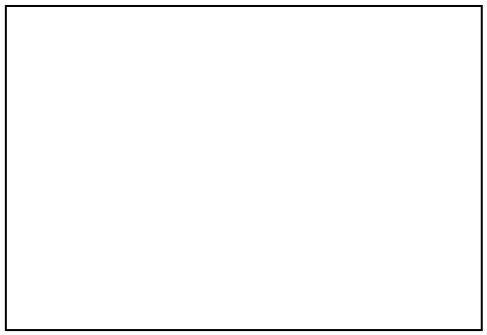


图 4-9 (16x16) 图形样版

图 4-8: 在这个范例中，SDRAM 目前的资料

API for pattern fill :

```
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
)
```

范例:

```
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);

条件:

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

步骤一: 执行 DMA 功能从外接 FLASH 读一张图写入到 SDRAM

(0, 0)

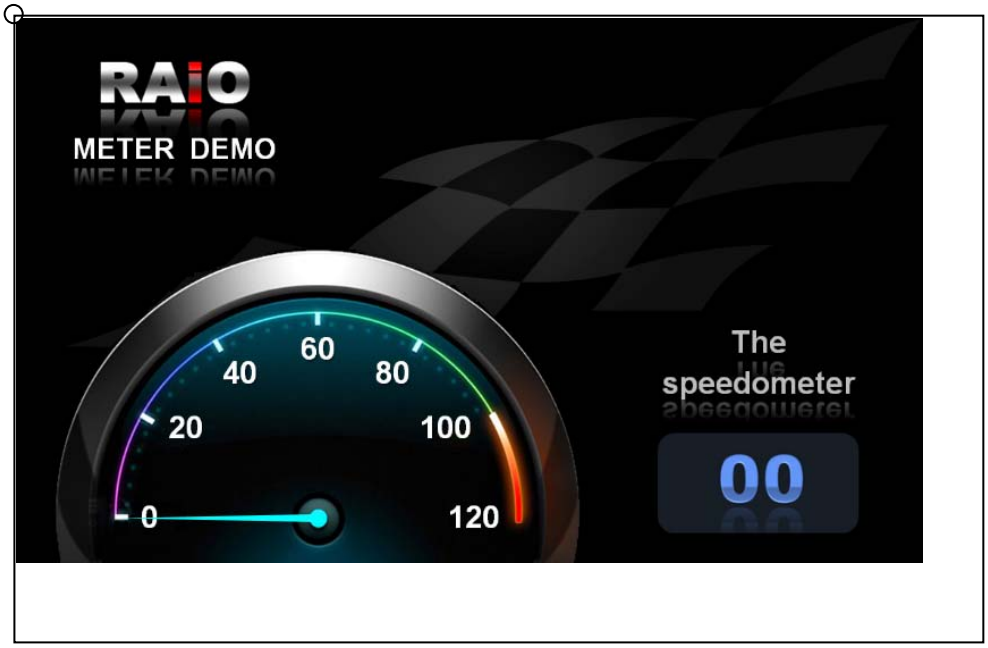


Figure 4-10: 使用 DMA 功能从外接 FLASH 读一张图写入到 SDRAM

.

步骤二:写入一个 **16x16 icon** 到记忆体(SDRAM)

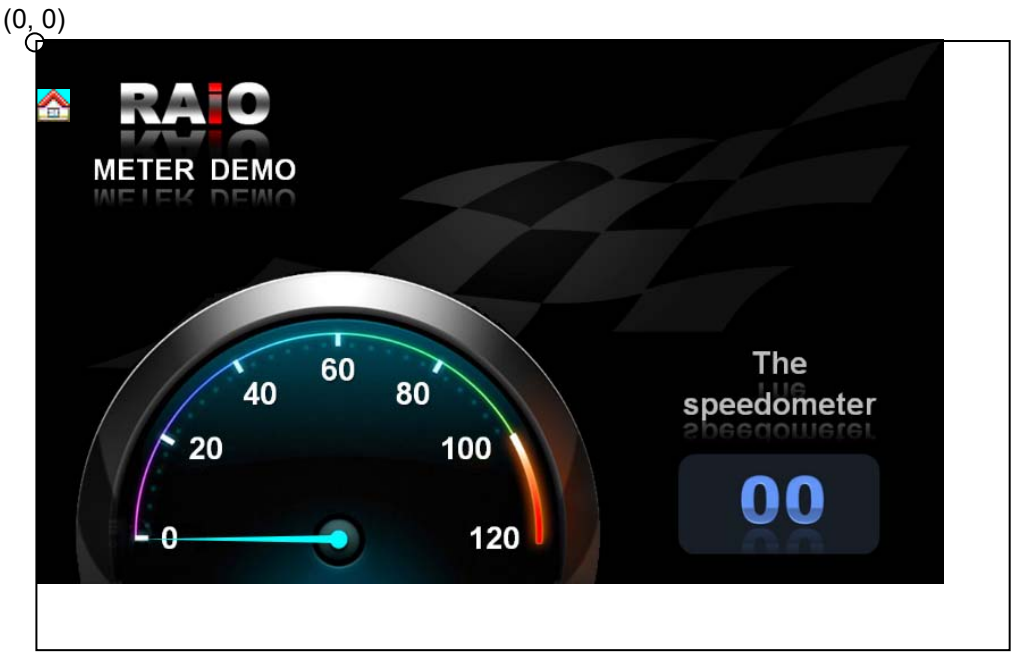


Figure 4-11: 写入一个 **16x16 icon** 到记忆体(SDRAM)

步骤三:执行 **Pattern fill** 功能

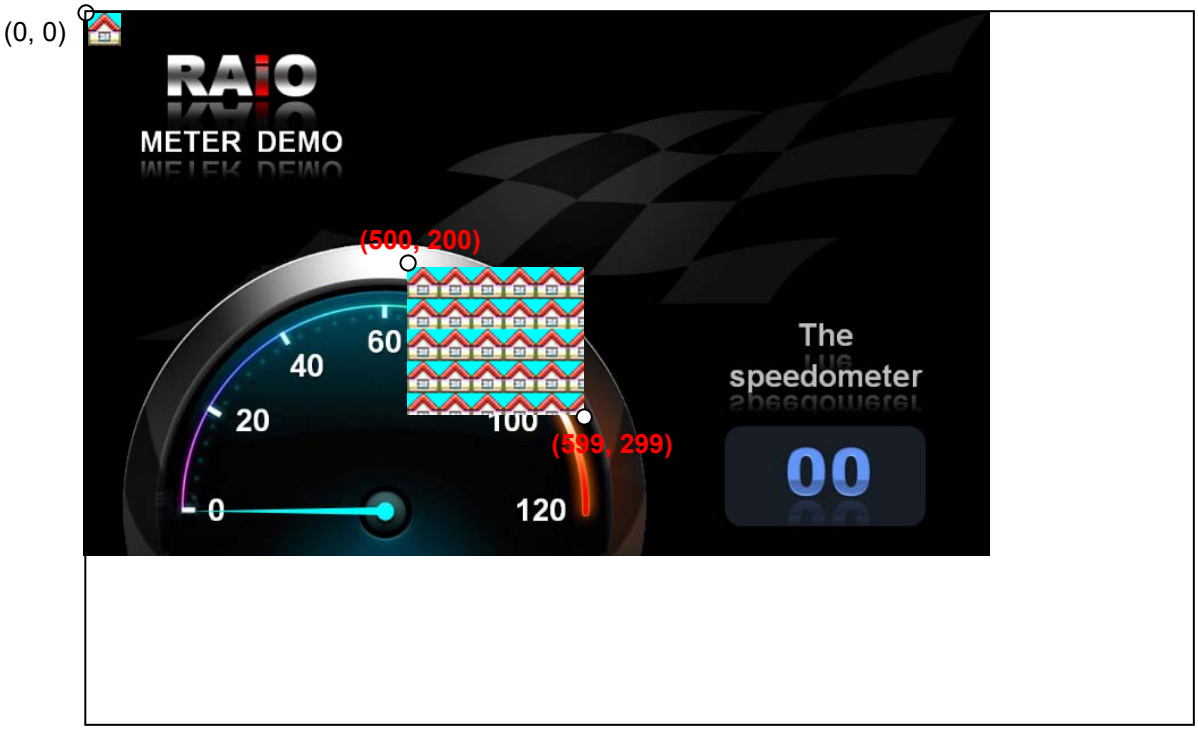


Figure 4-12: pattern fill 功能完成后的样子，来源范围从(0, 0)到(15, 15)，且目的地范围从(500,200)到 (599,299)

4.2.3:Pattern Fill with Chroma Key

“Pattern Fill with Chroma Key”功能可设定在一个 DDRAM 中的特定方形记忆体，并填入重覆的特定图形样板，此功能与「Pattern Fill with ROP」功能有相同的功能，不同的是，加入通透性的功能。也就是对於特定的「通透色」，此 BTE 功能会予以忽略。通透色是在 REG[D5h]~[D7h]中被设定，当图形样板中的颜色与通透色一致时，那部份的“目的地资料”将不会改变。

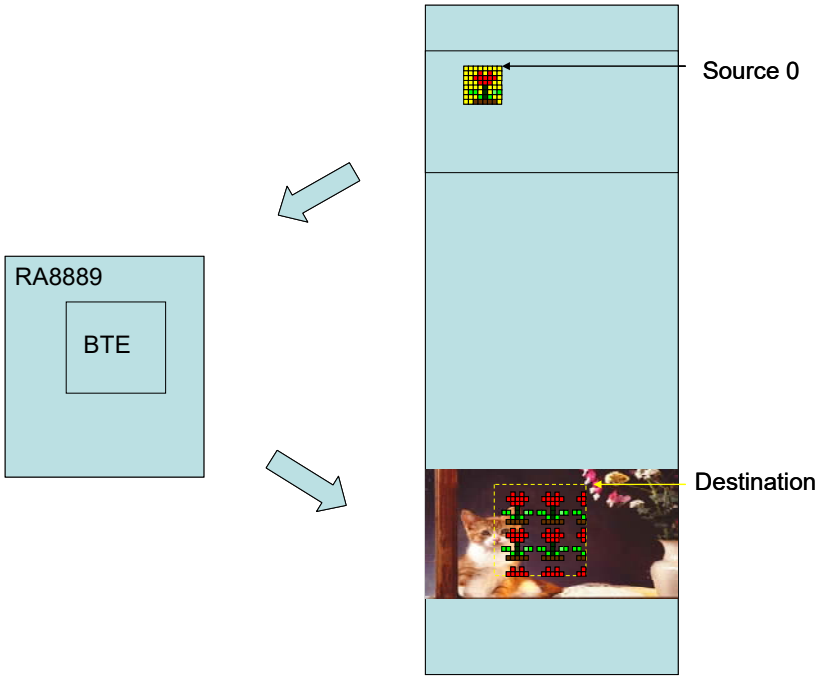


图 4-10：硬体资料流程

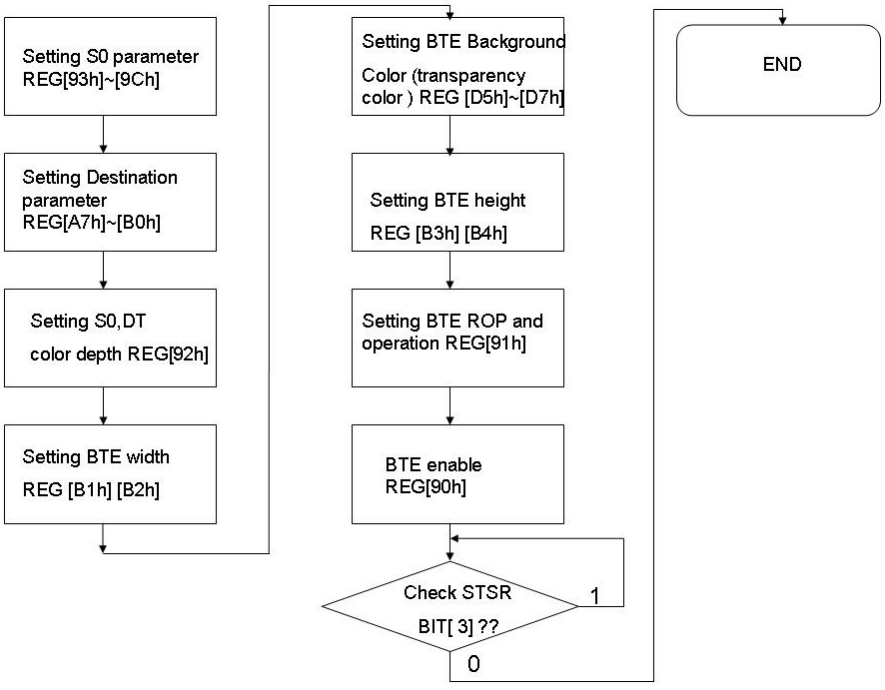


图 4-11：流程图

4.2.4: BTE Pattern Fill with Chroma key 功能在 LCD 上的显示结果:

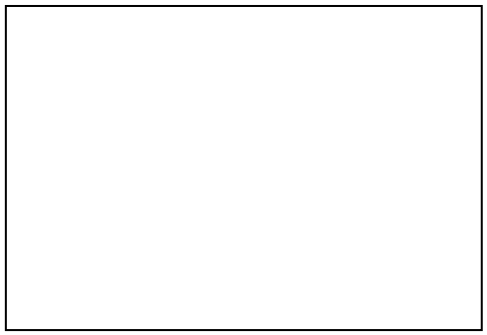


图 4-13 (16x16) 图形样版

图 4-12: 在这个范例中，SDRAM 目前的资料

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
)
```

范例:

```
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);
```

条件:

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)

步骤一：执行 DMA 功能从外接 FLASH 读一张图写入到 SDRAM

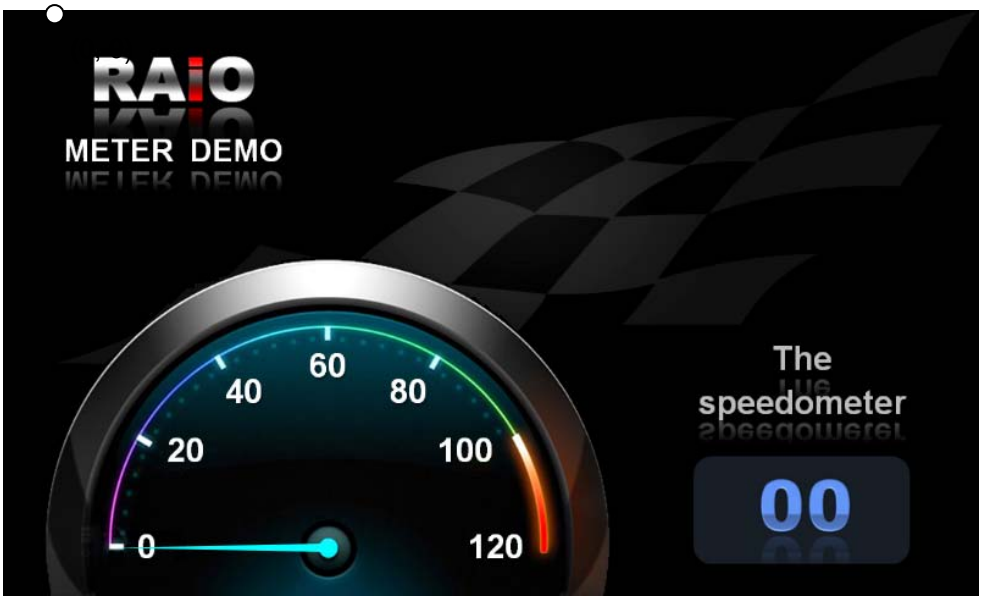


Figure 4-14: 执行 DMA 功能从外接 FLASH 读一张图写入到 SDRAM

步骤二：写入一个 16x16 icon 到记忆体(SDRAM)

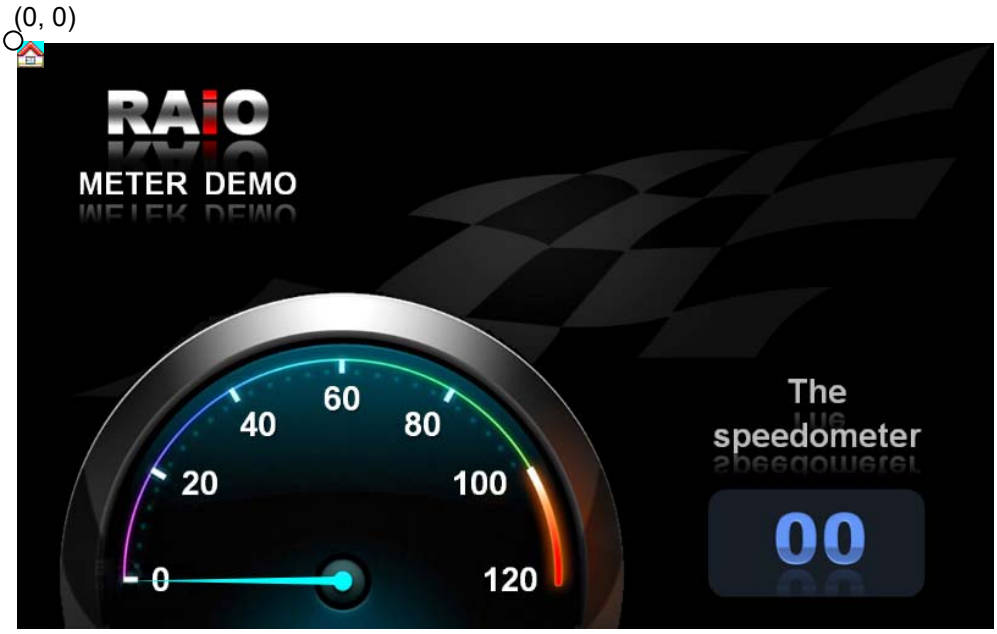


Figure 4-15: 写入一个 16x16 icon 到记忆体(SDRAM)

