RAiO

RA8889

Application Programming Interface

May 21, 2025

RAiO Technology Inc.

CONTENTS

Introduction

In this document, we will introduce 8 RA8889 functions. About MCU Write Data and Display Input Data Format, Draw, DMA, BTE, PIP, font and PWM function…etc, and provider Application Programming Interface about that. Firstly, you need to select your **MCU I/F Protocol**、**LCD Panel**、**MCU and Display Input data format**、**GT Font ROM** and **PLL**(SDRAM CLK、Core CLK、Pixel CLK) in "Userdef.h" for meeting application environment. Detailed description and settings, such as PLL Setting, LCD initialization, please refer appendix 1/2.

MCU I/F Protocol : RA8889 provider Parallel 8080、Parallel 6800、SPI – 3 wire、SPI – 4 wire、IIC for MCU Interface.

SDRAM : RA8889 build in 128Mbits SDRAM

MCU and Display Input data format:

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.

GT Font ROM : RA8889 supports the various fonts to display by using the external Genitop font ROM.

PLL(SDRAM CLK、Core CLK、Pixel CLK) : SDRAM CLK (Maximum 166MHz) 、Core CLK (Maximum 120MHz)、Pixel CLK (Maximum 100MHz).

SDRAM CLK must > Core CLK.

In 16bpp color depth : Core CLK ≧ 1.5 * Pixel CLK

In 24bpp color depth : Core CLK ≧ 2 * Pixel CLK

Chapter 1: MCU Write Data and Display Input Data Format

Overview:

RA8889 provide 6 tpye mode for MCU write data to SDRAM :

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

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

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

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

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

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

This chapter will help user easy to use the 6 type mode.

1.1.1:8-bit MCU interface, 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

Table 1 : 8-bit MCU, 8bpp mode data format

API :

Using 8 bits MCU interface and 8bpp color depth. MCU write data into 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
)
```

Example :

```
MPU8_8pp_Memory_Write(0,0,128,128 ,gImage_8);
MPU8_8pp_Memory_Write(200,0,128,128 ,gImage_8);
MPU8_8pp_Memory_Write(400,0,128,128 ,gImage_8);
//gImage_8 is 8bit array data ,8bpp color deepth ,size 128x128 picture data.
```

The example dispaly on LCD:

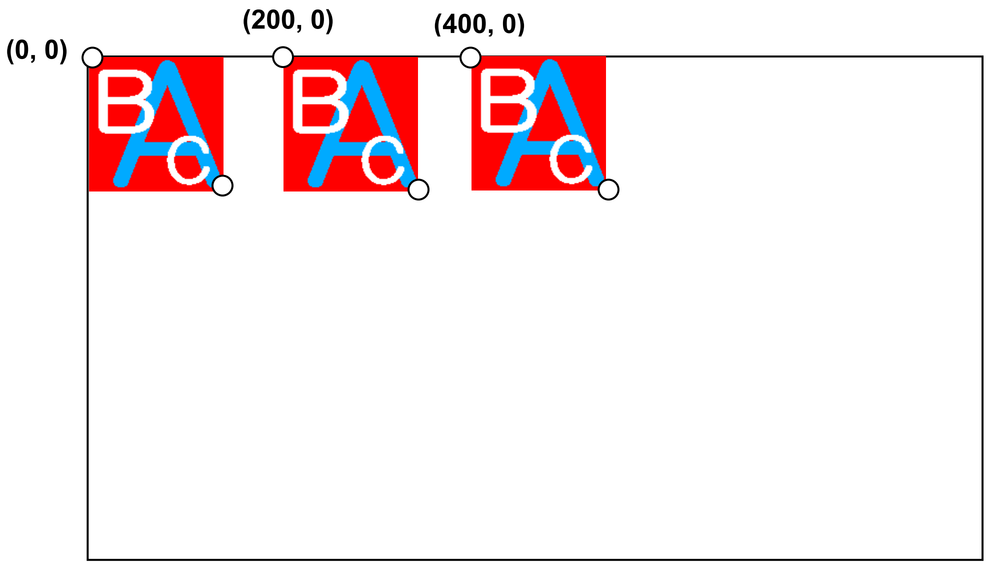


Figure 1.1 :Using 8 bits MCU interface and 8bpp color depth. MCU write data into SDRAM.

1.1.2:8-bit MCU interface, 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

Table 2 : 8-bit MCU, 16bpp mode data format

API :

Using 8 bits MCU interface and 16bpp color depth. MCU write data into 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
)
```

example:

```
MPU8_16pp_Memory_Write (0,0,128,128,gImage_16);
MPU8_16pp_Memory_Write (200,0,128,128,gImage_16);
MPU8_16pp_Memory_Write (400,0,128,128,gImage_16);
//gImage_16 is 8bit array data , 16bpp color deepth ,size 128x128 picture data.
```

The example dispaly on LCD:

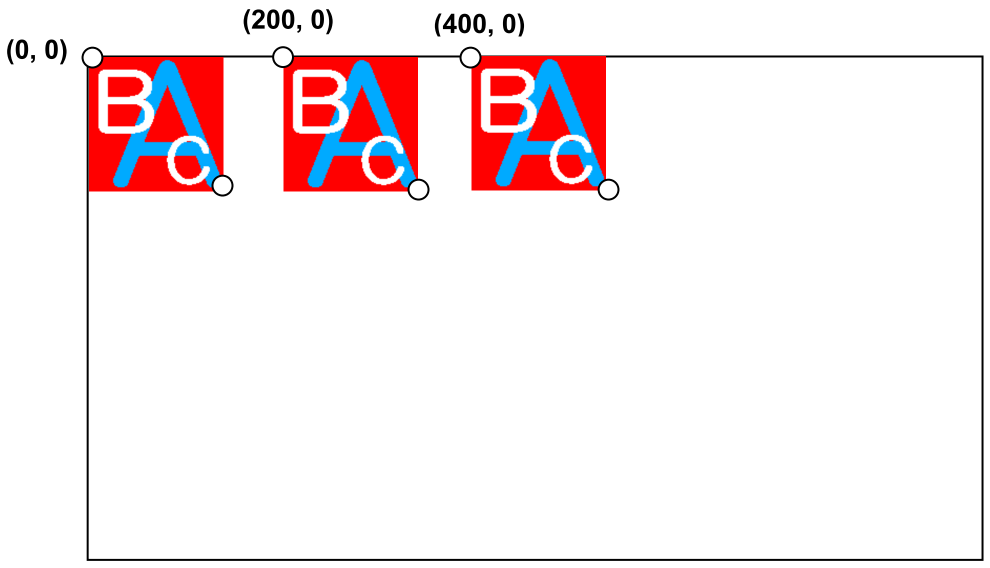


Figure 1.2 : Using 8 bits MCU interface and 16bpp color depth. MCU write data into SDRAM.

1.1.3 :8-bit MCU interface, 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

Table 3 : 8-bit MCU, 24bpp mode data format

API :

Using 8 bits MCU interface and 24bpp color depth. MCU write data into 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
)
```

example:

```
MPU8_24pp_Memory_Write (0,0,128,128 ,gImage_24);
MPU8_24pp_Memory_Write (200,0,128,128,gImage_24);
MPU8_24pp_Memory_Write (400,0,128,128,gImage_24);
//gImage_24 is 8bit array data , 24bpp color deepth ,size 128x128 picture data.
```

The example dispaly on LCD:

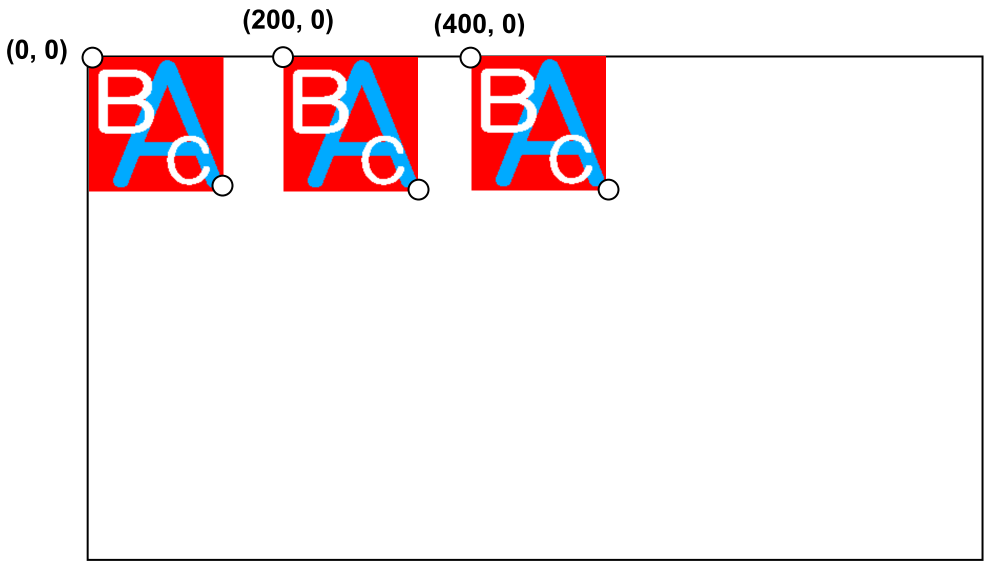


Figure 1.3 : Using 8 bits MCU interface and 24bpp color depth. MCU write data into SDRAM

1.1.4:16-bit MCU interface, 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

Table 4 : 16-bit MCU, 16bpp mode data format

API :

Using 16 bits MCU interface and 16bpp color depth. MCU write data into 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
)
```

example:

```
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 is 16bit array data , 16bpp color deepth ,size 128x128 picture data.
```

The example dispaly on LCD:

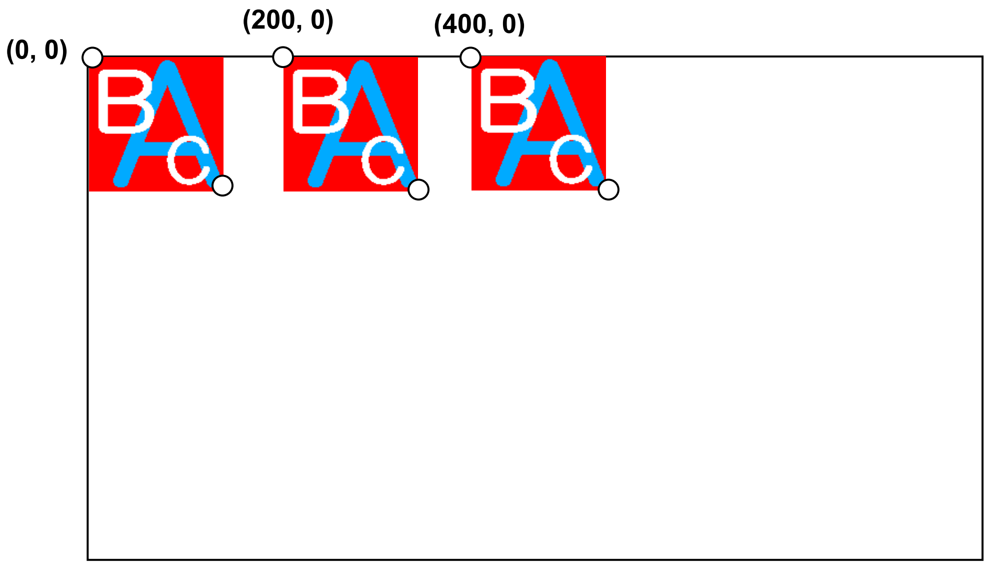


Figure 1.4 : Using 16 bits MCU interface and 16bpp color depth. MCU write data into SDRAM.

1.1.5 :16-bit MCU interface, 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

Table 5 : 16-bit MCU, 24bpp mode 1 data format

API :

Using 16 bits MCU interface and 24bpp color depth mode 1. MCU write data into 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
)
```

example:

```
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 is 16bit array data , 24bpp color deepth mode 1,size 128x128 picture data.
```

The example dispaly on LCD:

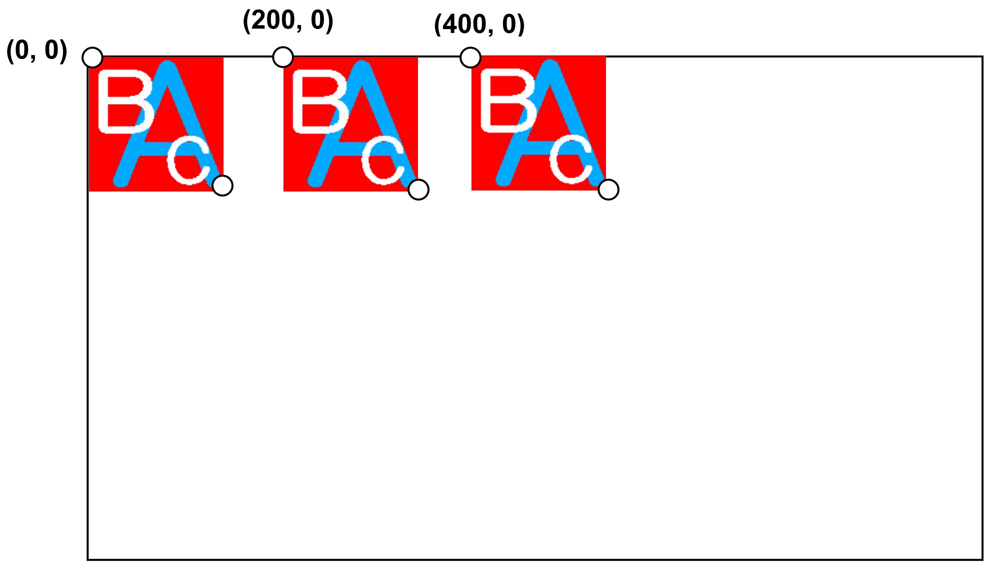


Figure 1.5 : Using 16 bits MCU interface and 24bpp color depth mode 1. MCU write data into SDRAM.

1.1.6:16-bit MCU interface, 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

Table 6 : 16-bit MCU, 24bpp mode 2 data format

API :

Using 16 bits MCU interface and 24bpp color depth mode 2. MCU write data into 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
)
```

example:

```
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 is 16bit array data , 24bpp color deepth mode 2,size 128x128 picture data.
```

The example dispaly on LCD:

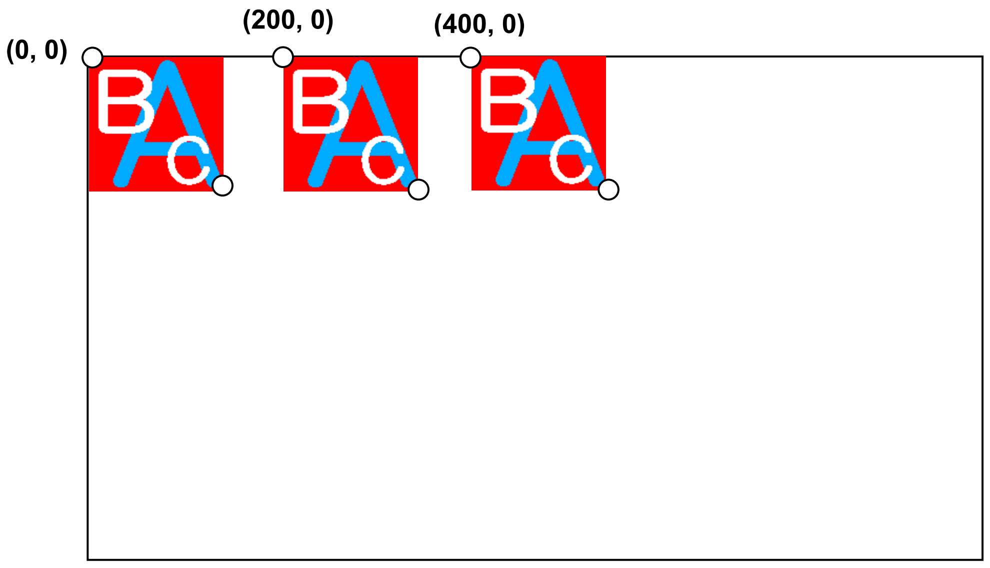


Figure 1.6 : 16 bits MCU and 24bpp color depth mode2 MCU Write data to SDRAM

Chapter 2 : Draw Functions

Overview:

In So many Human Machine Interface (HMI) applications, they often need to display some geometric figures for using as push button, sensor or the other specified symbols. If users want to support this display requirement through MCU firmware process, they must consume so many system resource or effort. RA8889 provides the geometric engine which can easy to achieve geometric display on th TFT-LCD by just a few commands. Users can even choose the color of geometric figure and should it be filled or not. The purpose of this application note will help users to know how useful RA8889 is.

2.1: Ellipse/Circle Input

RA8889 supports draw ellipse/circle drawing function makes user to draw ellipse/circle on the Display Data RAM only use by few MCU cycles. By setting the center of a ellipse/circle REG[7Bh~7Eh] ,the major and minor radius of a ellipse REG[77h~7Ah], the color of ellipse/circle REG[D2h~D4h], the draw ellipse/circle condition REG[76h] Bit5=0 and Bit4=0, and then setting start draw REG[76h] Bit7 = 1, RA8889 will draw a corresponding ellipse/circle on the Display Data RAM. Moreover, user can fill the circle by setting REG[76h] Bit6 = 1.

Note :The center of ellipse/circle should within active windows.
The procedure of drawing ellipse/circle just refers to the below figure:

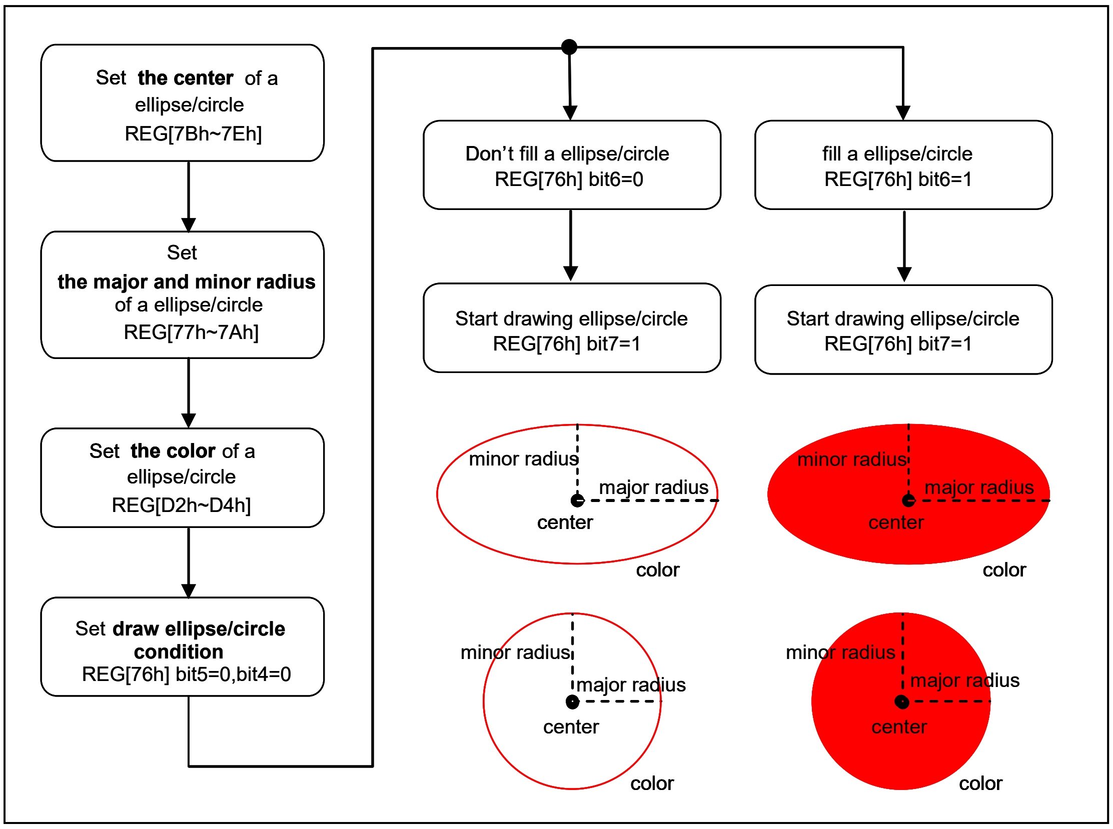

Set **the center** of a ellipse/circle REG[7Bh~7Eh]
Set **the major and minor radius** of a ellipse/circle REG[77h~7Ah]
Set **the color** of a ellipse/circle REG[D2h~D4h]
Set **draw ellipse/circle condition** REG[76h] bit5=0,bit4=0
Don't fill a ellipse/circle REG[76h] bit6=0
fill a ellipse/circle REG[76h] bit6=1
Start drawing ellipse/circle REG[76h] bit7=1
Start drawing ellipse/circle REG[76h] bit7=1
minor radius
major radius
center
color
minor radius
major radius
center
color
minor radius
major radius
center
color
minor radius
major radius
center
color

Figure 2.1

Draw Ellipse/Circle 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
)
```

Example 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_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);
```

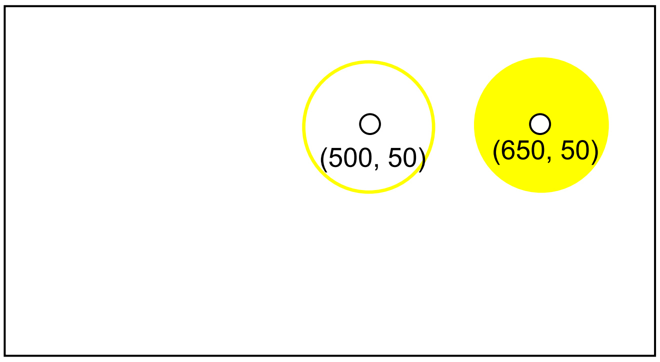


Figure 2.2 : Draw a Yellow Circle ,Center(500,50), R = 50, and draw a Yellow Circle Fill, Center(650,50), R = 50.

Example 2:

```
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);
```

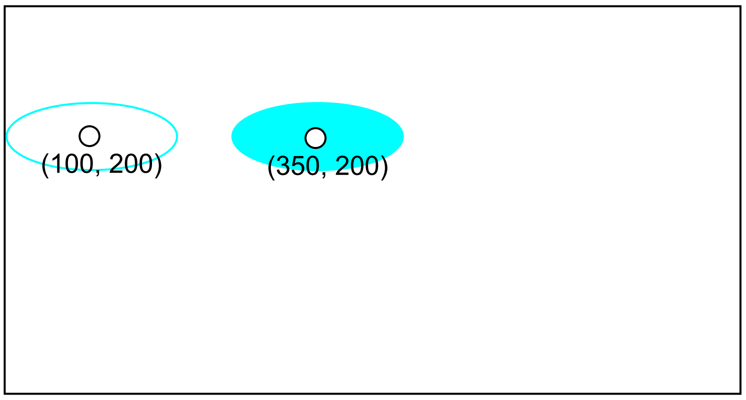


Figure 2.3 : Draw a blue-green Ellipse Center(100,200), X Radius = 100, Y Radius = 50, and Draw a blue-green Ellipse Fill Center(350,200), X Radius = 100, Y Radius = 50

2.2: Curve Input

RA8889 supports curve drawing function for user to draw curve on the Display Data RAM only by few MCU cycles. By setting the center of a curve REG[7Bh~7Dh] ,the major and minor radius of a curve REG[77h~7Ah], the color of curve REG[D2h~D4h], the draw curve condition REG[76h] Bit5=0 and Bit4=1, the curve part of the ellipse REG[76h] Bit[1:0], and then setting start draw REG[76h] Bit7 = 1, RA8889 will draw a corresponding curve on the Display Data RAM. Moreover, user can fill the curve by setting REG[76h] Bit6 = 1.

Note :the center of curve should within active windows.
The procedure of drawing curve just refers to the below figure:

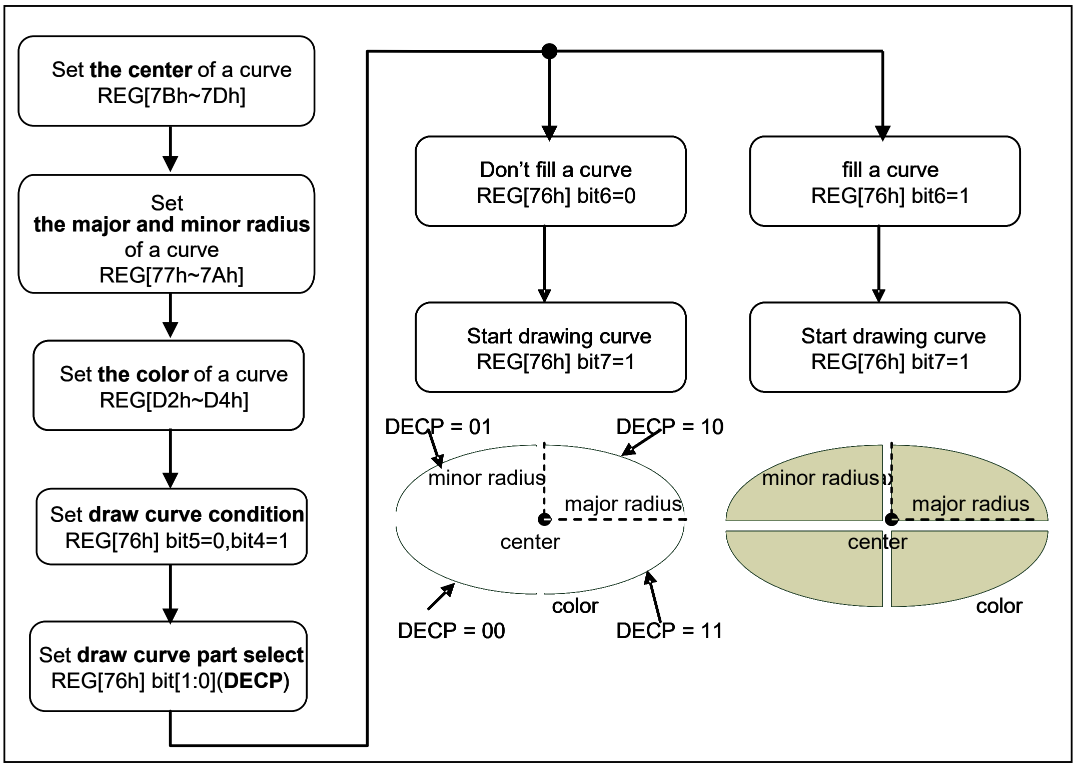

Set **the center** of a curve REG[7Bh~7Dh]
Set **the major and minor radius** of a curve REG[77h~7Ah]
Set **the color** of a curve REG[D2h~D4h]
Set **draw curve condition** REG[76h] bit5=0,bit4=1
Set **draw curve part select** REG[76h] bit[1:0](**DECP**)
Don't fill a curve REG[76h] bit6=0
fill a curve REG[76h] bit6=1
Start drawing curve REG[76h] bit7=1
Start drawing curve REG[76h] bit7=1
DECP = 01
DECP = 10
minor radius
major radius
center
color
DECP = 00
DECP = 11
minor radius
major radius
center
color

Figure 2-4

Draw Curve and Draw Curve Fill 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
)
```

Example :