步骤三：执行 **Pattern fill with chroma key** 功能

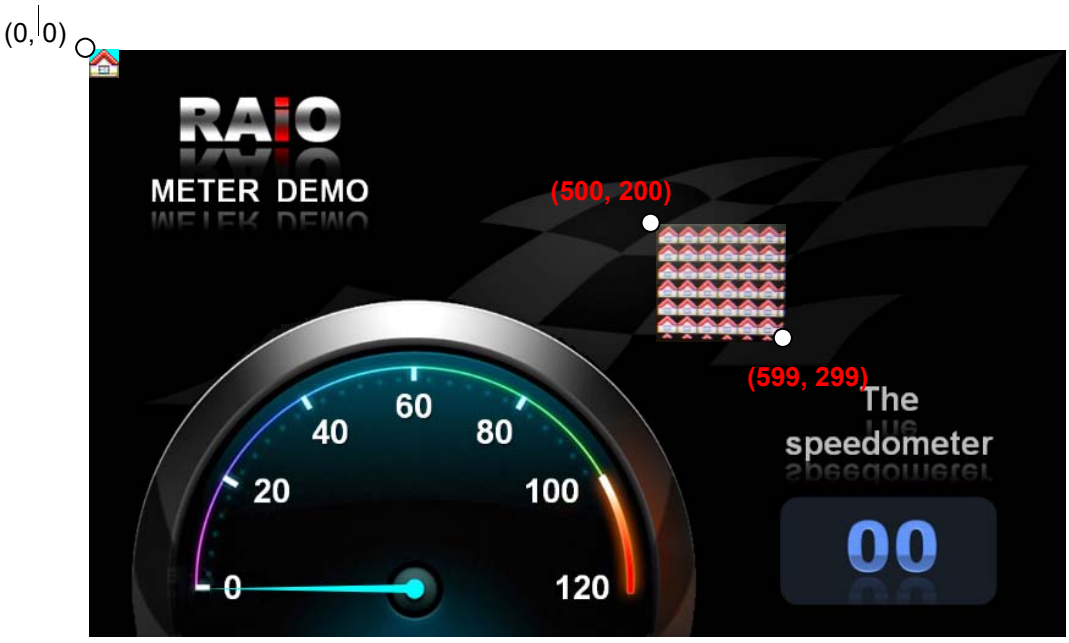


Figure 4-16: pattern fill with chroma key 功能完成后的样子，来源范围从(0, 0)到(15, 15)，且目的地范围从(500,200)到 (599,299)

4.3.1: MCU Write with ROP

MCU Write with ROP 的 BTE 功能，增加了 MCU 介面写入 SDRAM 的资料传送速度。透过 MCU，Write BTE 搭配光栅运算将资料填入特定的 SDRAM 区域，Write BTE 功能支援全部 16 种光栅运算。Write BTE 功能需要 MCU 端提供资料。

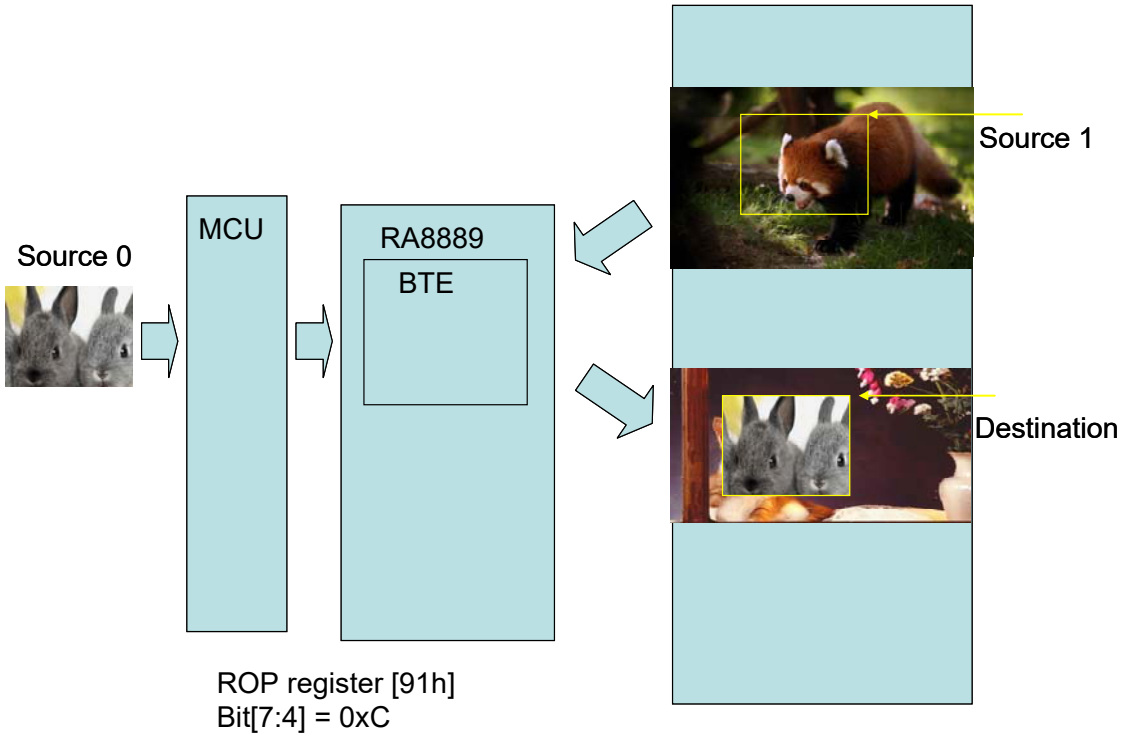


图 4-17：硬体资料流程

以下是建议的程式流程以及寄存器设定：

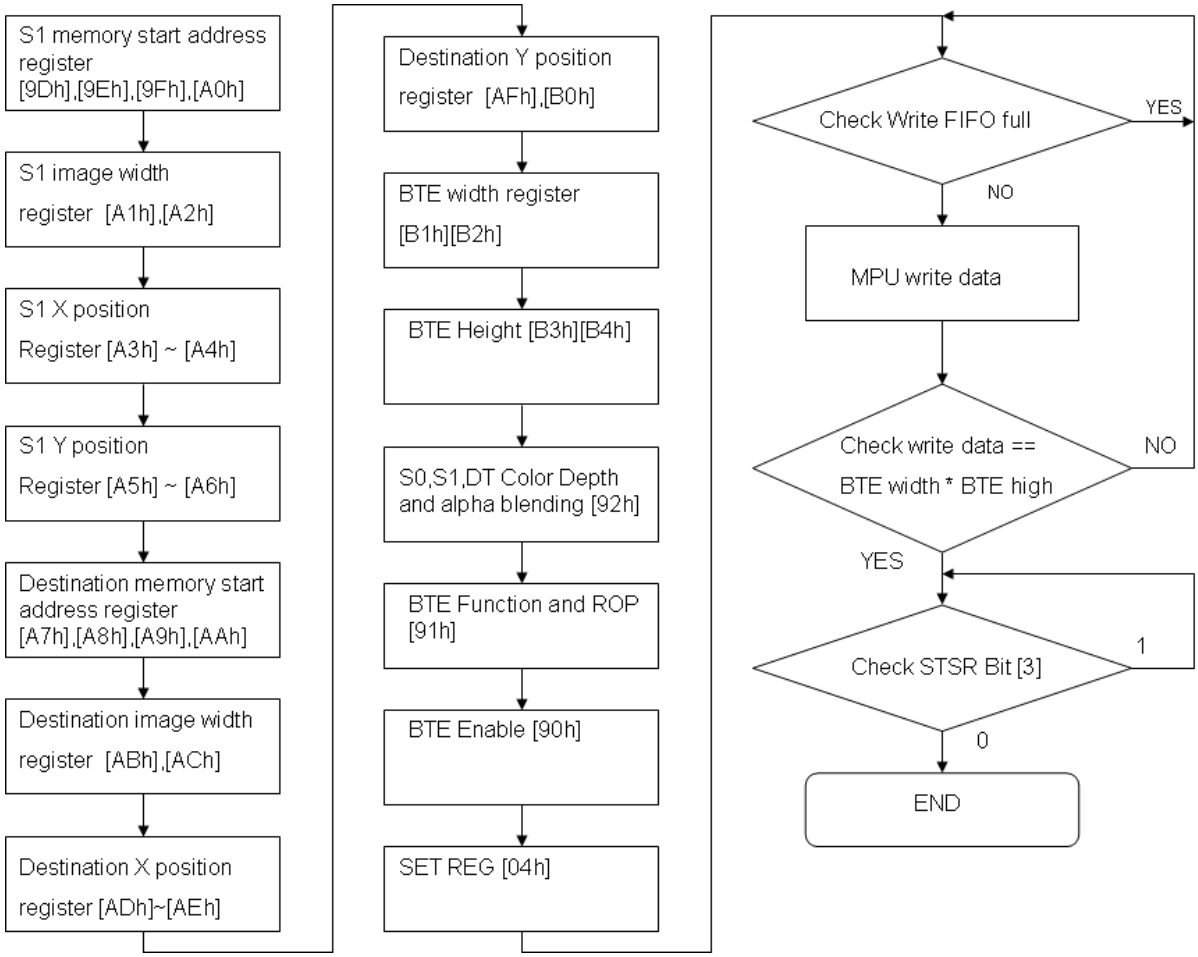


图 4-18：流程图

4.3.2: BTE MCU Write with ROP 功能在 LCD 上的显示结果:

在 BTE MCU Write with ROP 中，我们针对 MCU 8bit 和 16bit 各提供一组 API 供使用者使用，图 4-19 为 SDRAM 的资料，图 4-20 为 MCU 端的图资，使用 BTE MCU Write with ROP 功能，搭配上 ROP 参数，可以将 MCU 端输入的图资做逻辑运算後写入 SDRAM。以下为 API 程式与范例解说:

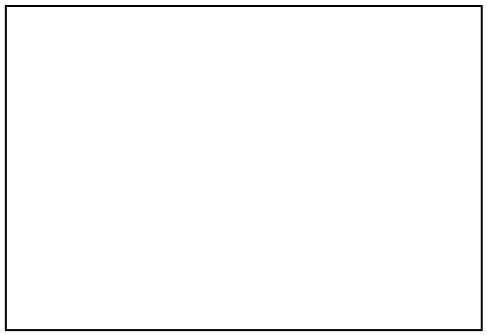


图 4-19: 在这个范例中，SDRAM 目前的资料

图 4-20：MCU 写入的资料(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
)
```

范例:

```
//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);
```

Condition:

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 .

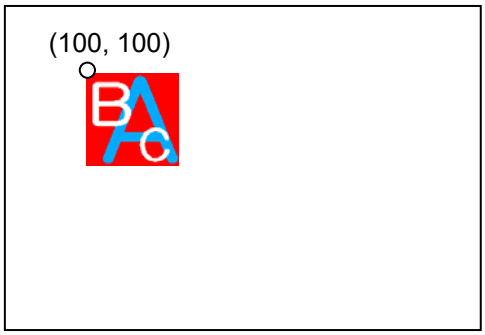


图 4-21: 用 BTE MCU Write 将图片写入 SDRAM 中，目的地位置从(100,100)到(227,227)

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

BTE 通透性写入功能(MCU Write With Chroma Key),可以加增加 MCU 端写入显示资料至 SDRAM 的传送速度,一旦 BTE 通透性写入功能开始,BTE 引擎会持续动作直到所有的像素都被写入为止。

BTE 通透性写入功能可用來更新一个 SDRAM 的特定区域,而由 MCU 提供显示资料來源,不同於前面章节提到的 MCU Write with ROP 的 BTE 功能,BTE 通透性写入功能会忽略某些特定颜色的操作,此特定的通透色可由使用者设定,在 RA8889 中,此特定通透色设定於寄存器中的「BTE 背景色」中,当讀到來源资料的颜色,为符合通透色设定时,RA8889 便会忽略这部份的显示资料,不执行写入的功能。此功能在处理将一张图片的部分图形复制到 SDRAM 时很有帮助,不需被复制的地方,在來源图片中便以「通透色」來处理,在 BTE 通透性写入功能执行时,便不会进入被写入 SDRAM。利用此功能可以很快的在任意背景图上,写入一个前景图案。如图 4-22 为例,來源图案为一个红色的背景搭配黄色的圆形图案,藉着设定红色为「通透色」,并且执行 BTE 通透性写入功能,就相当於将一个黄色的圆形图案,贴到目的地位置的功能。BTE 通透性写入功能在來源和目的资料设定皆支援“线性”和“区块”定址模式。

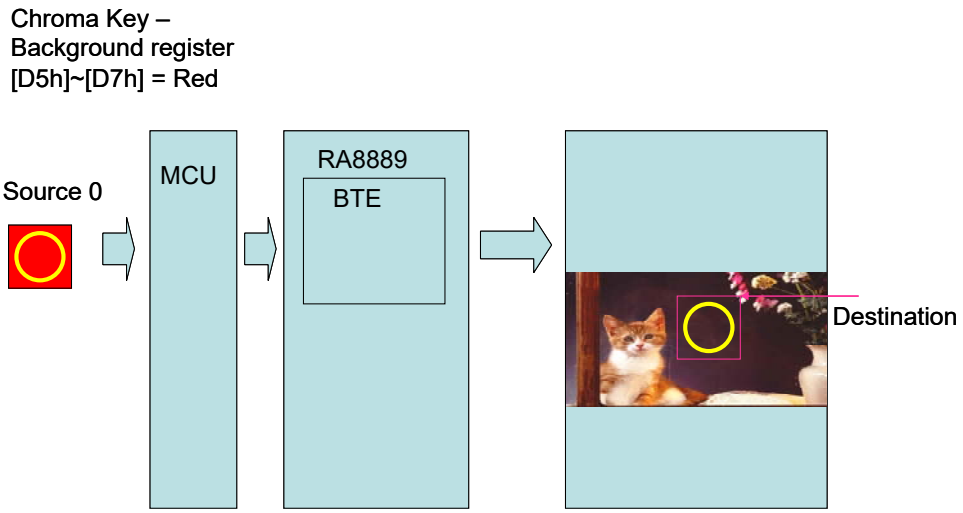


图 4-22：硬体资料流程

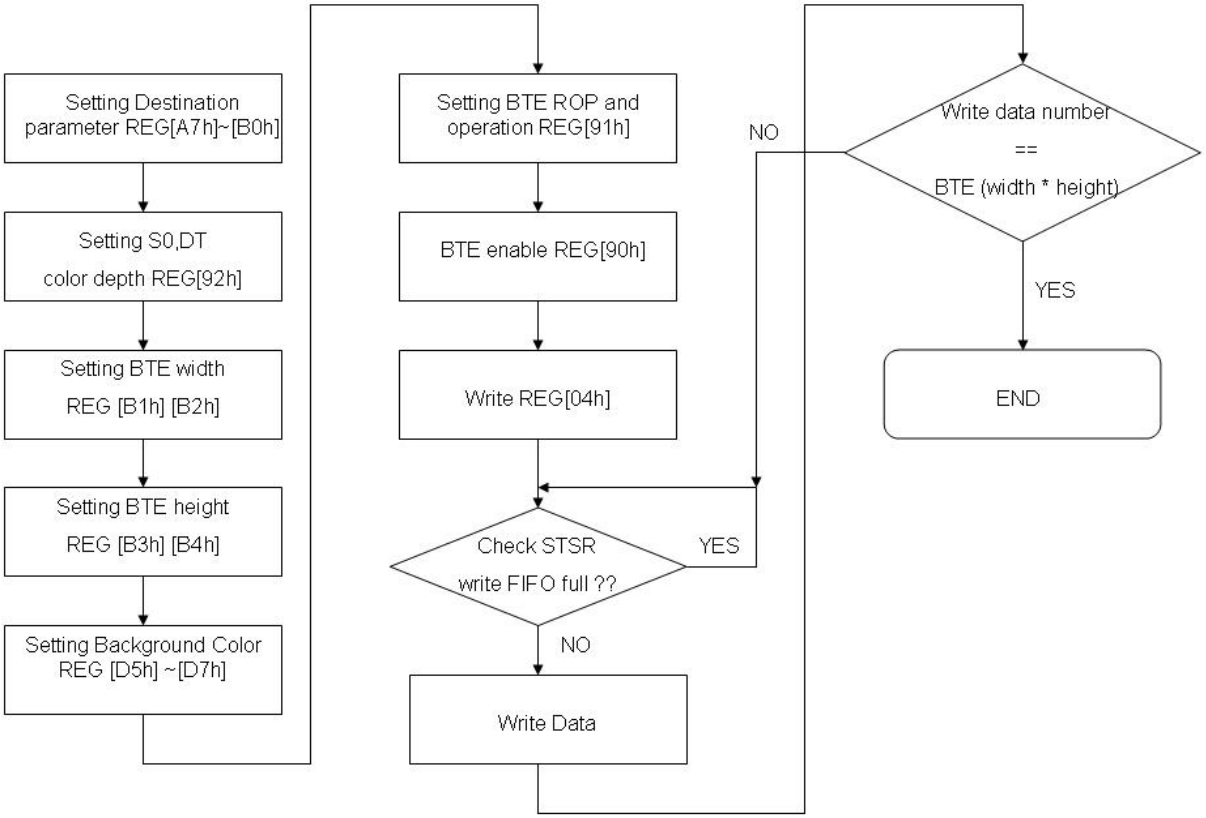

Setting Destination parameter REG[A7h]~[B0h]
Setting S0,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting Background Color REG [D5h] ~[D7h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Write REG[04h]
Check STSR write FIFO full ??
YES
NO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

图 4-23：流程图

4.3.4:BTE MCU Write With Chroma Key 功能在 LCD 上的显示结果:

BTE 通透性写入功能，可以增加 MCU 端写入显示资料至 SDRAM 的传送速度，一旦 BTE 通透性写入功能开始，BTE 引擎会持续动作直到所有的像素都被写入为止。

不同於前面章节提到的 MCU Write with ROP 的 BTE 功能，MCU Write with chroma key 将忽略 MCU 设定的通透色，此特定通透色设定於寄存器中的「BTE 背景色」中，当讀到來源资料的颜色为被设定的通透色时，RA8889 便会忽略这部份的显示资料，不执行写入的功能。举例来说，如图 4-25，来源图片有一个蓝色字母 A 与两个白色字母 B 与 C 在搭配上红色背景。经由设定红色为通透色，然後使用 MCU Write with chroma key 功能，就会将背景红色滤掉，只剩下蓝、白色字母的资料。

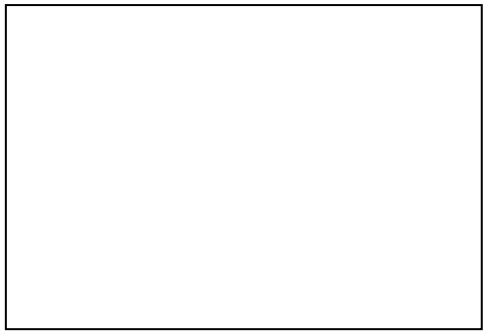


图 4-24: 在这个范例中，SDRAM 目前的资料

图 4-25 :MCU 写入的资料(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
)
```

范例:

```
//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.

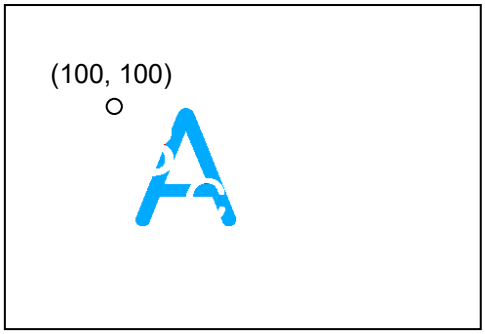


图 **4-26: BTE MCU Write With Choma Key**，通透色为红色，目的地位置从**(100,100)**到**(227,227)**。

4.4.1: Memory Copy with ROP

Memory Copy with ROP 的 BTE 功能，是将 SDRAM 的一个特定区块的资料，搬移到另外一个区块。这个功能可以加速一个区块的资料复制，而且能省下很多 MCU 端的执行时间以及负担。

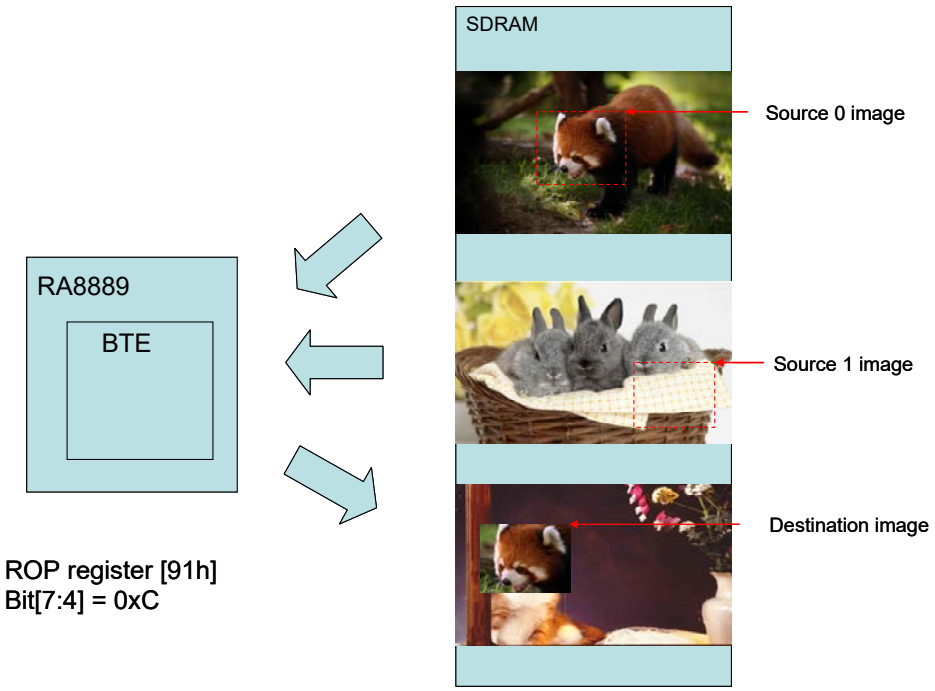


图 4-27：硬体资料流程

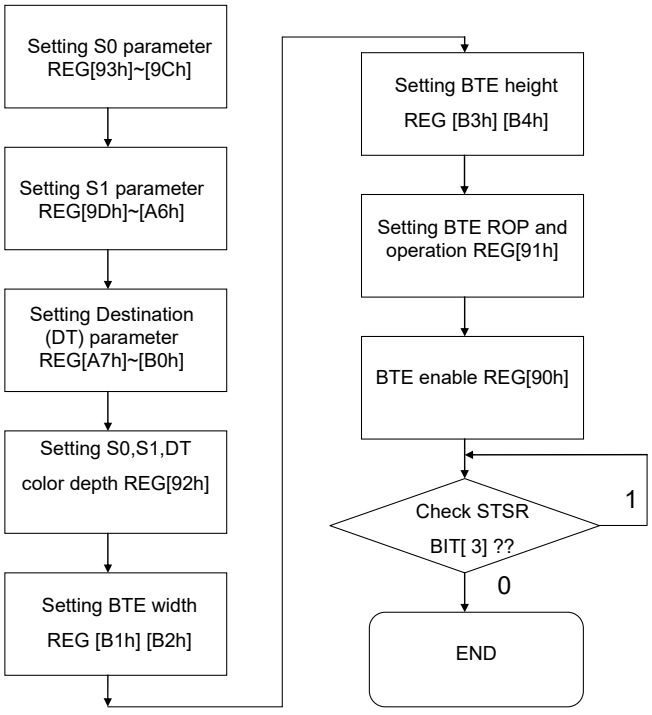


图 4-28：程图流

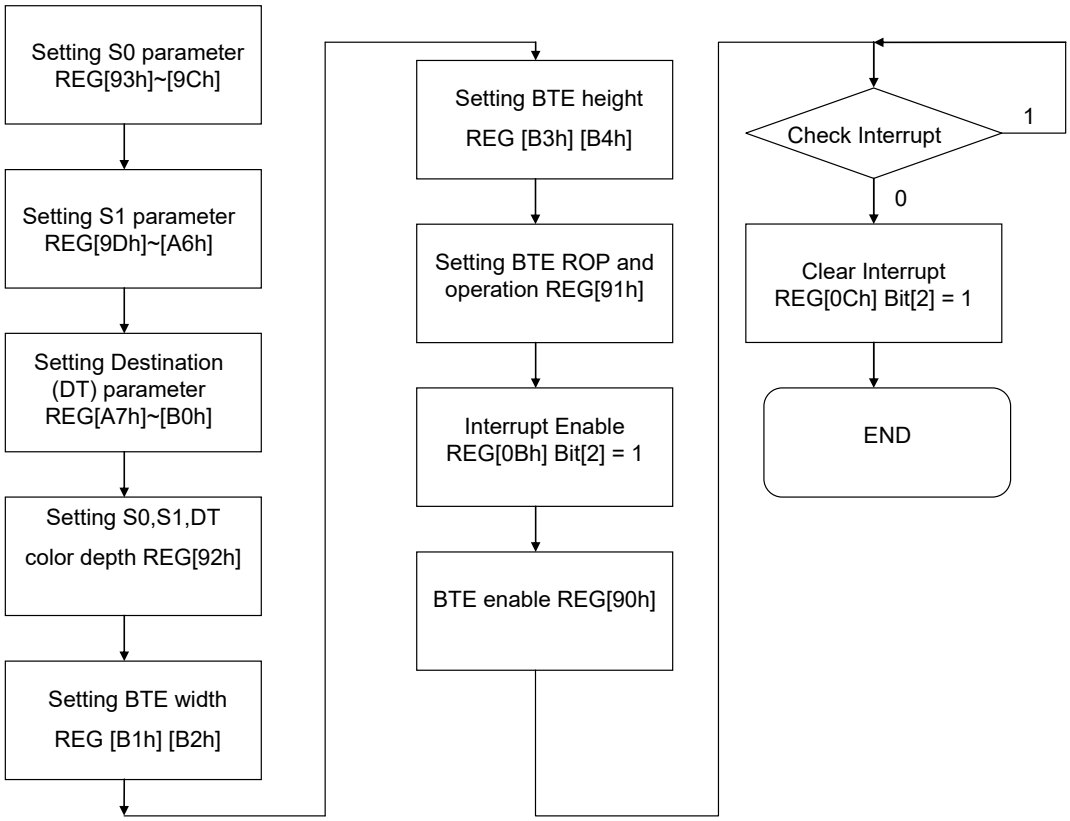


图 4-29：流程图 — Check Int

4.4.2: BTE Memory Copy 功能在 LCD 上的显示结果:

以下为 BTE Memory Copy API 功能的解说与范例，先利用了 BTE Solid Fill 功能画出一块填满红色的方形，以及用画圆功能画出一个黄色的实心圆(如图 4-30)，再透过 BTE Memory Copy 复制一个一样的图案(如图 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
)
```

范例:

```
/*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);
```

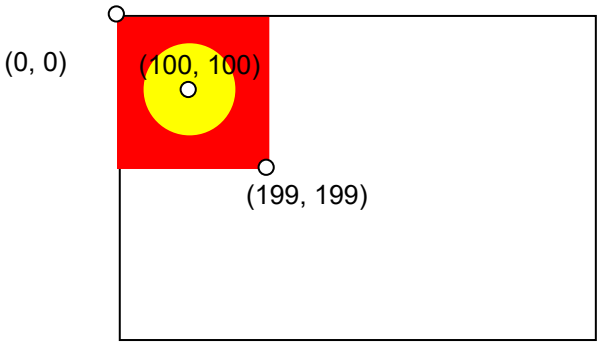


图 4-30: 绘一黄色的实心圆与红色实心矩形

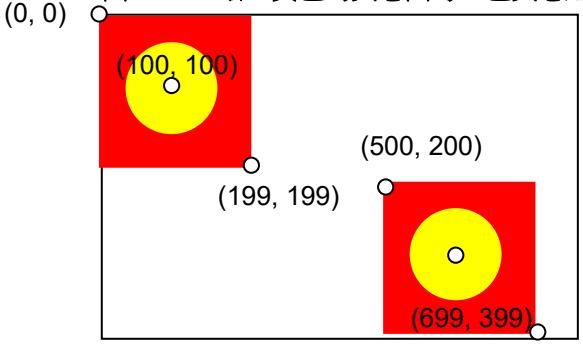


图 4-31：使用 BTE Memory Copy with ROP 功能复制一个一样的图案。

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

Memory Copy With Chroma Key (w/o ROP)的 BTE 功能，是将 SDRAM 的一个特定区块的资料，搬移到另外一个区块，而且滤掉特定的通透色。而通透色设定是在背景色寄存器。当通透色与被复制的资料是一致时，RA8889 便会忽略这部份的显示资料，不执行写入的功能，故该目的地的区块资料不会有所改变。在这个功能下，来源 0 、来源 1 以及目的地的资料区块都是在 DDRAM 里面。

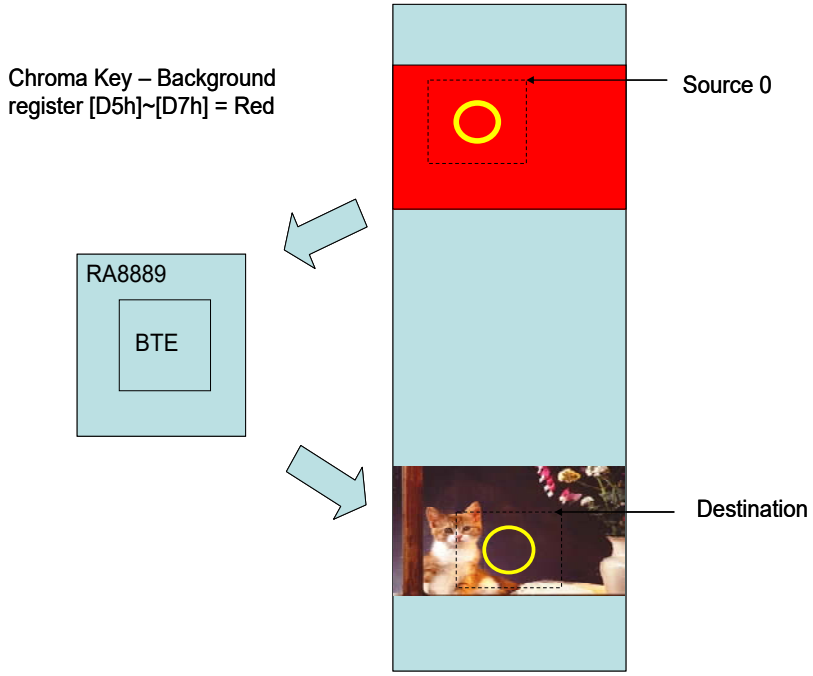


图 4-32：硬体资料流程

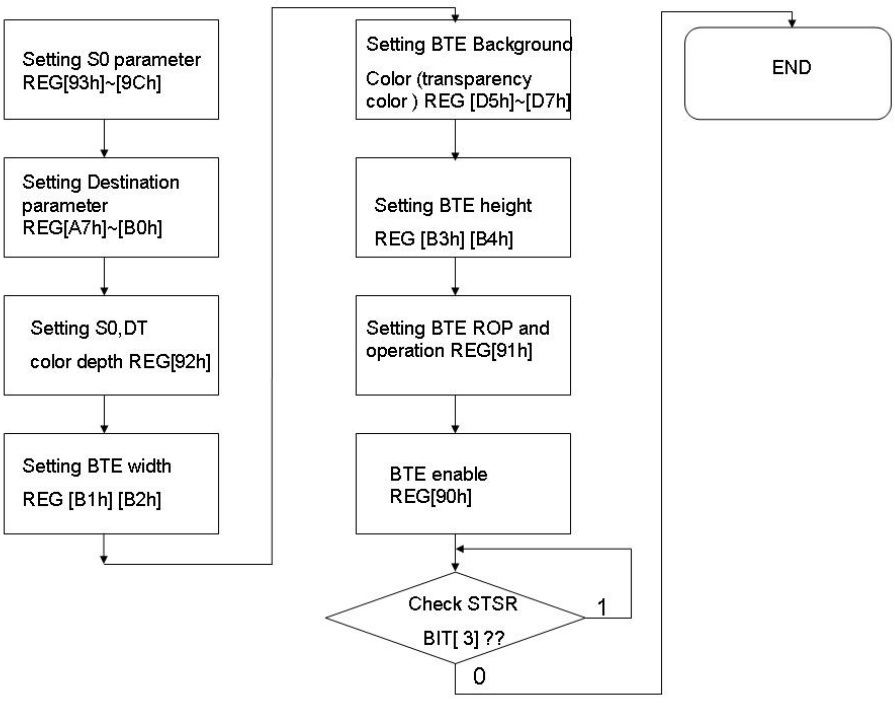


图 4-33：流程图

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

以下为 BTE Memory Copy with Chroma key(w/o ROP) API 功能的解说与范例，先利用了 BTE Solid Fill 功能画出一块填满红色的方形，以及用画圆功能画出一个黄色的实心圆(如图 4-34)，再透过 BTE Memory Copy with Chroma key(w/o ROP)复制一个将红色部分滤掉的图案(如图 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
)
```

条件:

```
/*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,500,200,0xff0
000,200,200);
```

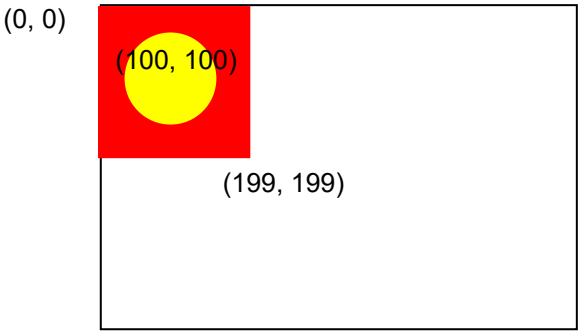


图 **4-34:** 绘一黄色的实心圆与红色实心矩形

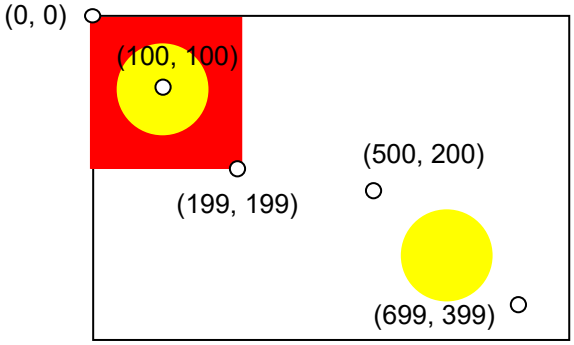


图 **4-35:** 使用 **BTE Memory Copy with chroma key** 功能复制黄色的实心圆，并过滤掉红色实心矩形的部分。

4.5.1: MCU Write With Color Expansion

MCU Write with Color Expansion 是一个很有用的功能，用來处理 MCU 的单色图形资料转换为彩色图形资料，并写入 SDRAM 中。此功能的來源资料为 MCU 提供的单色图形资料 (Monochromes Bitmap)。而每一个位元根据内容被转换为 BTE 前景色或背景色。若单色资料來源为”1”则会被转换为 BTE 前景色，若为”0”则会转换为 BTE 背景色。此功能可以大大降低将单色系统资料转换为彩色系统资料的成本。颜色扩充功能会根据 MCU 的资料汇流排宽度，持续讀入 16 位元或 8 位元的资料做转换，并且可以位元为单位，设定每一行的第一笔单色图形资料的起始转换位元，并且在每一行的最後一笔资料讀入後，超过范围的位元也会被忽略而不写入，而下一行则从下一笔资料开始执行同样的操作。这样以位元为单位的运算大大增加此功能的弹性。另外，每一笔资料的处理方向是从最高位元 (MBS) 至最低位元 (LSB)。

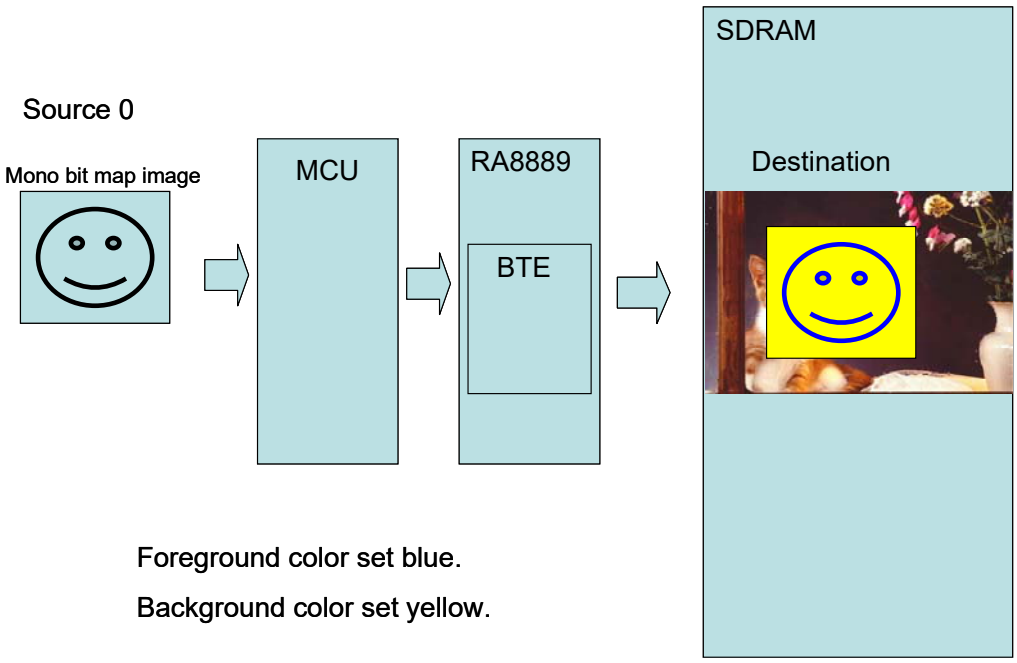


图 4-36：硬体资料流程

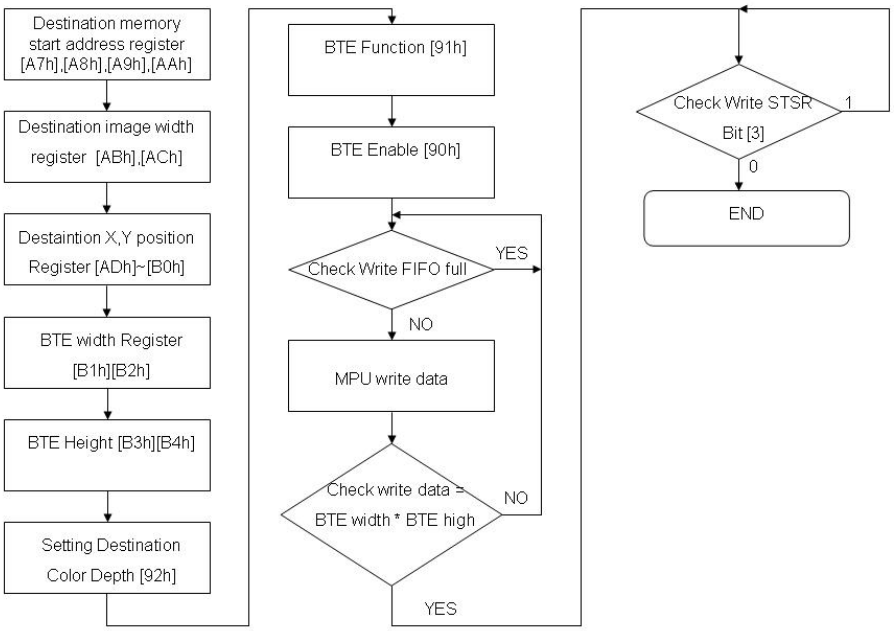


图 4-37：流程图

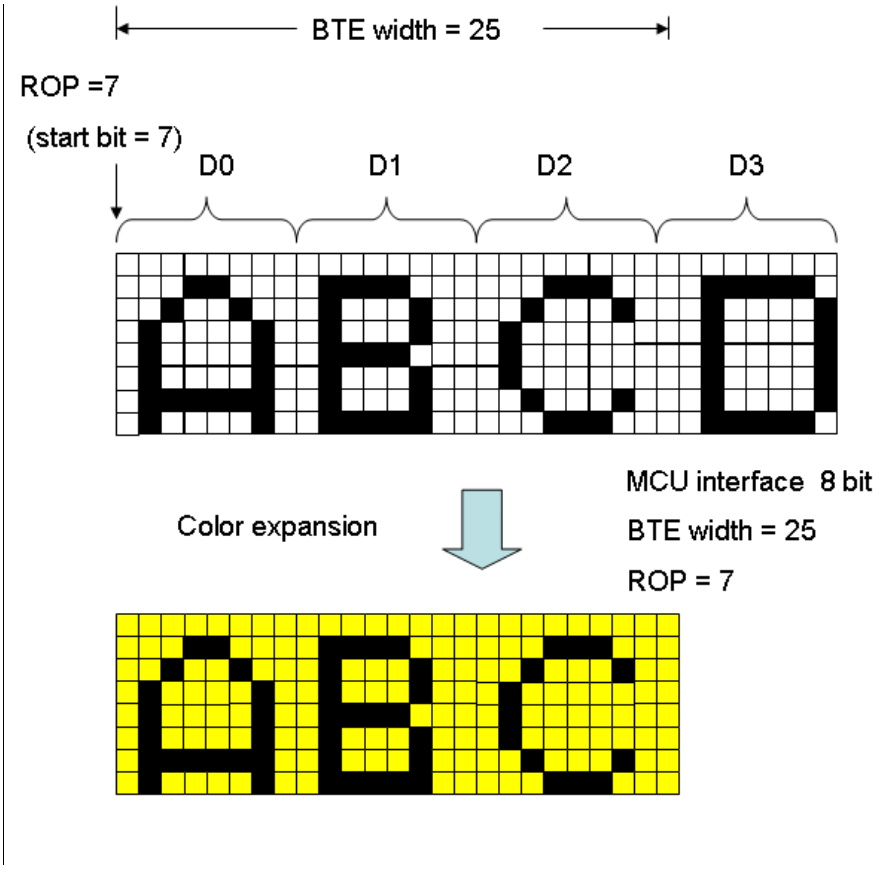


图 4-38：起始位元范例一

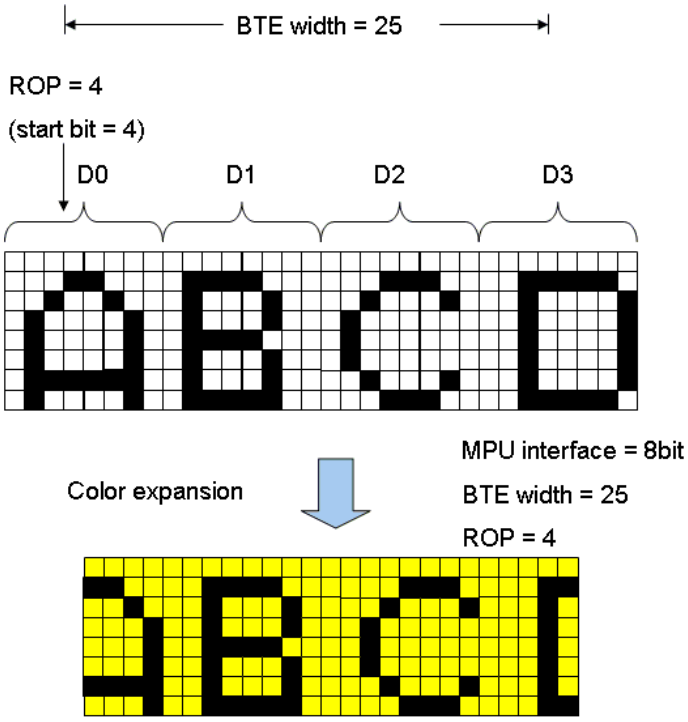


图 4-39：起始位元范例二

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

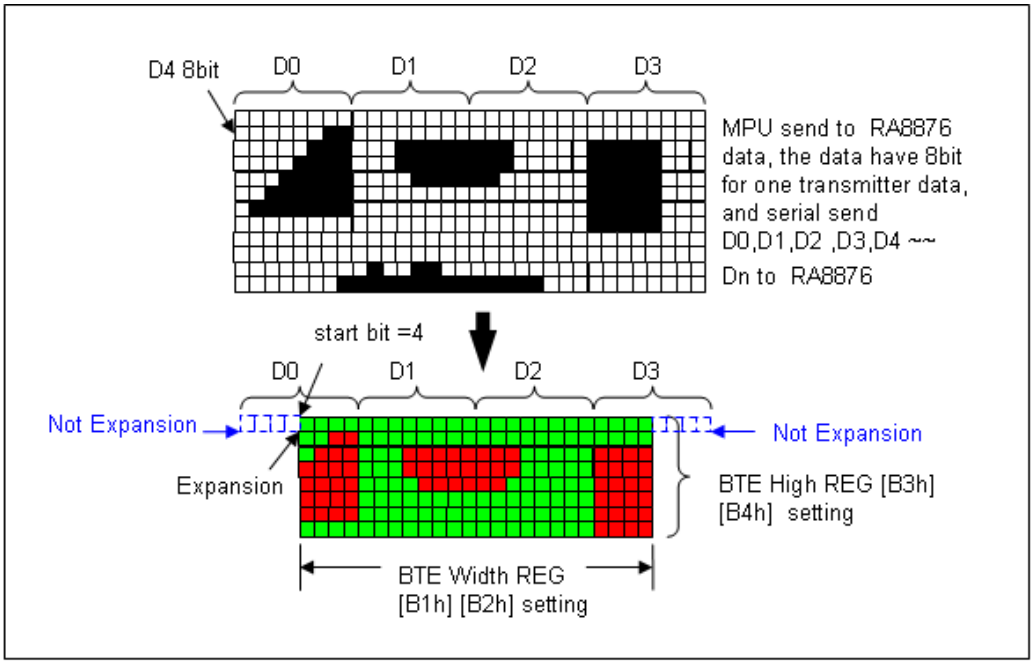


图 4 40：Color Expansion 资料图

4.5.2: BTE MCU Write Color Expansion 在 LCD 上的显示说明:

图 4-41 为 128x128 的单色黑白图像，假设前景色设定为绿色，背景色设定为蓝色，使用了 BTE MCU with Write Color Expansion 功能後，就会转出像图 4-42 的图案一样。以下我们针对 MCU 8bit 与 16bit 各提供一组 API 以及说明与范例供使用者参考。

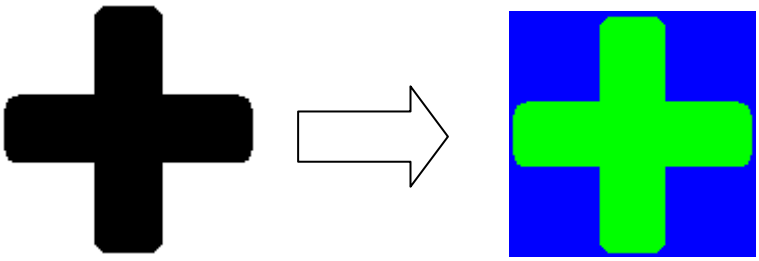


图 4-41：
128x128 黑白单色图资

图 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
)
```

范例:

```
/*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 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,gIma
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 上的显示画面:

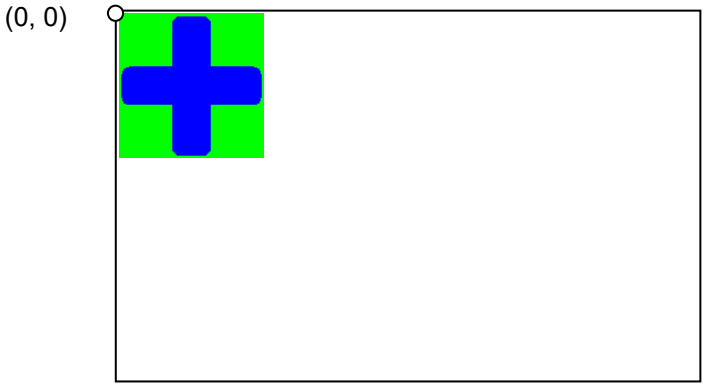


图 4-43 : BTE MCU Write with Color Expansion

4.5.3: MCU Write with Color Expansion and Chroma key

除了将 BTE 的背景色忽略，用来当作通透色，此 BTE 功能与 BTE MCU Write Color Expansion 功能几乎是相同的。就是所有输入单色资料值为”1”的位元将会被转换为 BTE 的前景色并且写入目的位置，所有输入单色资料值为”0”的位元将不被转换，而保持原來的目的资料颜色值。

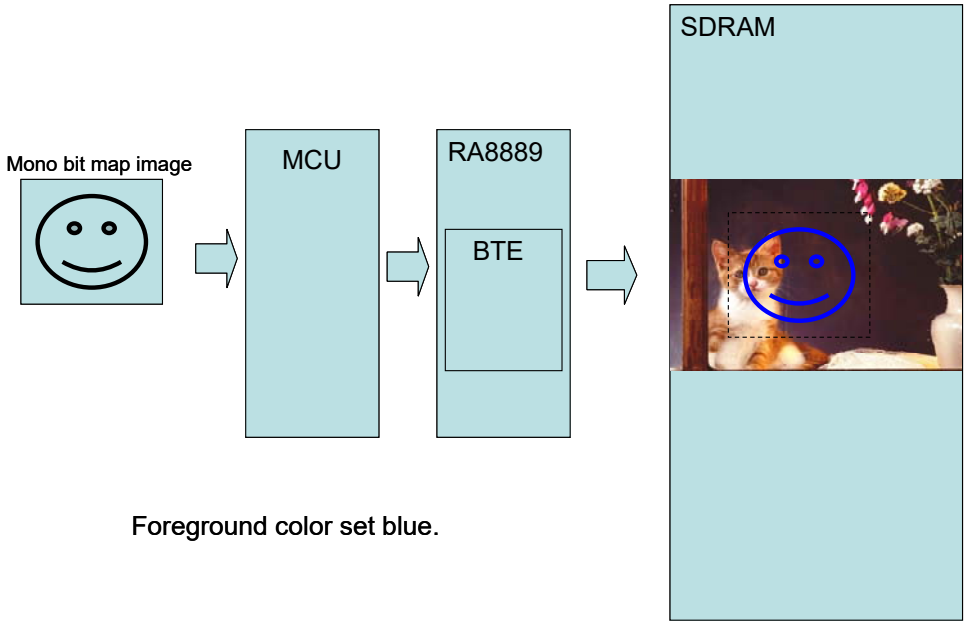


图 4-44：硬体资料流程

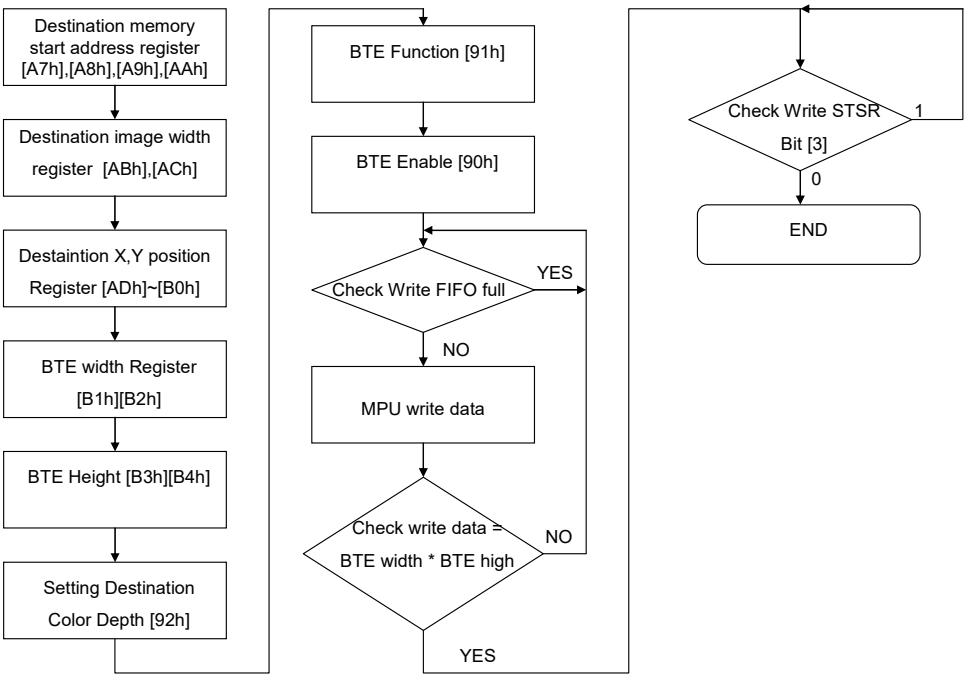


图 4 45：流程图

4.5.4: BTE MCU Write With Color Expansion and Chroma key 的 API 在 LCD 上的显示说明:

图 4-46 为范例所用到的 128x128 黑白的单色图，透过 BTE MCU Write With Color Expansion and Chroma key 功能，可以转换成其他颜色的图案，图 4-47 就是将图 4-46 使用了此功能後将原本的图转换成绿色。以下我们也分别针对 MCU8bit 与 16bit 提供了两组 API 以及范例说明。

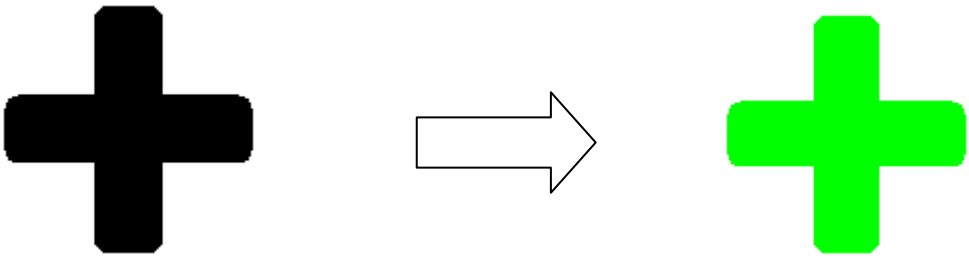


图 4-46 :
128x128 黑白单色图资

图 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
)
```