```
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);
```

8 step for this program:

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**

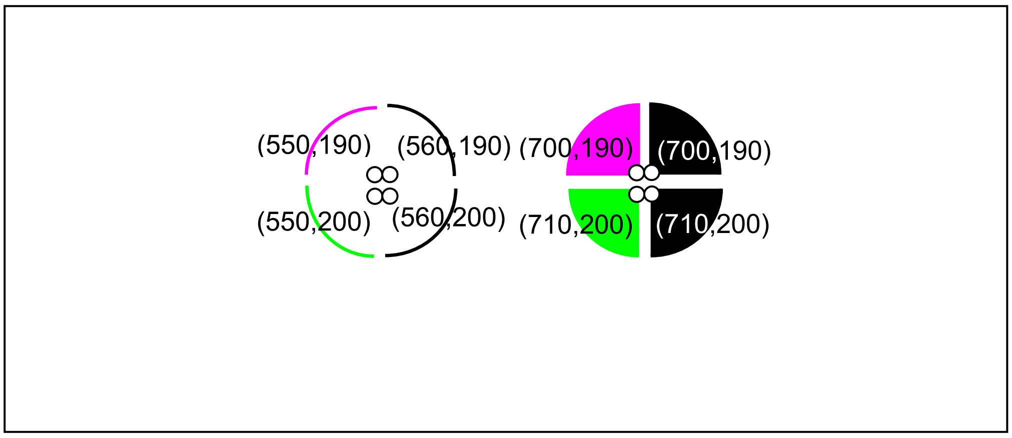


Figure 2.5 : Draw curve example.

2.3:Square Input

RA8889 supports square drawing function for user to draw square on the Display Data RAM only by few MCU cycles. By setting the start point of a square REG[68h~6Bh] ,the end point of a square REG[6Ch~6Fh] and the color of a square REG[D2h~D4h], then setting draw a square REG[76h] Bit5=1, Bit4=0 and start draw REG[76h] Bit7 = 1, RA8889 will draw a corresponding square on the Display Data RAM. Moreover, user can fill the square by setting REG[76h] Bit6 = 1.

Note：the start point and the end point of a square should within active windows.

The procedure of drawing square just refers to the below figure:

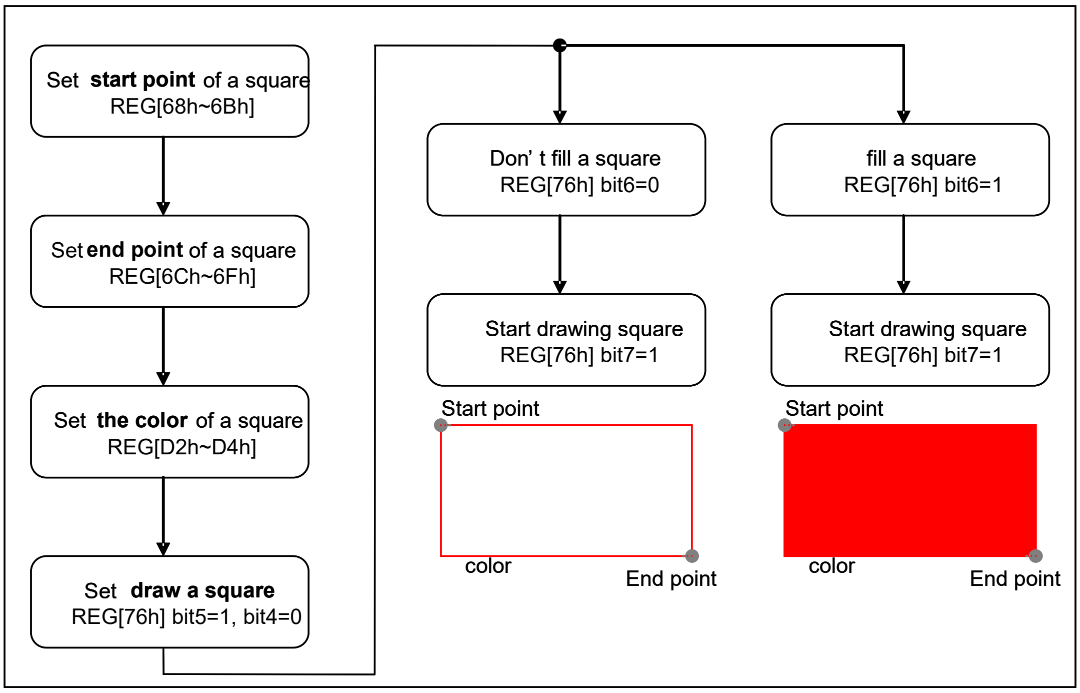


Figure 2.6 : Geometric Pattern Drawing- Draw Rectangle

Draw Square 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

)
```

Example 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);
```

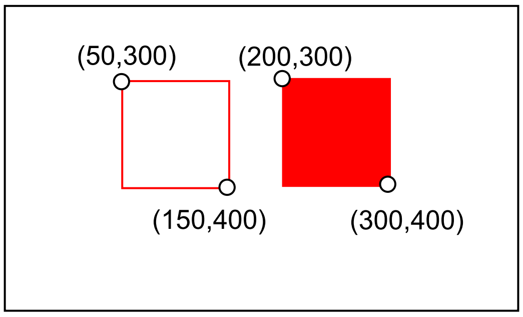


Figure 2.7 : Draw a red Square from Point1(50,300) to Point2(150,400) and Draw a red Square Fill from Point1(200,300) to Point2(300,400)

2.4:Line Input

RA8889 supports line drawing function for user to draw line on the Display Data RAM only by few MCU cycles. By setting the start point of a line REG[68h~6Bh] ,the end point of a line REG[6Ch~6Fh] and the color of a line REG[D2h~D4h], then setting draw a line REG[67h] Bit1 = 0 and start draw REG[67h] Bit7 = 1, RA8889 will draw a corresponding line on the Display Data RAM.

Note : the start point and the end point of line should within active windows.

The procedure of drawing line just refers to the below figure:

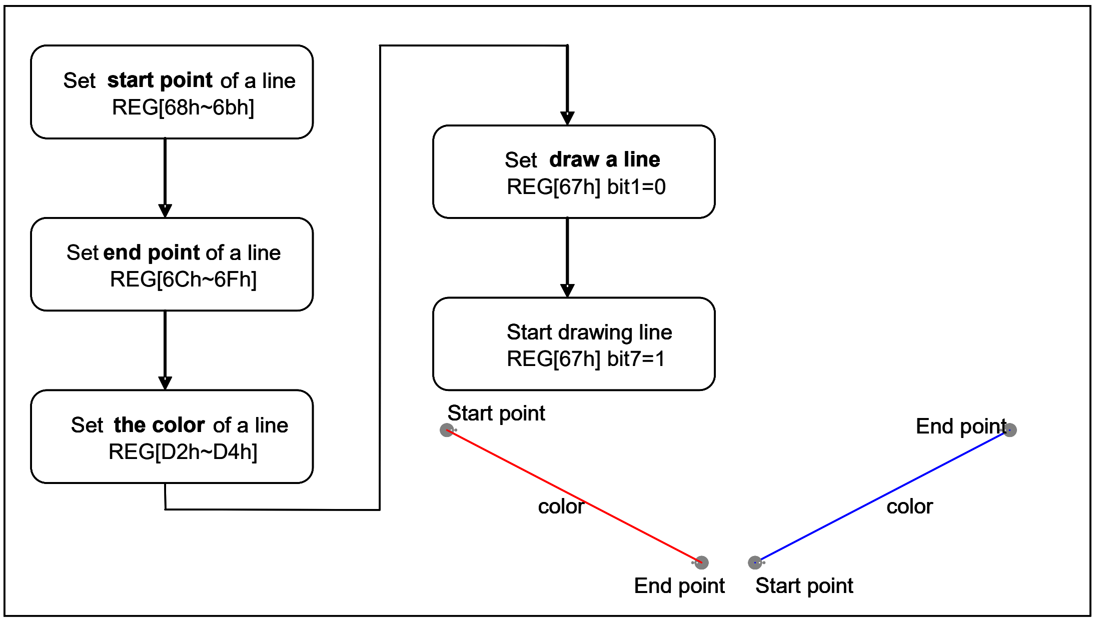


Figure 2.8: Geometric Pattern Drawing- Draw Line

Draw Line 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

)
```

Example :

```
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);
```

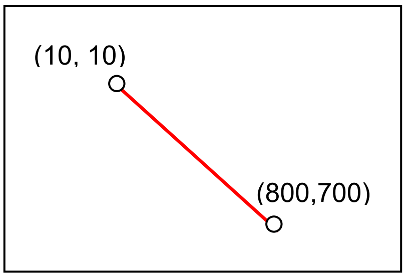


Figure 2.9 : Draw a red Line from Point1(10,10) to Point2(800,700).

2.5: Triangle Input

RA8889 supports triangle drawing function for user to draw line on the Display Data RAM only by few MCU cycles. By setting the point0 of a triangle REG[68h~6Bh], the point1 of a triangle REG[6Ch~6Fh], the point2 of a triangle REG[70h~73h] and the color of a triangle REG[D2h~D4h], then setting draw a triangle REG[67h] Bit1 = 1 and start draw REG[67h] Bit7 = 1, RA8889 will draw a corresponding triangle on the Display Data RAM. Moreover, user can fill the triangle by setting REG[67h] Bit5 = 1.

Note : the point0, point1 and point2 of triangle should within active windows.
The procedure of drawing triangle just refers to the below figure:

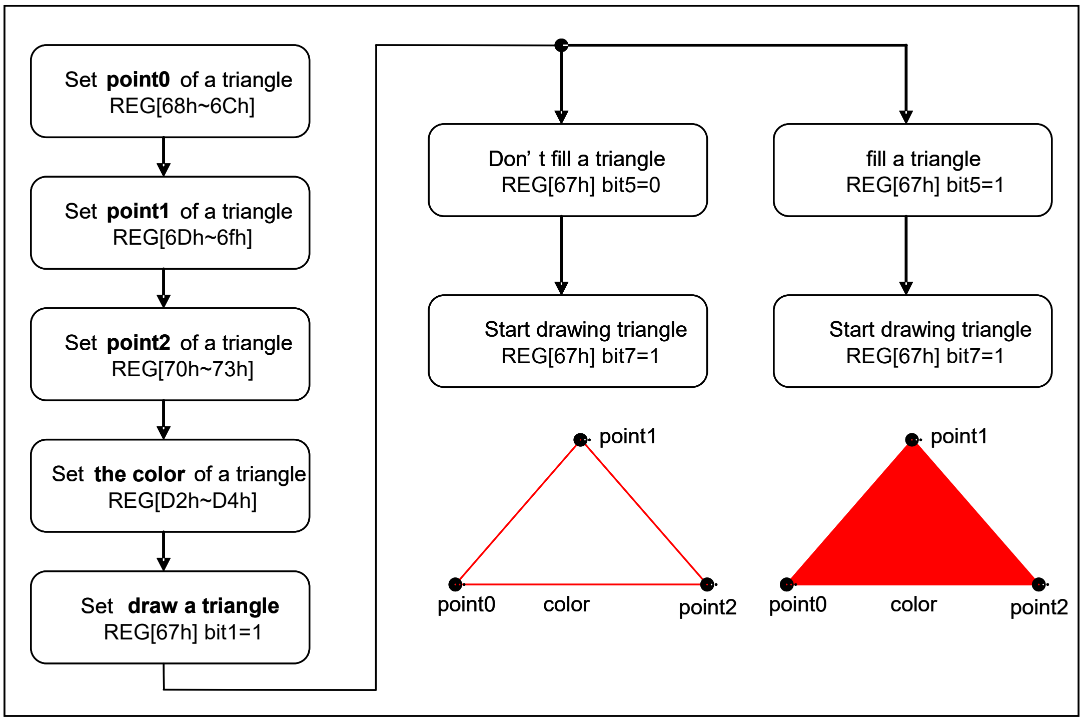

Set **point0** of a triangle REG[68h~6Ch]
Set **point1** of a triangle REG[6Dh~6fh]
Set **point2** of a triangle REG[70h~73h]
Set **the color** of a triangle REG[D2h~D4h]
Set **draw a triangle** REG[67h] bit1=1
Don’ t fill a triangle REG[67h] bit5=0
fill a triangle REG[67h] bit5=1
Start drawing triangle REG[67h] bit7=1
Start drawing triangle REG[67h] bit7=1
point1
point0
color
point2
point1
point0
color
point2