范例:

```
/*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);

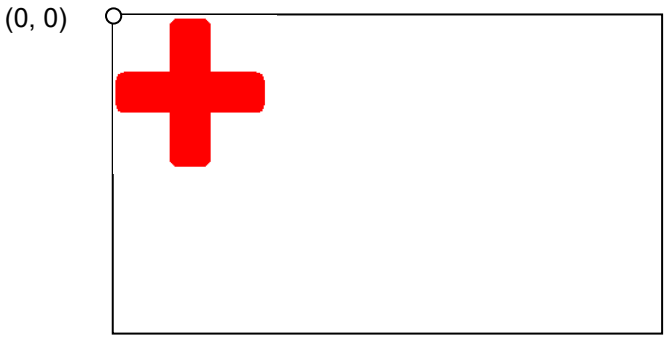


图 4-48 : BTE MCU Write With Color Expansion and Chroma key

4.6 :BTE Alpha Blending

概要:

在许多显示功能中，alpha blending 是一张图像与其他层的显示资料在画面上做出半透明的效果。RA8889 也提供了硬体做 Alpha Blending 的功能，使用者不用花费很多的系统资源，就可以得到更绚丽的显示效果。这份应用说明书将帮助我们的使用者，去了解并使用 RA8889 的 Alpha Blending 功能。

4.6.1: Memory Write with Alpha Blending

“Memory Copy with opacity” 可以混合来源 0 资料与来源 1 资料然后再写入目的记忆体。这个功能有两个模式 – Picture 模式与 Pixel 模式。 Picture 模式可以被操作在 8 bpp/16bpp/32bpp 色深下并且对于全图只具有一种混合透明度 (alpha level)，混合度被定义在 REG[B5h]。 Pixel 模式只能被操作在来源 1 端是 8bpp/16bpp 模式，而各个 Pixel 具有其各自的混合度，在来源 1 为 16bpp 色深下像素的 bit [15:12] 是透明度(alpha level)，剩余的 bit 则为色彩资料；而来源 1 为 8bpp 色深情形下像素 bit [7:6] 是透明度(alpha level)， Bit [5:0] 则是被使用在索引调色盘(palette color) 的颜色。根据 32bpp 色深下的像素图像，必须将 S1 颜色深度设置为 16bpp，并且必须将 S1 宽度设置为与原始图像相同的宽度（宽度）。在 32-bit 像素模式下，S1 图像的 bit [31:24]代表其 alpha 值，bit [23：0]代表像素数据。 Figure 13 31 显示了有关如何通过 MPU 接口将 RGB 图像数据写入 SDRAM 的流程。

Picture mode 的目的地资料 = (来源 0* (1- alpha Level)l) + (来源 1 * alpha Level)

Pixel mode 16bpp 下的目的地资料 = (来源 0 * (1 - alpha Level)) + (来源 1 [11:0] * alpha Level)

Pixel mode 8bpp 下的目的地资料 = (来源 0 * (1 - alpha Level)) + (调色盘索引 (Source 1[5:0]) * alpha Level)

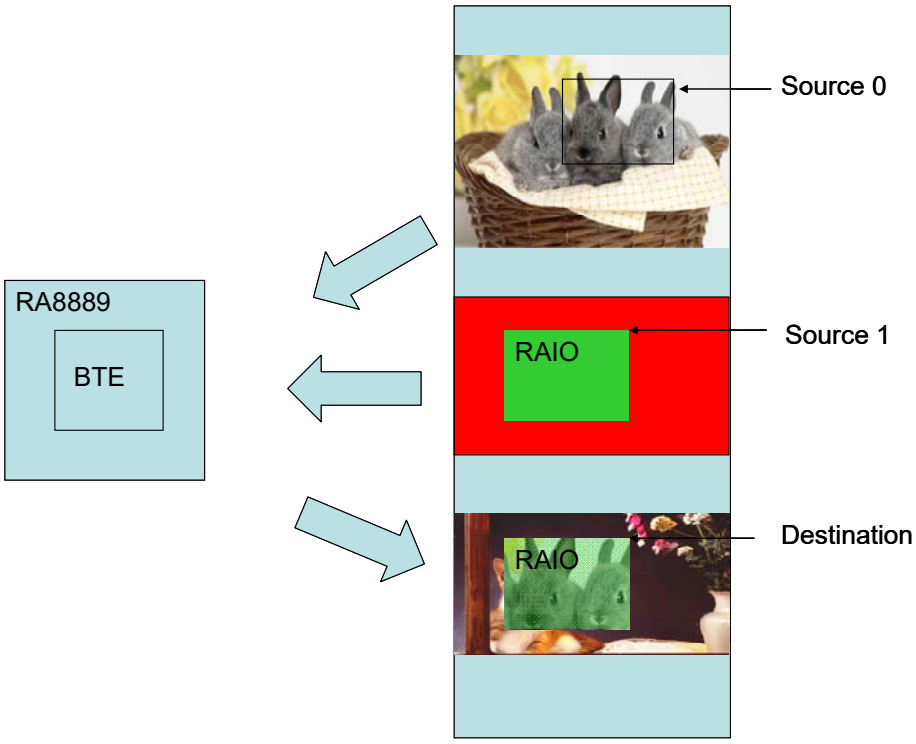


图 4-49 : Picture Mode 硬体资料流程

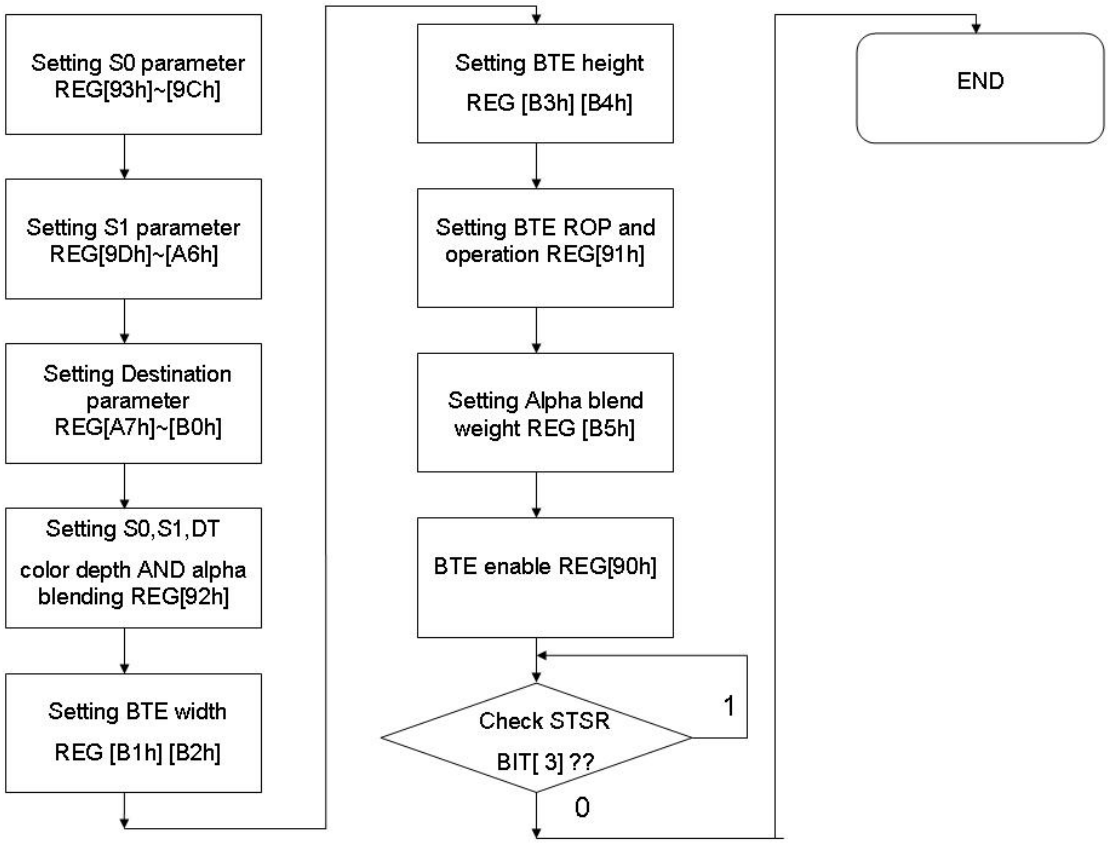


图 **4-50 : Picture Mode** 流程图

4.6.2: BTE Memory Copy with Alpha Blending in Picture mode 在 LCD 上的显示说明:

” Memory Copy with Alpha

blending”这个功能可以混合来源0与来源1这个功能有两种模式。然後显示在目的地的显示区块，的资料 －

Picture mode与Pixel mode我们使用，在这边，picture mode。做说明

Picture mode目的地资料 = (来源0 * (1- alpha Level)l) + (来源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)
)
```

范例:

```
/*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);

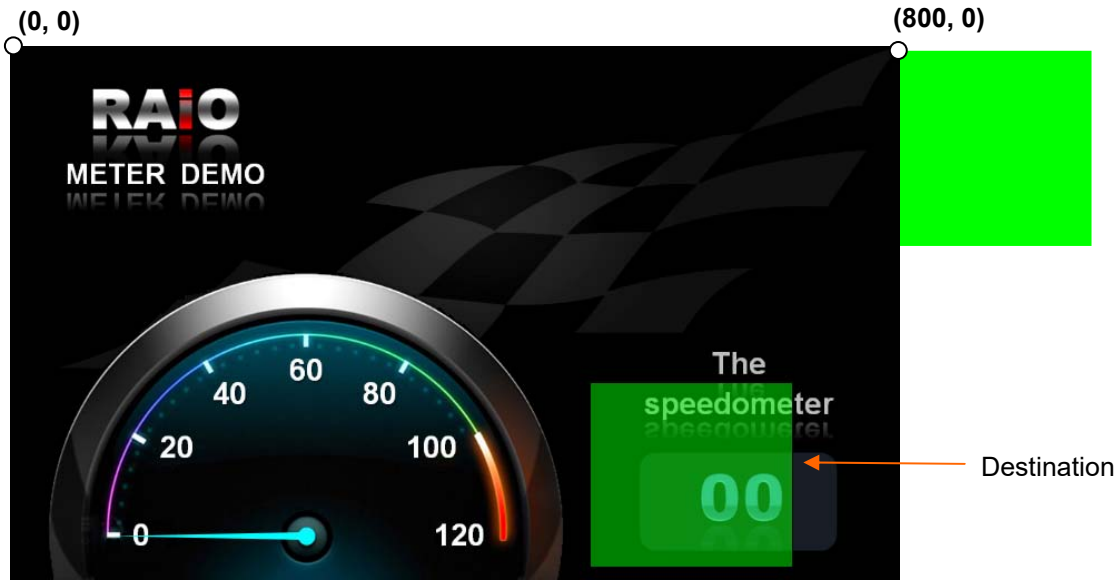


Figure 4-51：來源[(500,300) ~ (699,499)] ,來源 1[(800,0) ~ (999,199)] 執 alpha blending and showing on Destination[(500,300) ~ (699,499)], alpha = 16.

Destination:

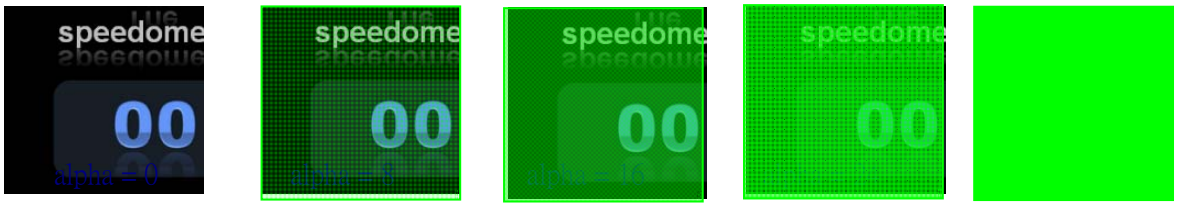


Figure 4-52: Destination in different alpha/transparent level.

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 上的显示说明

```
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
)
```

范例:

步骤 1:写一张 PNG(32bit ARGB) 图资到 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

步骤 2:写入一张 JPG 图片到 SDRAM

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

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

JPG Pic

步骤 3:来源 0 = JPG PIC，来源 1 = ARGB Data，执行 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);

执行 Alpha blending pixel mode 的显示画面

第五章 ：Picture In Picture Function

5.1:PIP Window

RA8889 在主要显示视窗里，支援使用者可以使用两个 PIP 视窗。PIP 视窗不支援通透功能，它仅提供使用者开启或关闭将图片资料覆盖在主显示画面上。当 PIP1 与 PIP2 画面重叠时，PIP1 会覆盖在 PIP2 之上。PIP 视窗的位置与大小由 REG[2Ah]到 REG[3Bh]与 REG[11h]来设定，PIP1 与 PIP2 共用设定寄存器，而且根据 REG[10h] 的 Bit[4]，去选择 REG[2Ah~3Bh]是 PIP1 或是 PIP2 视窗的参数。而相关的 PIP window 功能可经由若干功能位元做设置。PIP 视窗大小与起始位置水平方向是 4 个像素为一个单位，垂直则为一个像素。

5.2:PIP Windows Settings

一个 PIP 视窗的位置与大小是由 PIP 图像起始位址、 PIP 图像宽度、 PIP 显示 X/Y 座标、 PIP 图像 X/Y 座标、 PIP 视窗的色彩深度、 PIP 视窗的宽度与高度等等的寄存器来界定。PIP1 与 PIP2 共用设定寄存器，然後根据 REG[10h]Bit[4]来选择 REG[2Ah~3Bh]是 PIP1 或 PIP2 视窗的参数设定。

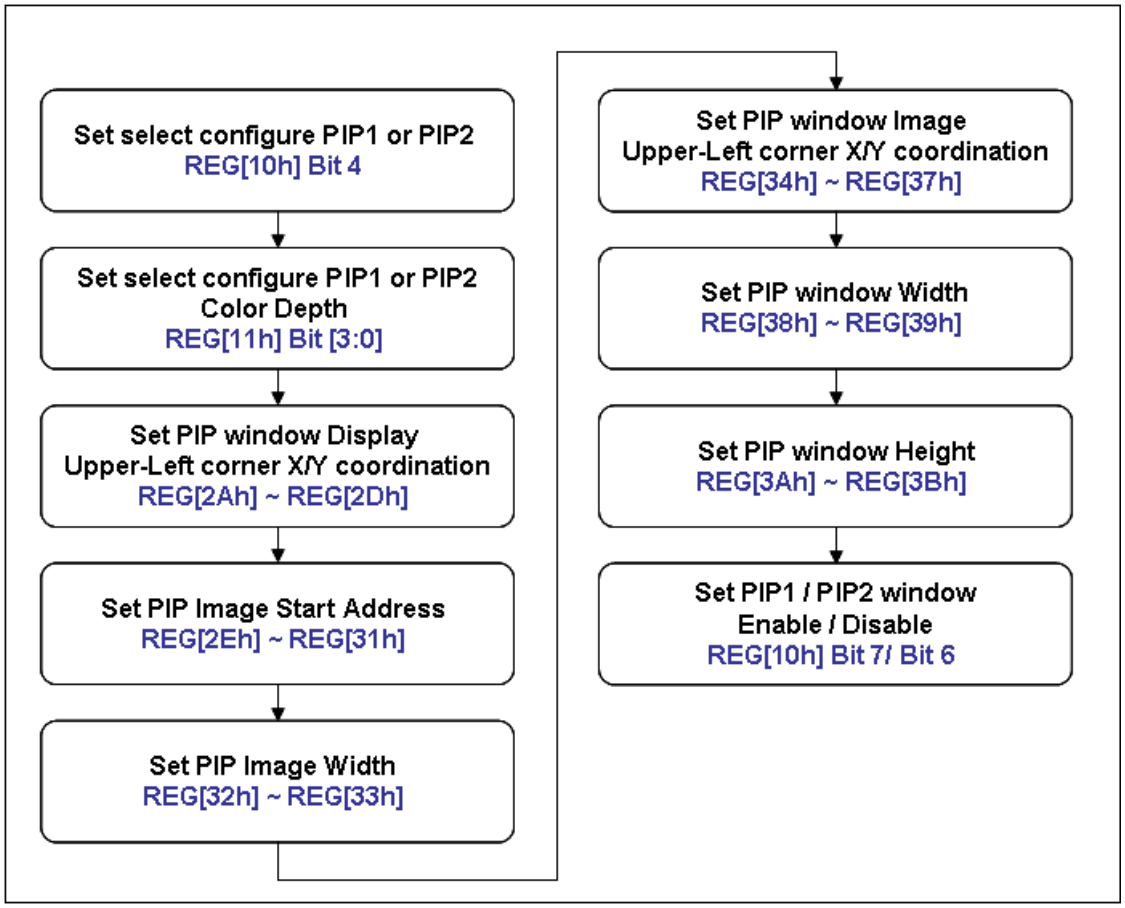


图 5-1： 流程图

5.3: PIP 功能图解:

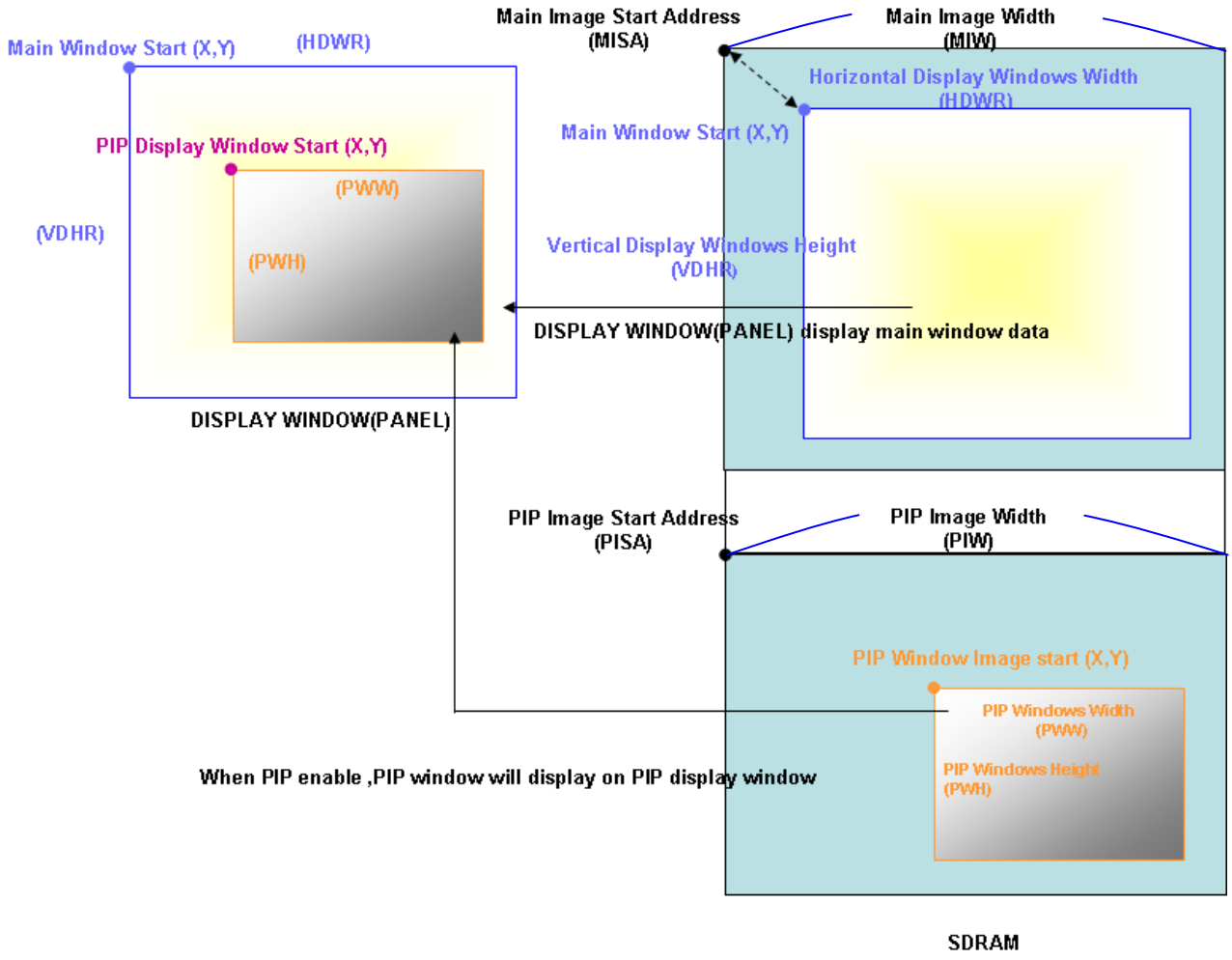


图 5-2

5.4:PIP 功能在 LCD 上的显示说明:

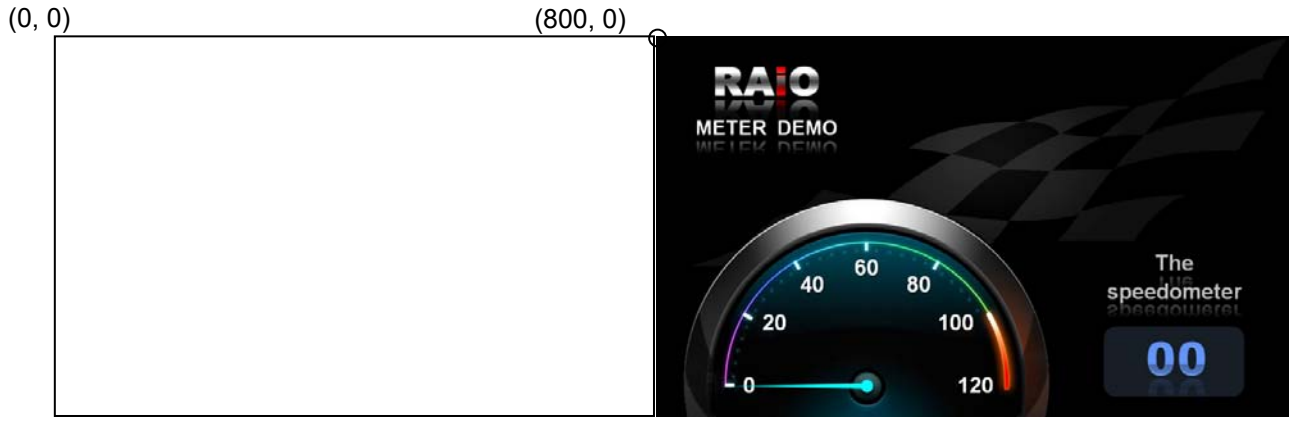
RA8889 支援使用者可以使用两个 PIP 视窗。PIP 视窗不支援通透功能，它仅提供使用者开启或关闭将图片资料覆盖在主显示画面上。当 PIP1 与 PIP2 画面重叠时，PIP1 会覆盖在 PIP2 之上。这个应用程式介面将会帮助使用者容易的去使用 PIP 这个功能。

```
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.
)
```

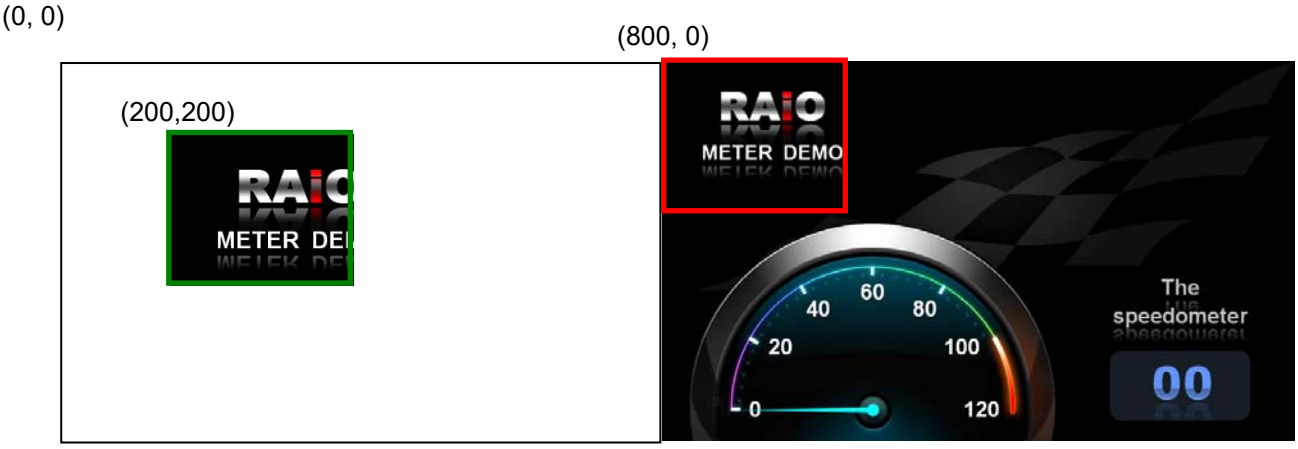
范例: (DMA Reference Chapter 3)

```
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);
```

步骤 1 :利用 DMA 功能写入一张图片到 SDRAM



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



红色方框为 PIP 来源，绿色方框为 PIP 显示视窗

第六章 ： 文字

RA8889 可支援两种字库来源，分为内建字库与外部字库，内建字库支援 ISO/IEC 8859-1/2/4/5 等字型。外部字库搭配上海集通公司 (Genitop Inc) 的部分串列字型 ROM，可支援多国字库或字型标准的显示，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 的文字输入可分为两种来源:

1. 内建字
2. 外部字型 ROM

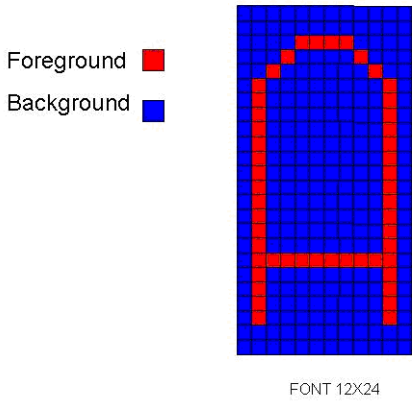


图 6-1： 范例

文字有一些参数设定，在 REG[CCh]~REG[DEh]，当你需要改变任何的字型参数，你可以参考下面的流程图。

字型的颜色设定在前景色与背景色的寄存器 REG[D2h]~[D7h]。

例如：我们写 font1 与 font2 两笔字串，各别有 64 笔字库的资料。

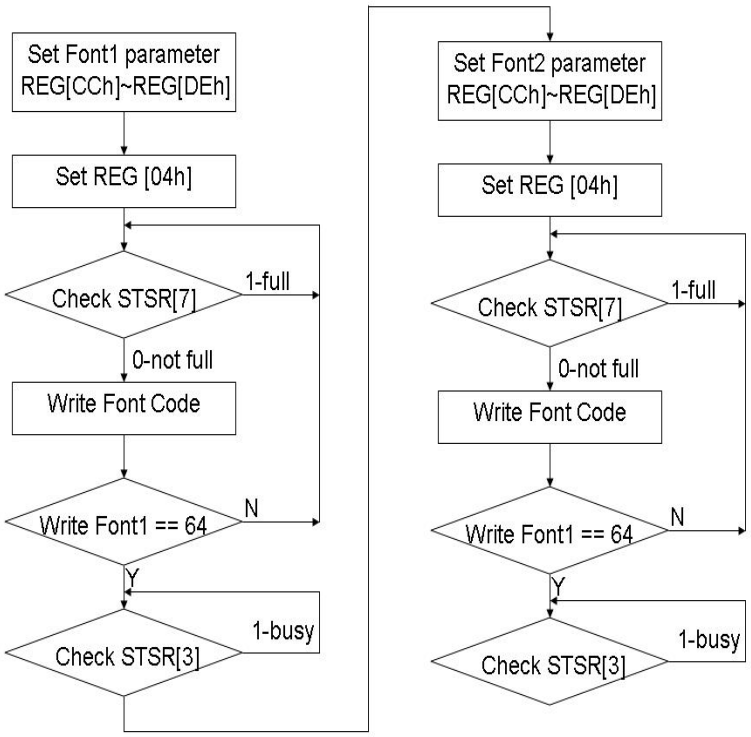


图 6-2：文字显示流程图

内建字

RA8889 内建 12x24 点的 ASCII 字型 ROM，提供使用者更方便的方式，用特定编码 (Code) 输入文字。内建的字集支援 ISO/IEC 8859-1/2/4/5 编码标准，此外，使用者可以透过 REG[D2h~D4h] 选择文字前景颜色，以及透过 REG[D5h~D7h] 选择背景颜色，文字写入的程序请参考下图:

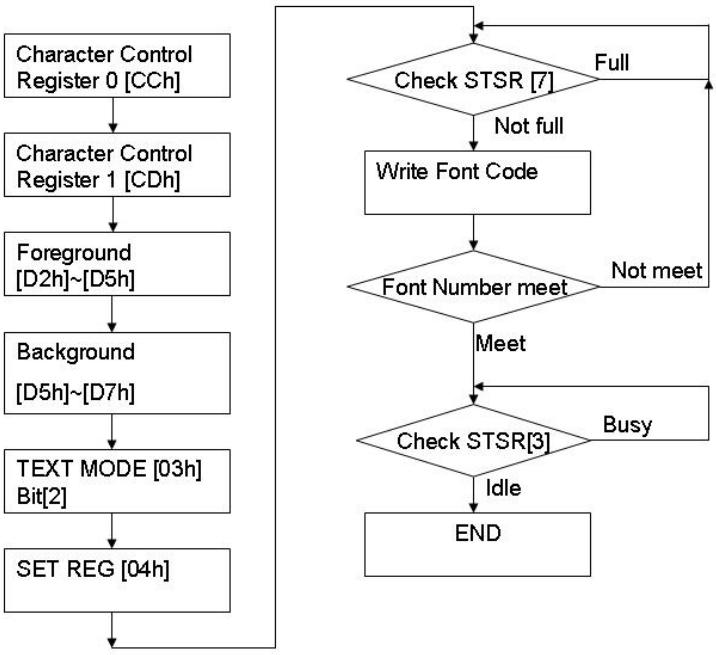


图 6-3：内部ASCII字库使用流程

表 6-1 内含ISO/IEC 8859-1 。标准的字集ISO，是国际标准化组织的简称ISO/IEC 8859-1 又称 ”Latin-1” 是国际标准化组织内，「西欧语言」或ISO/IEC 8859 的第一个发展的8 以。位元字集ASCII包含了，为基础0xA0到0xFF的范围内192

包括阿尔巴尼亚语、巴斯克语、布列塔尼语、加泰羅尼亚语，此字集编码使用遍及西欧。个拉丁字母及符号、丹麦语、荷蘭语、法羅语、弗裡斯语

(Frisian)

语、爱尔蘭盖尔语、义大利语、义大利语、拉丁语、盧森堡语、挪威、加利西亚语、德语、格陵蘭语、冰岛。语、葡萄牙语、里托羅曼斯语、苏格蘭盖尔语、西班牙语及瑞典语

但仍会标明为，英语虽然没有重音字母ISO 8859-1 如南非荷蘭语、斯瓦希里，欧洲以外的部份语言，编码语、印尼语及马來语、菲律宾他加洛语 (Tagalong) 也可使用ISO8859-1 。编码

表 6-1 : ASCII Block 1(ISO/IEC 8859-1)

表 6-2 内为 ISO/IEC 8859-2 的标准字集，又称 Latin-2 或「中欧语言」，是国际标准化组织内 ISO/IEC 8859 的第二个 8 位字元集。此字元集主要支援以下文字：克羅埃西亚语、捷克语、匈牙利语、波蘭语、斯洛伐克语、斯洛维尼亚语、索布语。而阿尔巴尼亚语、英语、德语、拉丁语也可用此字元集显示。芬蘭语中只有外來语才有 å 字元，若不考虑此字元，ISO/IEC8859-2 也可用於瑞士及芬蘭语。

表 6-2: ASCII Block 2 (ISO/IEC 8859-2)

	☺	☻	♥	♦	♣	♠	●	+	○	○	♂	♀	♪	♫	☼
►	◄	↕	‼	¶	§	▬	↨	↑	↓	→	←	⌐	↔	▲	▼
	!	”	#	$	%	&	’	(	)	*	+	,	—	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
	Ą	˘	Ł	¤	Ľ	Ś	§	¨	Š	Ş	Ť	Ź	-	Ž	Ż
°	ą	˛	ł	´	ľ	ś	ˇ	¸	š	ş	ť	ź	˝	ž	ż
Ŕ	Á	Â	Ă	Ä	Ĺ	Ć	Ç	Č	É	Ę	Ë	Ě	Í	Î	Ď
Đ	Ń	Ň	Ó	Ô	Ő	Ö	×	Ř	Ů	Ú	Ű	Ü	Ý	Ţ	ß
ŕ	á	â	ă	ä	ĺ	ć	ç	č	é	ę	ë	ě	í	î	ď
đ	ń	ň	ó	ô	ő	ö	÷	ř	ů	ú	ű	ü	ý	ţ	˙

表 6-3 内为 ISO/IEC 8859-4 之标准字集，又称 Latin-4 或「北欧语言」，是国际标准化组织内 ISO/IEC 8859 的第四个 8 位字元集，它设计來表示爱沙尼亚语、格陵蘭语、拉脫维雅语、立陶宛语及部分萨米语 (Sami) 文字，此字元集同时能支援以下文字：丹麦语、英语、芬蘭语、德语、拉丁语、挪威语、斯洛维尼亚语及瑞典语。

表 6-3: ASCII Block 3 (ISO/IEC 8859-4)

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																
9																
A		Ą	ĸ	Ŗ	¤	Ĩ	Ļ	§	¨	Š	Ē	Ģ	Ŧ	-	Ž	¯
B	°	ą	˛	ŗ	´	ĩ	ļ	ˇ	¸	š	ē	ģ	ŧ	Ŋ	ž	ŋ
C	Ā	Á	Â	Ã	Ä	Å	Æ	Į	Č	É	Ę	Ë	Ė	Í	Î	Ī
D	Đ	Ņ	Ō	Ķ	Ô	Õ	Ö	×	Ø	Ų	Ú	Û	Ü	Ũ	Ū	ß
E	ā	á	â	ã	ä	å	æ	į	č	é	ę	ë	ė	í	î	ī
F	đ	ņ	ō	ķ	ô	õ	ö	÷	ø	ų	ú	û	ü	ũ	ū	˙

表 6-4 内为 ISO/IEC 8859-5 之标准字集，是国际标准化组织内 ISO/IEC 8859 的第五个 8 位字元集，它设计來表示保加利亚语、俄语、塞尔维亚语与马其顿人语。

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

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																
9																
A		Ё	Ђ	Ѓ	Є	S	I	Ї	J	Љ	Њ	Ћ	Ќ	-	Ў	Џ
B	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
C	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
D	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
E	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
F	№	ё	ђ	ѓ	є	s	i	ї	j	љ	њ	ћ	ќ	§	ў	џ

外部字型ROM

RA8889 的外部串列 ROM 介面针对不同应用提供了许多种字体。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援产品编号包含: GT21L16T1W 、GT30L16U2W 、GT30L24T3Y 、GT30L24M1Z 、GT30L32S4W 、GT23L24F6Y 、GT23L24S1W。根据不同的产品，它提供了不同的大小，其中包含 16x16 、24x24 、32x32 以及各种不同宽度的字体大小。详细的说明如下表列：

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 显示字

6.2.1: 内建字:

RA8889内建12x24点的ASCII字形ROM提供使用者更方便的方式用特定编码（Code）输入文字。内建字库支援了ISO//IEC 8859-1/2/4/5的标准字集。