Figure 2.10 : Geometric Pattern Drawing- Draw Triangle

Draw Triangle 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
)
```

Example 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);
```

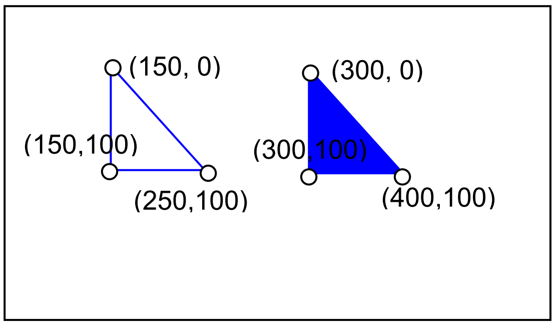


Figure 2.11 : Draw a blue Triangle by Point1(150,0) ,Point2(150,100) and Point3(250,100) and draw a blue Triangle Fill by Point1(300,0) ,Point2(300,100) and Point3(400,100)

2.6: Square Of Circle Corner Input

RA8889 supports circle-square drawing function for user to draw circle square on the Display Data RAM by few MCU cycles. By setting the start point of a square REG[68h~6Ch] ,the end point of a square REG[6Dh~6Fh], the major and minor radius of a ellipse/circle REG[77h~7Ah] and the color of a circle square REG[D2h~D4h], then setting draw a circle square REG[76h] Bit5=1, Bit4=1 and start draw REG[76h] Bit7 = 1, RA8889 will draw a corresponding circle square on the Display Data RAM. Moreover, user can fill the square by setting REG[76h] Bit6 = 1.

Note1 : (End point X – Start point x) must large than (2*major radius + 1) and (End point Y – Start point Y) must large than (2*minor radius + 1)

Note2 :the start point and the end point of a square should be within active windows.

The procedure of drawing square just refers to the below figure:

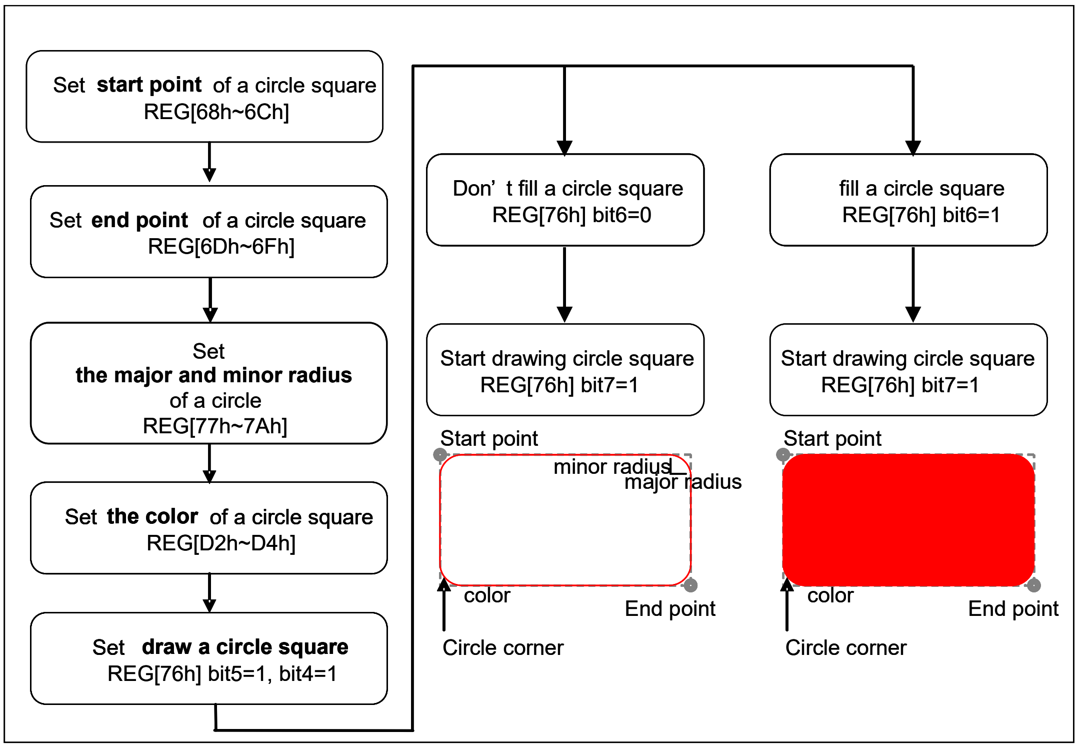


Figure 2.12 : Geometric Pattern Drawing- Draw Circle-Square

Draw Square Of Circle Corner 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
)
```

Example 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);

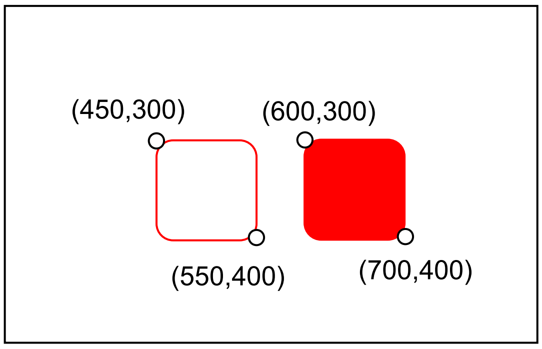


Figure 2.13 : Draw a red Square Of Circle Corner from Point(450,300) to Point(550,400) with X_R = 20,Y_R=30, and draw a red Square Of Circle Corner Fill from Point(600,300) to Point(700,400) with X_R =20,Y_R=30.

Chapter 3 : Serial Flash Control Unit

RA8889 builds in a SPI master interface for accessing the external Serial Flash/ROM, supporting for protocol of 4-BUS (Normal Read), 5-BUS (FAST Read), Dual mode 0, Dual mode 1 with Mode 0/Mode 3 and Quad mode. Serial Flash/ROM function can be used for FONT mode and DMA mode. FONT mode means that the external serial Flash/ROM is treated as a source of character bitmap. To support the most useful characters, RA8889 is compatible with the character ROM of professional font vendor—Genitop Inc. in Shanghai. DMA mode means that the external Flash/ROM is treated as the data source of DMA (Direct Memory Access). User can speed up the data transfer to display memory and need not MCU intervene by this mode.

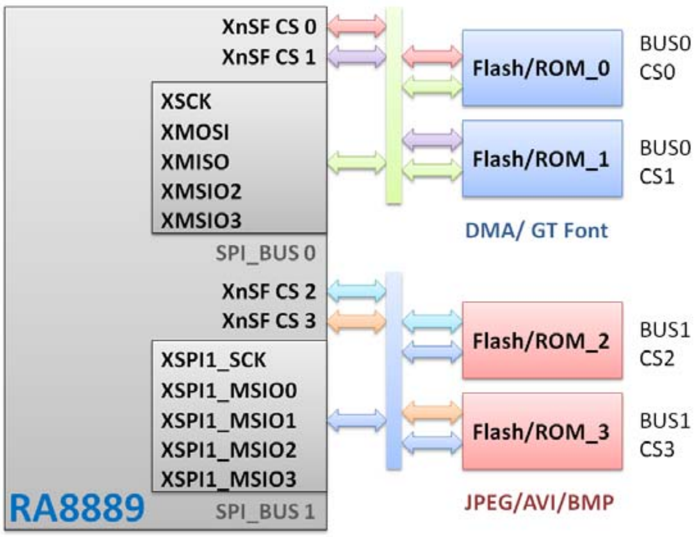


Figure 3.1: RA8889 Serial Flash/ROM System

Table 3-1 : Read Command Code & Behavior Selection

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

Then DMA block mode is used for moving graphic function from external serial flash memory to the display memory (SDRAM) of RA8889. The process unit of DMA function is pixel. Regarding the flowchart of DMA function, please refer to the description as below.

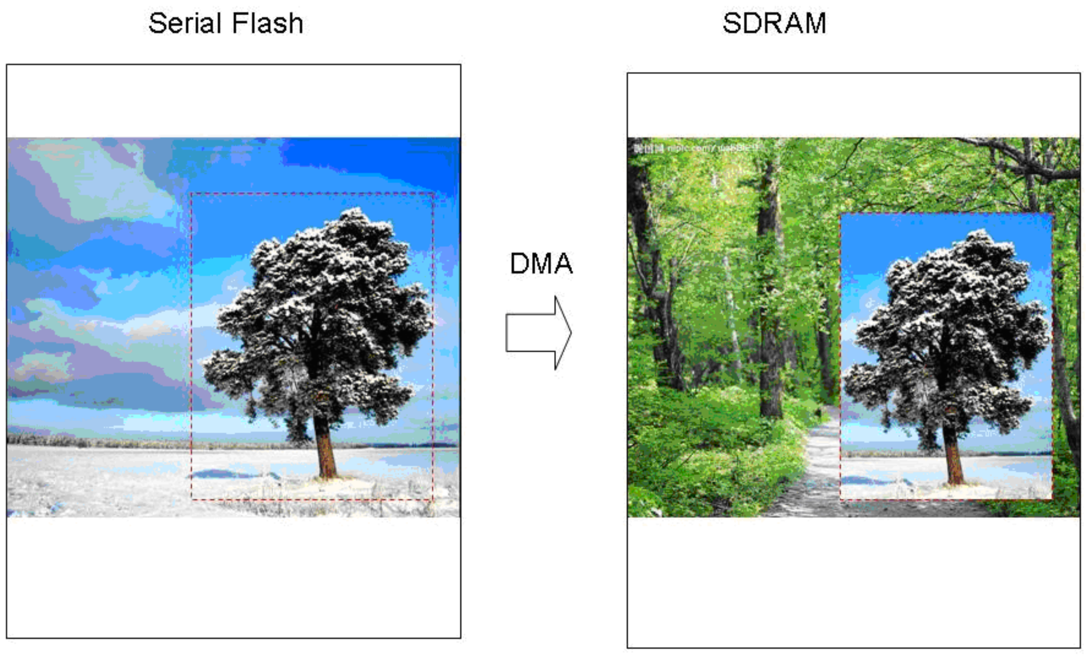


Figure 3.2: DMA Function

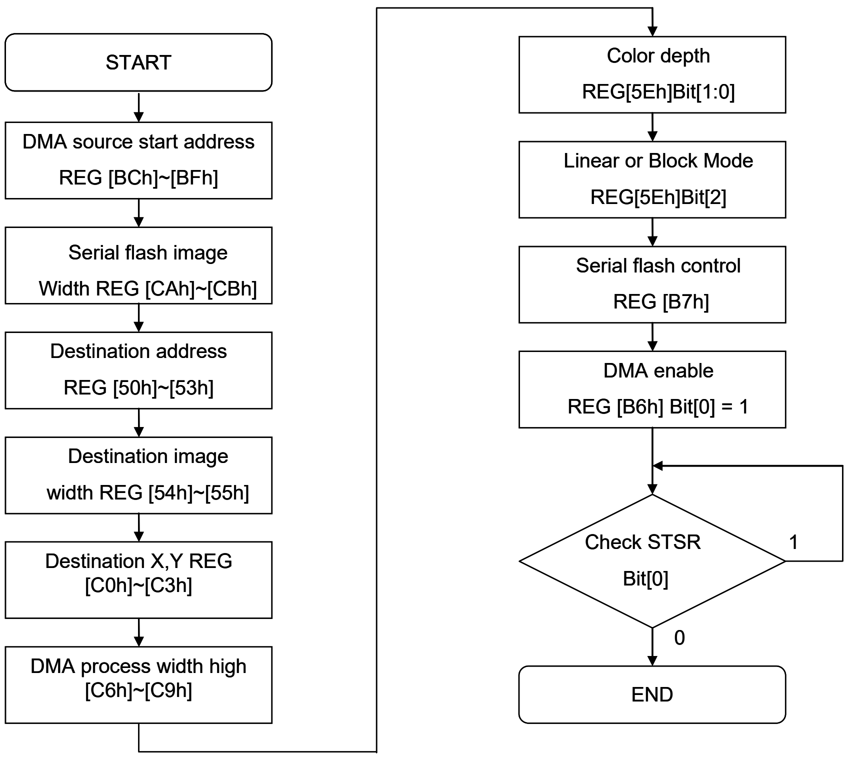

START
DMA source start address REG [BCh]~[BFh]
Serial flash image Width REG [CAh]~[CBh]
Destination address REG [50h]~[53h]
Destination image width REG [54h]~[55h]
Destination X,Y REG [C0h]~[C3h]
DMA process width high [C6h]~[C9h]
Color depth REG[5Eh]Bit[1:0]
Linear or Block Mode REG[5Eh]Bit[2]
Serial flash control REG [B7h]
DMA enable REG [B6h] Bit[0] = 1
Check STSR Bit[0]
1
0
END

Figure 3.3 : Enable DMA Procedure – Check Flag

3.1.2 : API Example and Display results on LCD for the DMA Function

We provider 7 API for user as below:

```
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 : Only for 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
)
```

Example :

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.

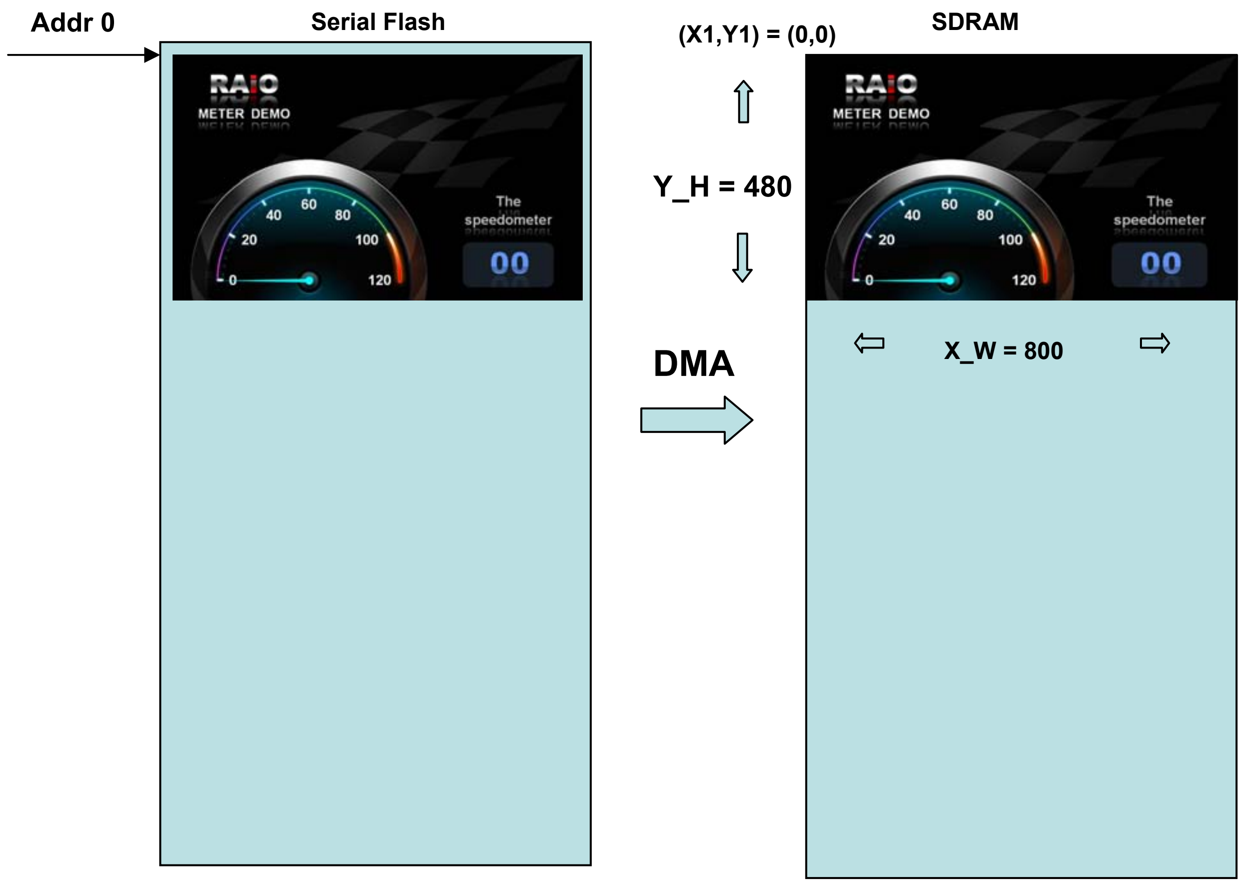


Figure 3.4: Using DMA Write data into SDRAM from Serial Flash

3.2.1 :Media Decode Unit

RA8889 provides media decode unit, which supports JPEG (ISO/IEC 10918-1 Baseline profile, YUV444, YUV422, YUV420, YUV400, and not support restart interval format), BMP (raw data), and AVI (motion jpeg) formats. RA8889 can auto distinguish the above three formats and auto parse them to the relative decoder. In video functions, RA8889 provides auto play, pause, and stop functions. User should load images or videos into serial flash in advance and show them on LCD screen via setting DMA, CANVAS and PIP relative registers.

According to the address for image write, please refer to CANVAS relative registers. Because video is displayed on PIP1 or PIP2 window, user should set the PIP relative registers before play video. Besides, RA8889 also provides interrupt and busy flags for check.

Notice,
1. For serial flash interface, please use quad mode, and the frequency of core clock is recommended above 100MHz.
2. AVI frame rate can be 30, 29.97, 25, 24, 23.97, 20, and 15.
3. The color depth of PIP should be consistent with that of Main Window.
4. The width and height of AVI/JPG must be 8 multiples.
5. The DMA length of serial flash should equal to the file size of image or video.

3.2.2 The Decode Flow Of Image In Hardware

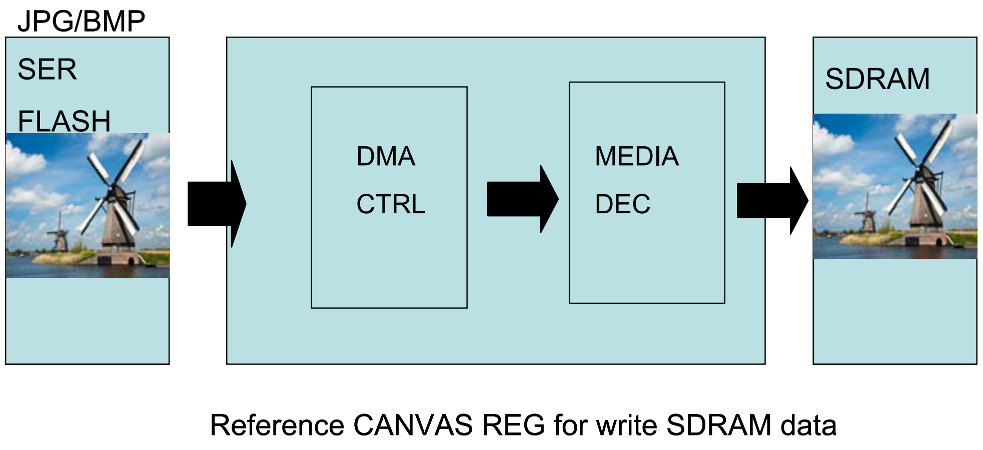


Figure 3-5

3.2.3 The Flow Chart For Image Decode

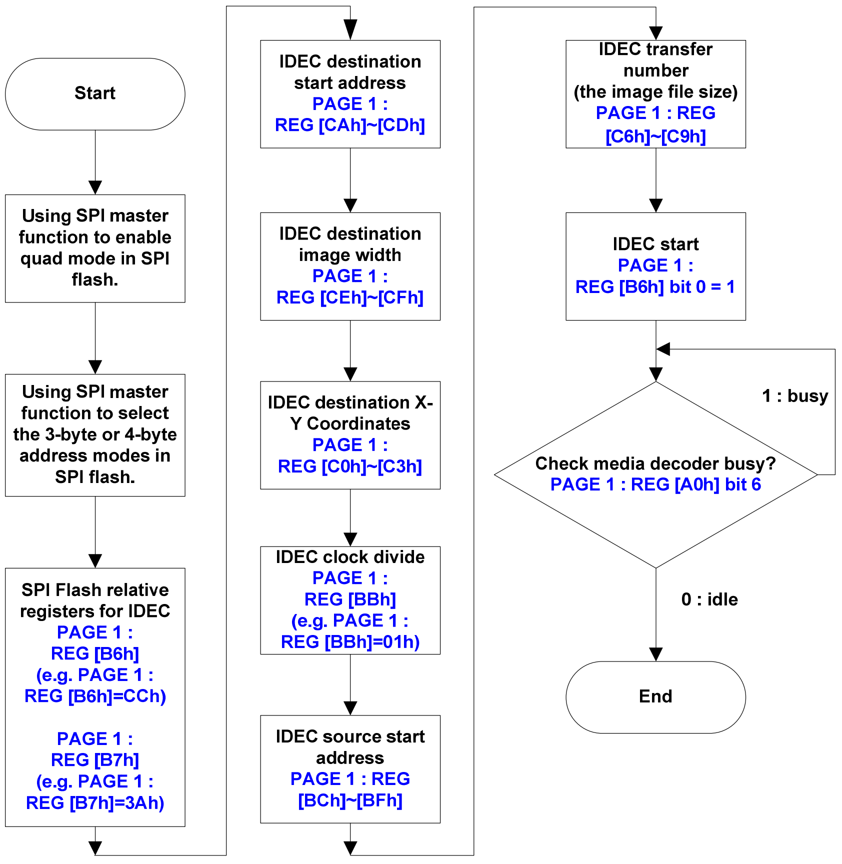
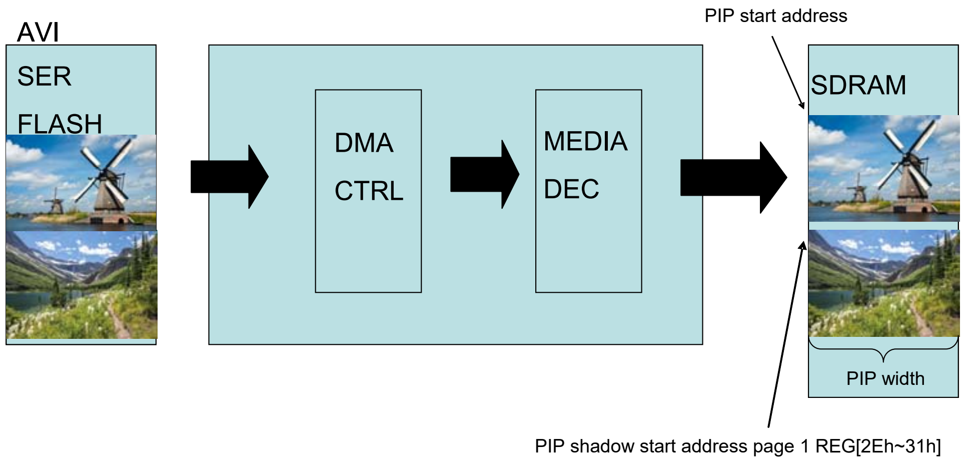

Start
Using SPI master function to enable quad mode in SPI flash.
Using SPI master function to select the 3-byte or 4-byte address modes in SPI flash.
SPI Flash relative registers for IDEC PAGE 1 : REG [B6h] (e.g. PAGE 1 : REG [B6h]=CCh) PAGE 1 : REG [B7h] (e.g. PAGE 1 : REG [B7h]=3Ah)
IDEC destination start address PAGE 1 : REG [CAh]~[CDh]
IDEC destination image width PAGE 1 : REG [CEh]~[CFh]
IDEC destination X-Y Coordinates PAGE 1 : REG [C0h]~[C3h]
IDEC clock divide PAGE 1 : REG [BBh] (e.g. PAGE 1 : REG [BBh]=01h)
IDEC source start address PAGE 1 : REG [BCh]~[BFh]
IDEC transfer number (the image file size) PAGE 1 : REG [C6h]~[C9h]
IDEC start PAGE 1 : REG [B6h] bit 0 = 1
1 : busy
Check media decoder busy? PAGE 1 : REG [A0h] bit 6
0 : idle
End

Figure 3-6

Note : IDEC means image format supported by media decoding unit (JPEG, BMP and AVI)

3.2.4 The Decode Flow Of AVI In Hardware



Reference PIP REG for write SDRAM data

Figure 3-7

3.2.5 The Flow Chart For AVI Decode

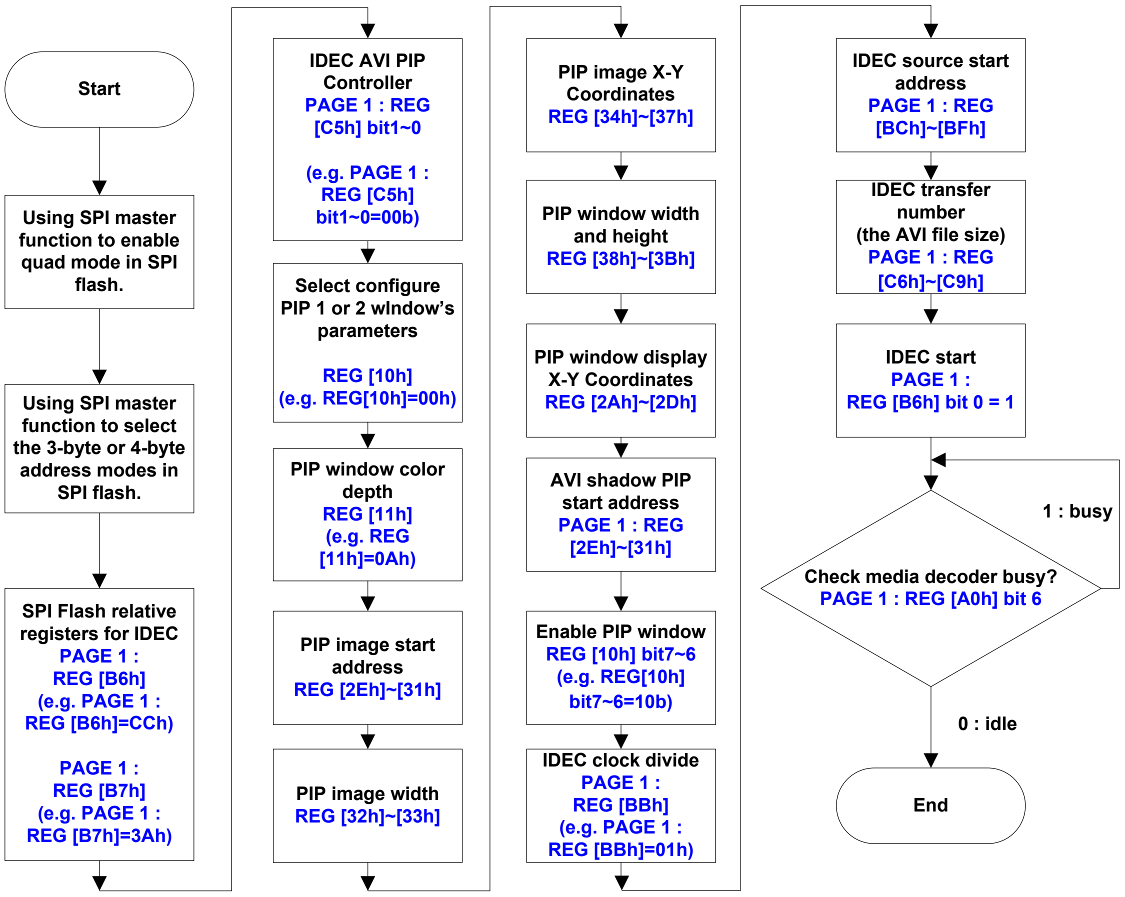

Start
Using SPI master function to enable quad mode in SPI flash.
Using SPI master function to select the 3-byte or 4-byte address modes in SPI flash.
SPI Flash relative registers for IDEC PAGE 1 : REG [B6h] (e.g. PAGE 1 : REG [B6h]=CCh) PAGE 1 : REG [B7h] (e.g. PAGE 1 : REG [B7h]=3Ah)
IDEC AVI PIP Controller PAGE 1 : REG [C5h] bit1~0 (e.g. PAGE 1 : REG [C5h] bit1~0=00b)
Select configure PIP 1 or 2 wIndow's parameters REG [10h] (e.g. REG[10h]=00h)
PIP window color depth REG [11h] (e.g. REG [11h]=0Ah)
PIP image start address REG [2Eh]~[31h]
PIP image width REG [32h]~[33h]
PIP image X-Y Coordinates REG [34h]~[37h]
PIP window width and height REG [38h]~[3Bh]
PIP window display X-Y Coordinates REG [2Ah]~[2Dh]
AVI shadow PIP start address PAGE 1 : REG [2Eh]~[31h]
Enable PIP window REG [10h] bit7~6 (e.g. REG[10h] bit7~6=10b)
IDEC clock divide PAGE 1 : REG [BBh] (e.g. PAGE 1 : REG [BBh]=01h)
IDEC source start address PAGE 1 : REG [BCh]~[BFh]
IDEC transfer number (the AVI file size) PAGE 1 : REG [C6h]~[C9h]
IDEC start PAGE 1 : REG [B6h] bit 0 = 1
Check media decoder busy? PAGE 1 : REG [A0h] bit 6
1 : busy
0 : idle
End

Figure 3-8

Note : IDEC means image format supported by media decoding unit (JPEG, BMP and AVI)

3.2.6 API Example and Display results on LCD for the Media Decode Function:

We provider 7 API for user as below:

```
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: Only support 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 Example :

```
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).

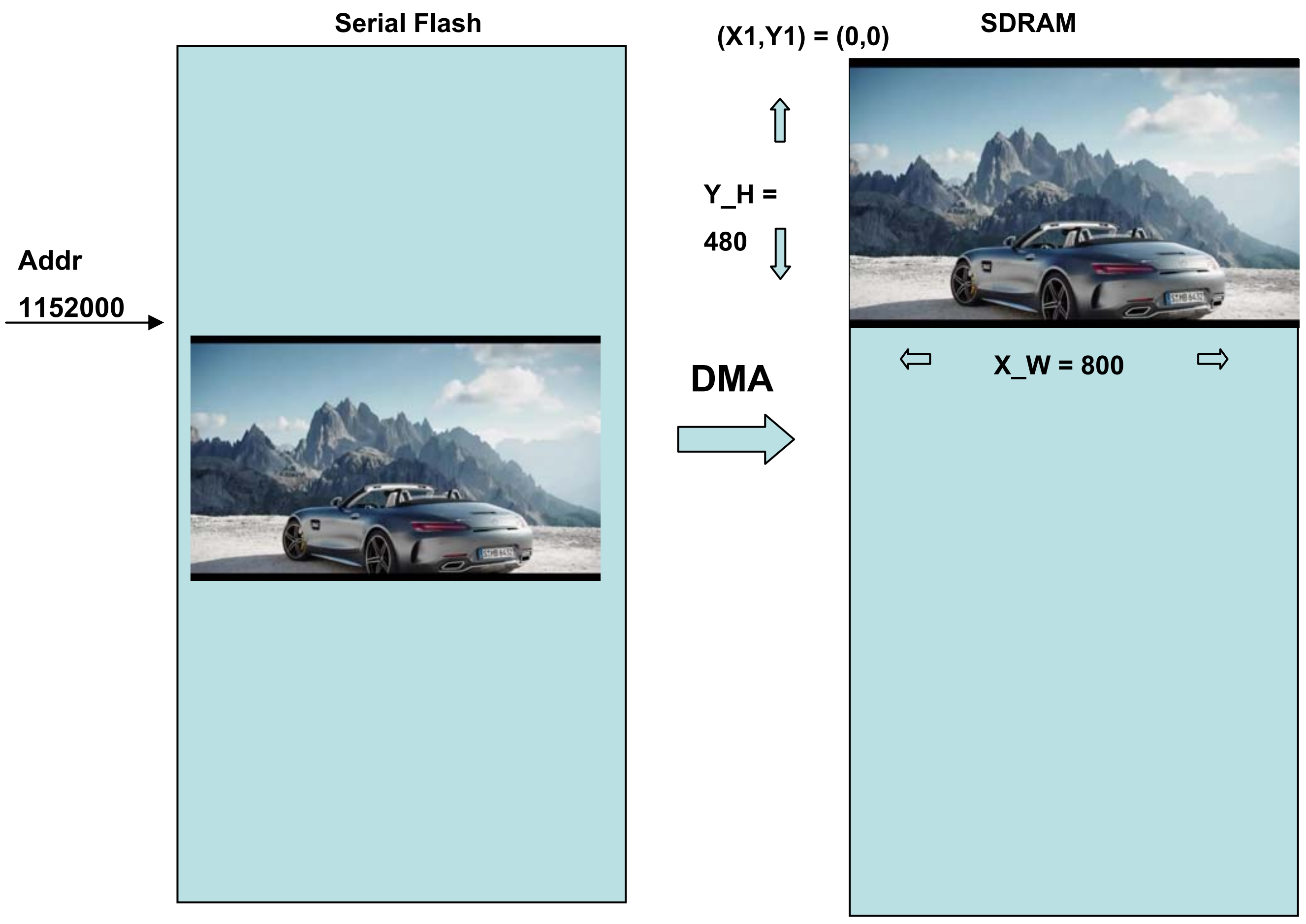


Figure 3.9: Using Media decoder load JPG pic into SDRAM from Serial Flash

AVI Example :