```
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
)
```

范例:

```
/*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");
```

实际的显示画面

图 6-4 内建字范例

6.2.2: 透过外部字库写入 Big5 字串

RA8889 的外部串列 ROM 介面针对不同应用提供了许多种字体。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援产品编号包含: GT21L16T1W 、 GT30L16U2W 、 GT30L24T3Y 、 GT30L24M1Z 、 GT30L32S4W 、 GT23L24F6Y 、 GT23L24S1W。根据不同的产品，它提供了不同的大小，其中包含 16x16 、 24x24 、 32x32 以及各种不同宽度的字体大小。

这个应用程式介面是使用了 RA8889 外部串列字库来列印出 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
)
```

范例:

```
/*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
```

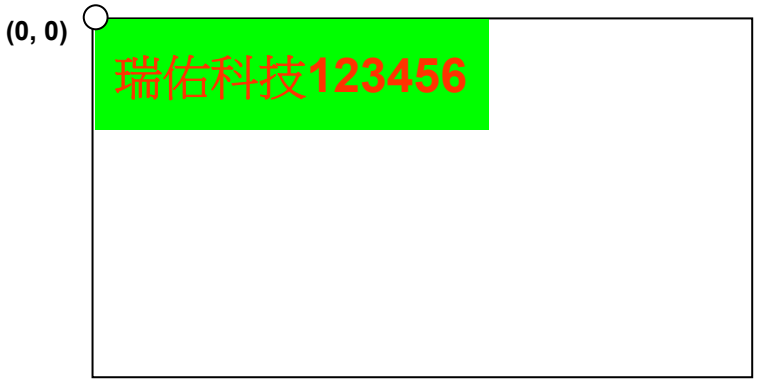


图 6-5 透过外部字库写入 Big5 字串

6.2.3: 透过外部字库写入 GB2312 字串:

RA8889 的外部串列 ROM 介面针对不同应用提供了许多种字体。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援产品编号包含: GT21L16T1W 、 GT30L16U2W 、 GT30L24T3Y 、 GT30L24M1Z 、 GT30L32S4W 、 GT23L24F6Y 、 GT23L24S1W。根据不同的产品，它提供了不同的大小，其中包含 16x16 、 24x24 、 32x32 以及各种不同宽度的字体大小。

这个应用程式介面是使用了 RA8889 外部串列字库来列印出 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
)
```

范例:

```
/*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
```

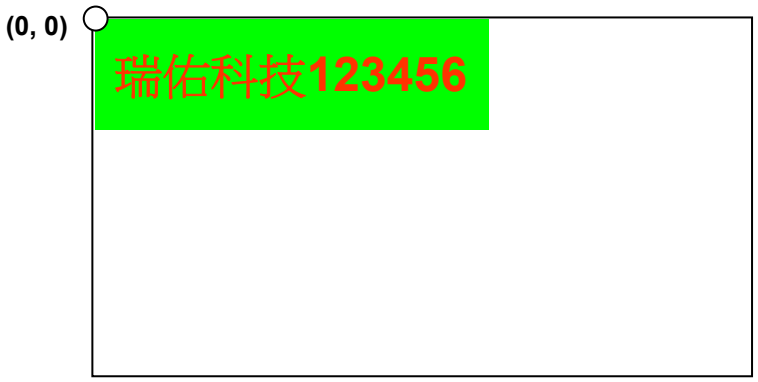


图 6-6: 透过外部字库写入 GB2312 字串

6.3.4: 透过外部字库写入 GB12345 字串

RA8889 的外部串列 ROM 介面针对不同应用提供了许多种字体。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援产品编号包含: GT21L16T1W 、 GT30L16U2W 、 GT30L24T3Y 、 GT30L24M1Z 、 GT30L32S4W 、 GT23L24F6Y 、 GT23L24S1W。根据不同的产品，它提供了不同的大小，其中包含 16x16 、 24x24 、 32x32 以及各种不同宽度的字体大小。

这个应用程式介面是使用了 RA8889 外部串列字库来列印出 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
)

范例:

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

图 6-7: 透过外部字库写入 GB12345 字串

6.3.5: 透过外部字库写入 Unicode 字串

RA8889 的外部串列 ROM 介面针对不同应用提供了许多种字体。此介面相容於集通公司 (Genitop Inc) 的部分串列字型 ROM，支援产品编号包含: GT21L16T1W 、 GT30L16U2W 、 GT30L24T3Y 、 GT30L24M1Z 、 GT30L32S4W 、 GT23L24F6Y 、 GT23L24S1W。根据不同的产品，它提供了不同的大小，其中包含 16x16 、 24x24 、 32x32 以及各种不同宽度的字体大小。

这个应用程式介面是使用了 RA8889 外部串列字库来列印出 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)*/
)
```

范例:

```
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");
```

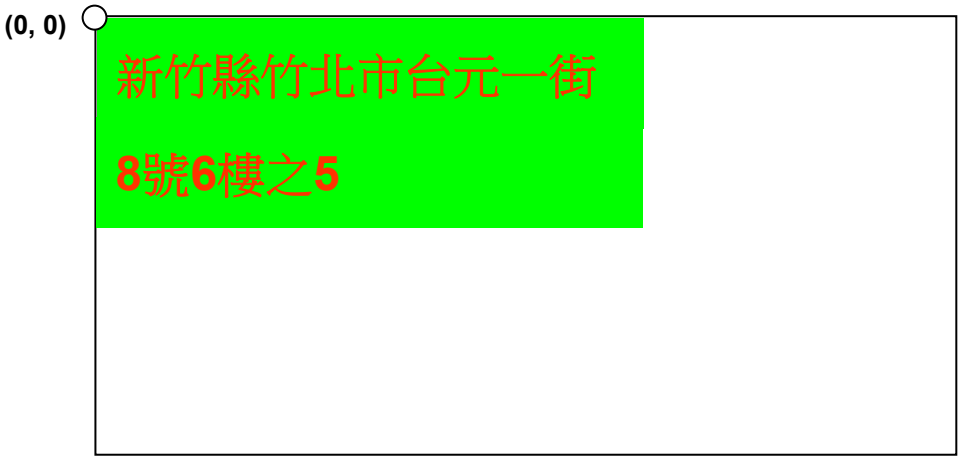


图 6-8: 透过外部字库写入 Unicode 字串

第七章 ：PWM (Pulse Width Modulation)

RA8889有2组16定时器。位定时器0和1）具有脉冲宽度调制PWM定时器。功能（0
。其用於以大电流的设备，具有一个死区生成器
定时器0和1有一个共用的8位元的预分频器(prescalar)从而产生，每个定时器还有一个时脉除频器。4
）个不同的除频信号1，1/2，1/4和1/8。（
最後透过timer count buffer (TCNTBn)用来调整PWM输出的周期时间以及利用timer compare buffer
(TCMPBn)）脉冲宽度调制的值来进行PWM详细公式与图解如下列说明。产生一个稳定的脉波，（:

PWM CLK = (Core CLK / Prescalar) /2[clock divided]

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

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

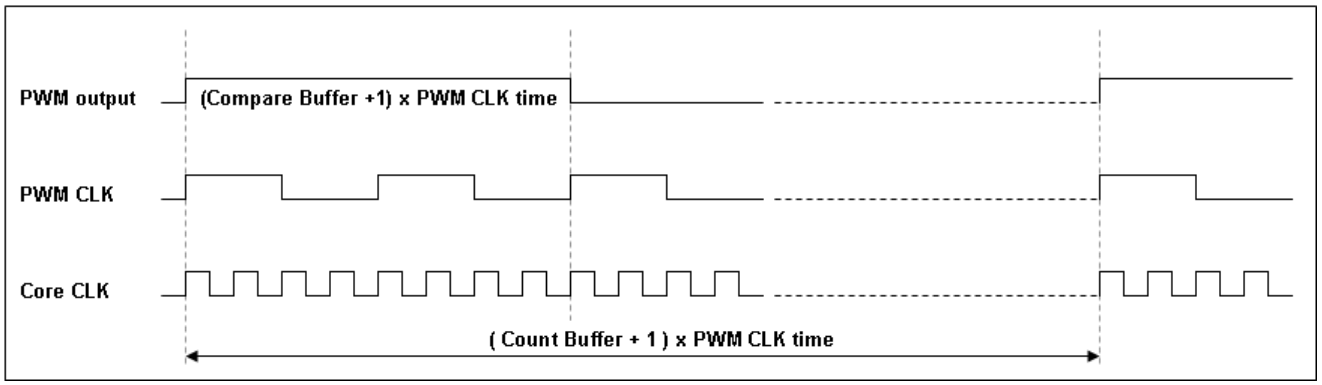


图 7-1 : PWM 时序图

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 */
)
```

范例:

当 Core CLK = 60MHz

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

PWM0 输出:

/*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:

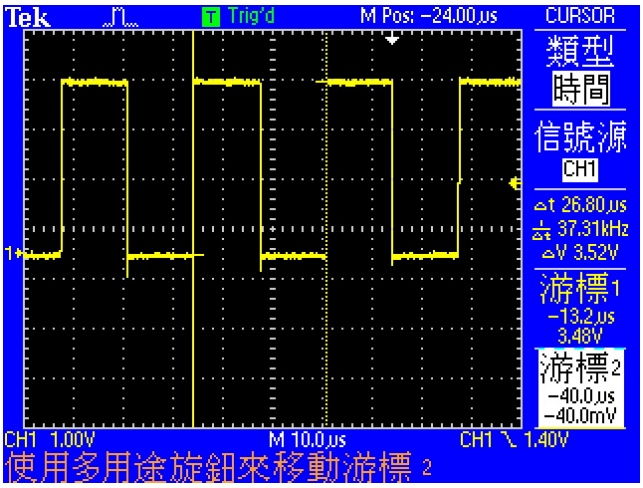


图 7-2：PWM0 输出波形周期

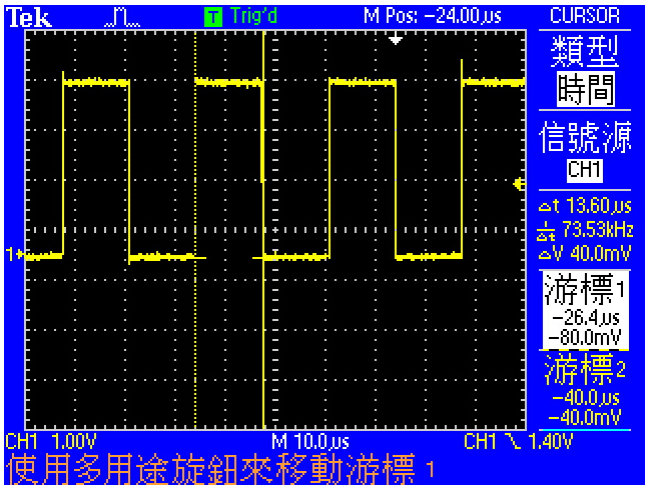


图 7-3：PWM0 输出高电位的时间

PWM1 输出:

/*On_off = 1, Enable PWM.

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:

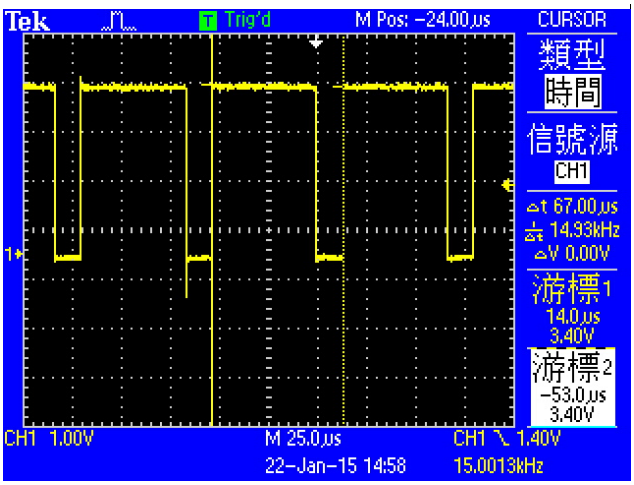


图 7-4：PWM1 输出波形周期

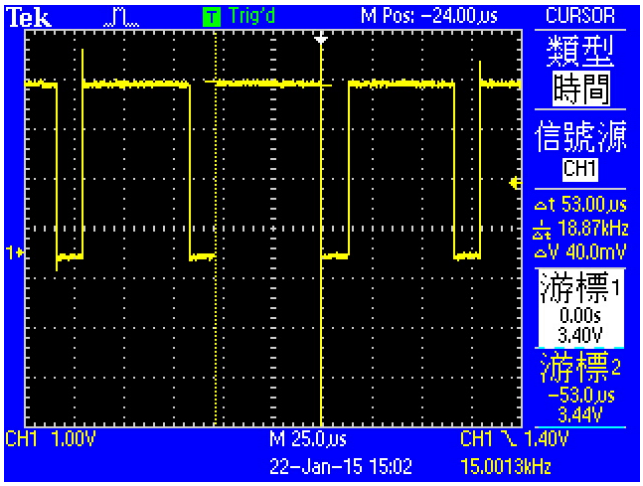


图 7-5：PWM1 输出高电位的时间

第八章 ： 键盘扫描

键盘扫描会扫描并读取键盘状态，而键盘矩阵会由硬件来切换扫描线。这个功能可以提供键盘应用，圖8-1 显示的是基本的键盘应用电路。RA8889 为因应外部电路需求，已经在 KIN[4:0] 内建上拉电阻。

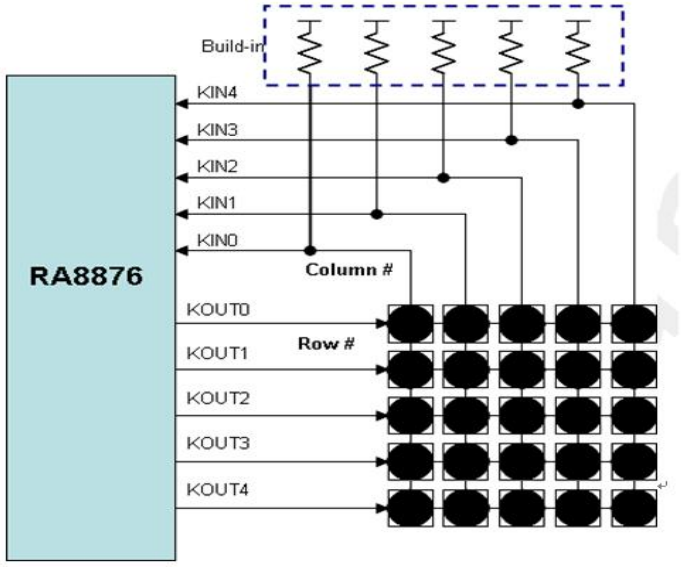


圖8-1 : Key-Pad Application

8.1 键盘扫描操作方式

RA8889 键盘扫描控制器的特点如下:

1. 最多支持 5x5 键盘矩阵
2. Key-Scan 具有可程序化的扫描频率与取样时间。
3. 可调整的长按键时间
4. 支持多键同时按
   注意: 可同时按 2 个按键或是要按 3 个按键 (但是 3 按键组成不能是 90°排列)
5. 可使用按键来唤醒系统

KSCR 是键盘扫描的状态缓存器，这个缓存器被使用在了解键盘扫描的操作状态，如取样时间、取样频率、致能长按键。而若是有按键按下，使用者也可以透过中断得知。在 KSCR2 bit1~0 纪录目前按下的按键数目。然后使用者可以透过读取 KSDR 得到按键码。

注：“Normal key” 是在以取样时间为基础上有被认知为合格的按下按键行为。“Long Key” 则是在长按键取样周期下有被认知为合格的按下按键行为。先产生 "Normal Key" 才会产生 “Long Key” ，有时在某些应用上需要分开使用。

表 8-1 是在 ”Nnomal Key” 下键码与键盘矩阵的对应，按下的按键键码会被存在 KSDR0~2。如果是长时间按下按键，则表现出的会是 ”Long Key”，而相关键码在表 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

当按下多键时，最多有三个按键会被存在 KSDR0，KSDR1 与 KSDR2 三个缓存器中。注意键码储存的方式与按键位置有关或者说与键码有关，而与按键顺序无关，请参下列例子:

在相同时间按下键码 0x34, 0x00 and 0x22，在 KSDR0~2 储存方式如下:

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

以上所提的键盘扫描设定介绍如下:

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]

致能键盘扫描功能(Key-Scan)，使用者可以使用下列方法检查按键状态:

1) Software check method: 检查 Key-scan 的状态值 (status)，来得知是否被按下。

2) Hardware check method: 由中断来得知是否有按键被按下。

若是设定中断致能 (INTEN bit[3]) 为 1，那么有键盘有被按下时就会产生中断。而当中断产生时，Key-scan 中断状态旗标 (bit[3] of INTF) 将永远为 1，无论使用何种方法，使用者在读取键码后必须清除中断状态旗标，否则以后就不会再产生中断。

此外，RA8889 在省电模式下支持“Key-stroke wakeup ”，经由设定完成后，任何按键触发都可以将 RA8889 由睡眠模式中唤醒。为了检知唤醒事件，MPU 可以透过软件程序去轮询 RA8889 的中断是否产生。

以上应用的缓存器程序设定流程图如下:

1. 软件方法

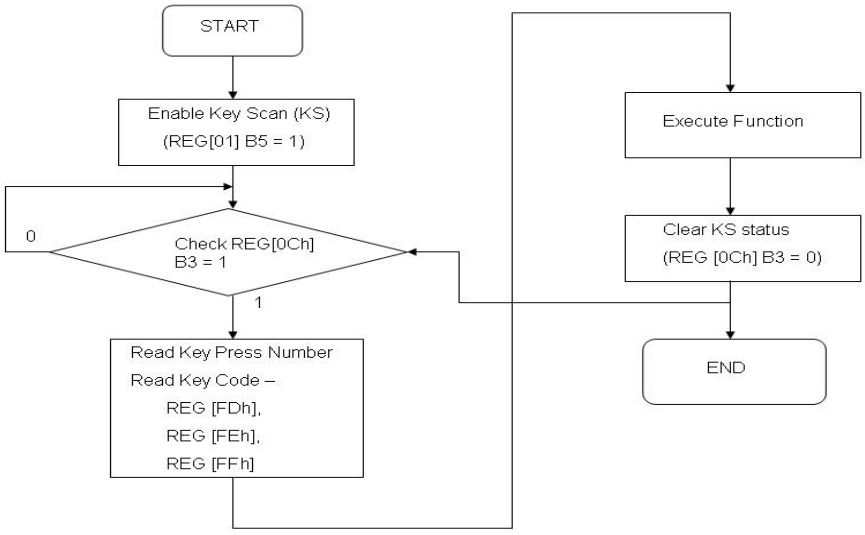


圖 8-2 : Key-Scan Flowchart for Software Polling

2. 硬件方法

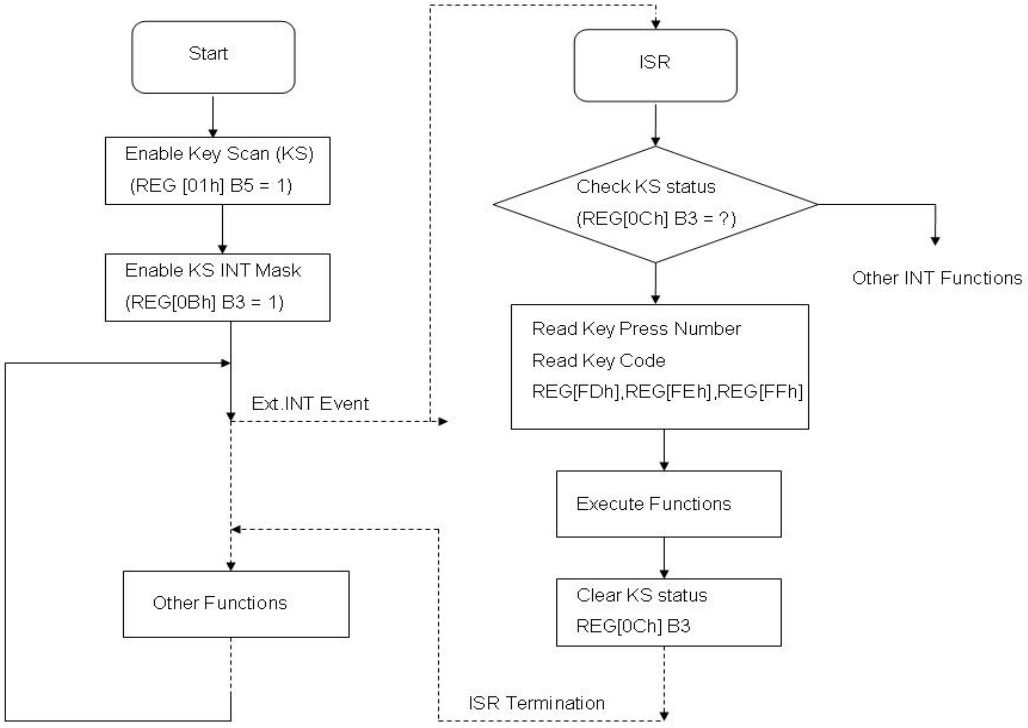


圖 8-3 : Key-Scan for Hardware Interrupt

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**

条件:

x:显示 Key Code 的 X 座标 = 0

y:显示 Key Code 的 Y 座标 = 0

FontColor:显示 Keycode 的颜色 = 黑色

BackGroundColor:显示 Keycode 的背景色 = 白色

Example:

如按下下列三个按键

	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 显示画面:

001123

附录 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 : SDRAM Initial

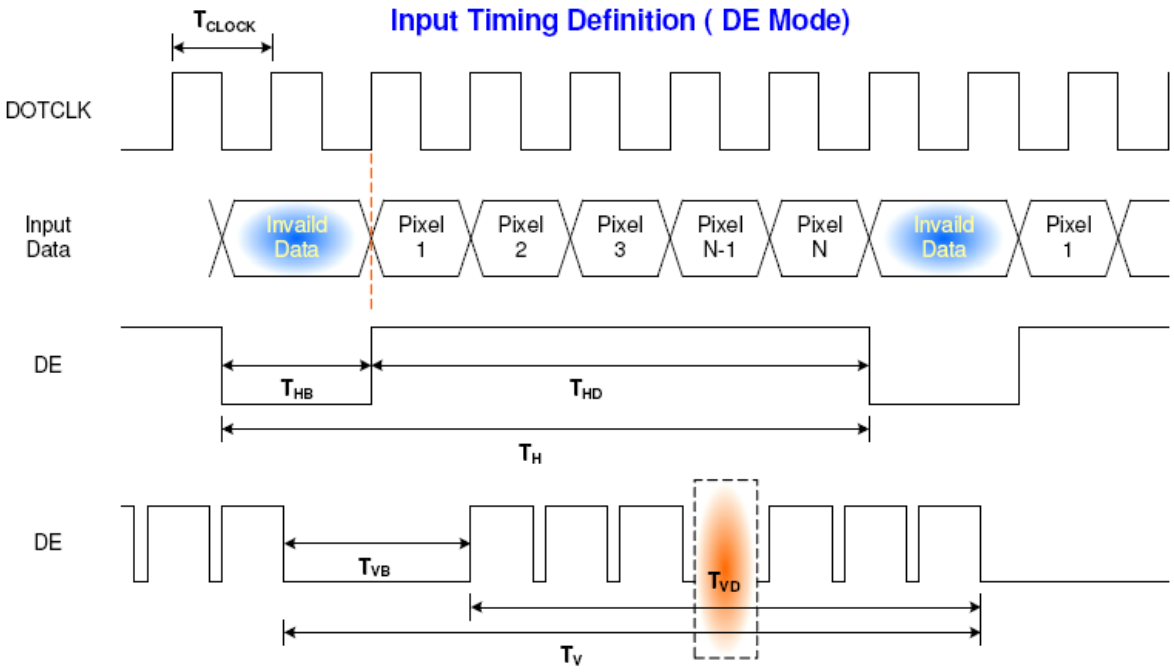
```
void RA8889_SDRAM_initial(void)
{
unsigned char   CAS_Latency;
unsigned short  Auto_Refresh;

      CAS_Latency=3;
      Auto_Refresh=(64*DRAM_FREQ*1000)/(4096);
      Auto_Refresh=Auto_Refresh-2; // Start [refresh] in advance to avoid just reaching the limits.

      LCD_RegisterWrite(0xe0,0x29);
      LCD_RegisterWrite(0xe1,CAS_Latency);          //CAS:2=0x02,CAS:3=0x03
      LCD_RegisterWrite(0xe2,Auto_Refresh);
      LCD_RegisterWrite(0xe3,Auto_Refresh>>8);
      LCD_RegisterWrite(0xe4,0x01);

      Check_SDRAM_Ready();
}
```

附录 3 : 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
```

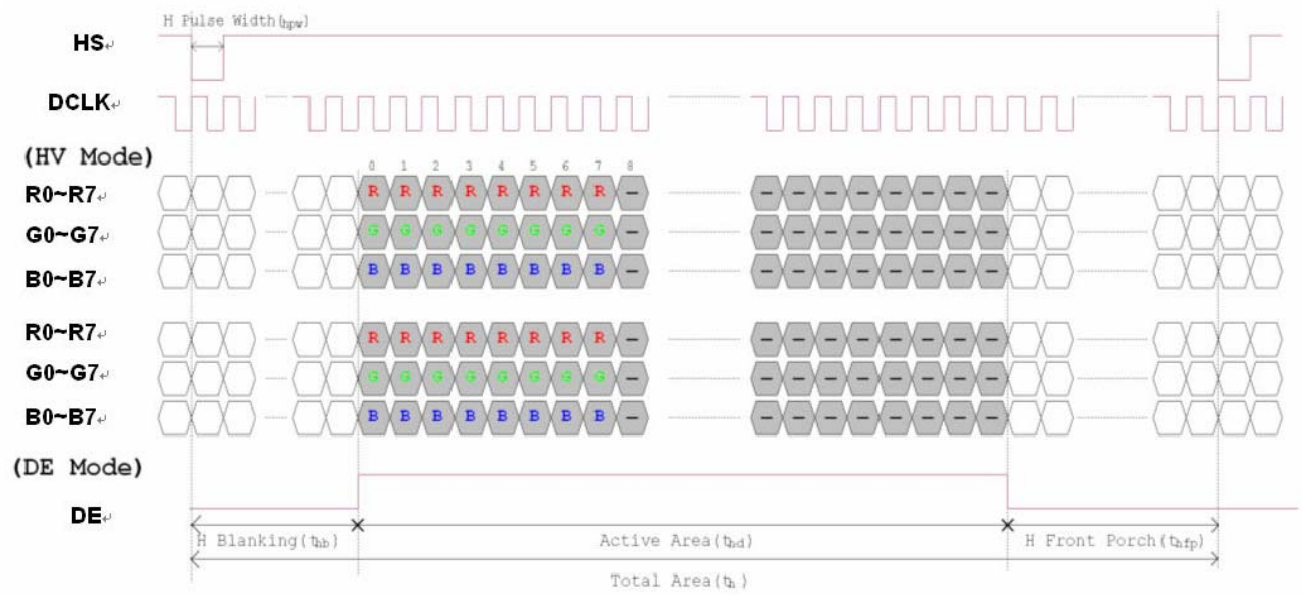
```
PDATA_Set_RGB();                //Parallel XPDAT[23:0] Output Sequence is set as RGB
DE_High_Active();               //Set XDE Polarity worked as High active.

//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);
}
```

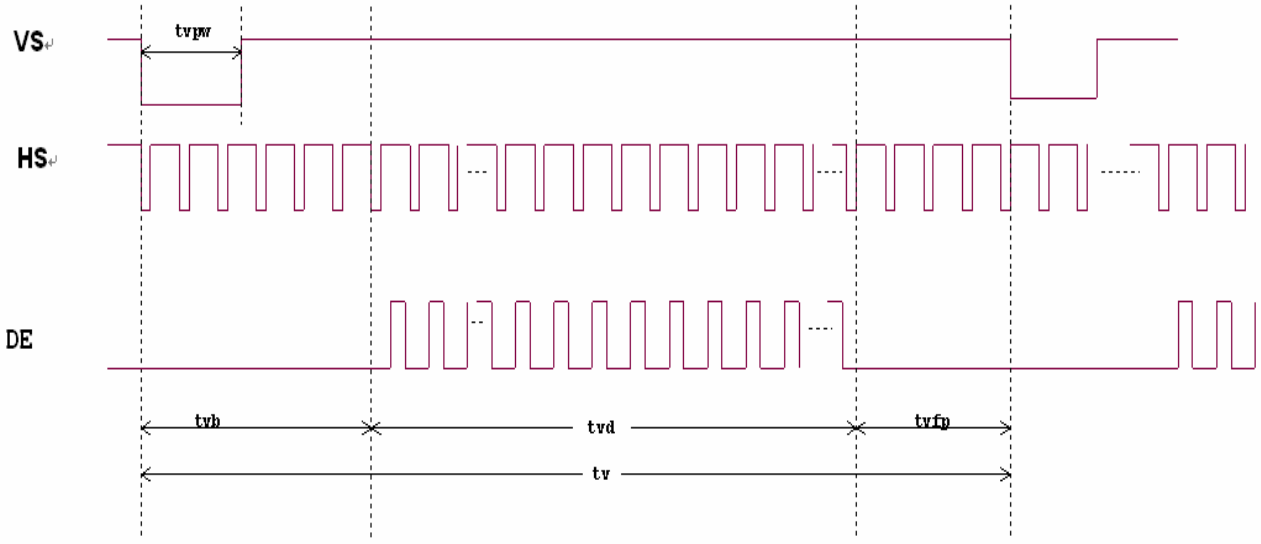
AT070TN92 Timing :

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);
```

附录 **4 : RAIO** 自建字库

RAIO 自制自建字库，提供三种大小的 ASCII 字体，分别为 8x12 、 16x24 、 32x48，本章节 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 几种模式，但并不支援 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
)
```

范例:

```
//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");
```

图 8-1： 自建字画面示意图