```
    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

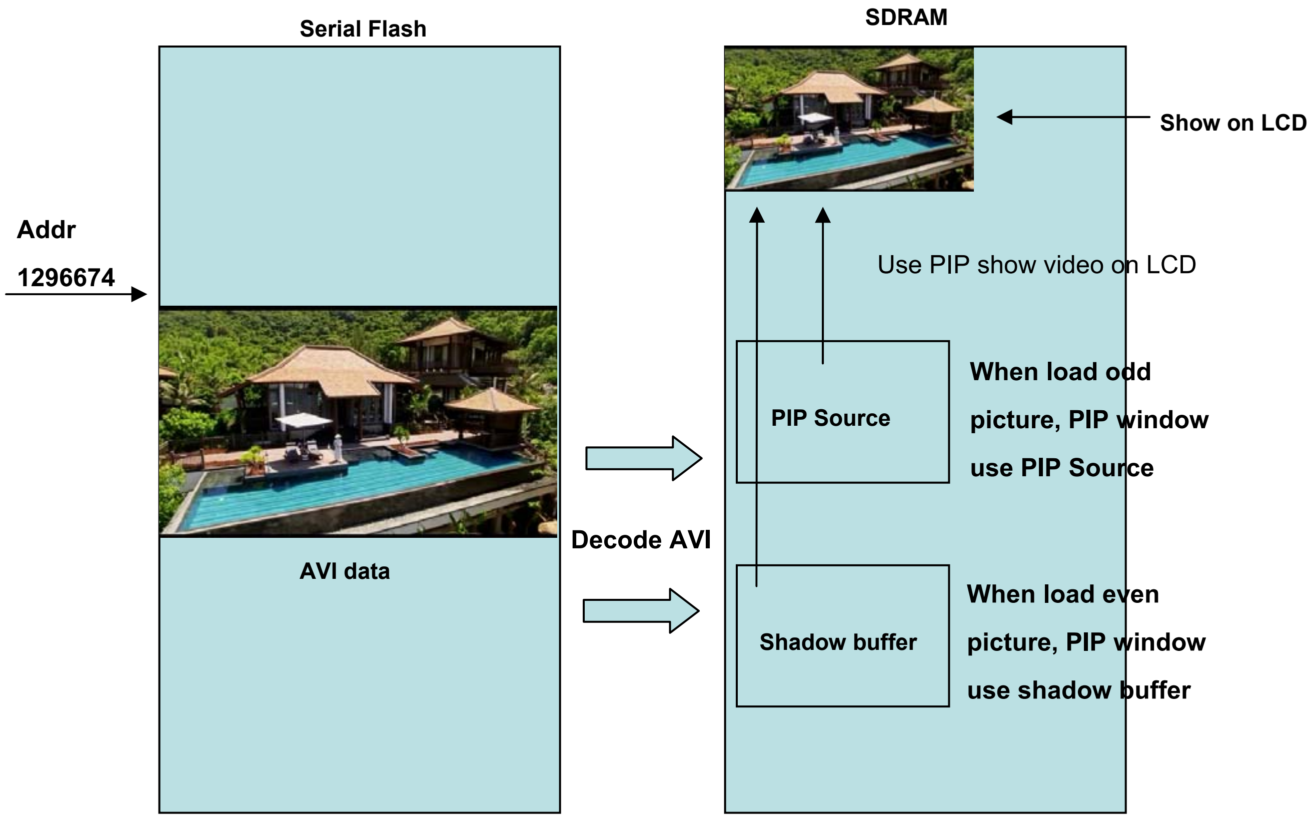

Serial Flash
SDRAM
Show on LCD
Addr
1296674
Use PIP show video on LCD
PIP Source
When load odd picture, PIP window use PIP Source
Decode AVI
AVI data
Shadow buffer
When load even picture, PIP window use shadow buffer

Figure 3.10: Using Media decoder load vedio into SDRAM from Serial Flash

Chapter 4 : Block Transfer Engine

Overview:

The RA8889 embedded a built-in 2D Block Transfer Engine(BTE) which can increase the performance of block transfer operation. When a block of data needs to be moved or do some logic operation with dedicated data, RA8889 can speed up the operation by BTE hardware and also simplify the MCU program. This section will discuss the BTE engine operation and functionality.

Before using the BTE function, user must select the corresponding BTE operation. RA8889 supports 13 BTE operations. About the operation description, please refer to Table 4-1. For each BTE operation, maximum 16 raster operations (ROP) are supported for different application. They could provide the different logic combinations for ROP source and ROP destination. Through the combination of the BTE operation and ROP, user can achieve many useful application operations. Please refer to the behind chapters for detail description.

This application note will focus on some of the BTE functions. e.g. **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.** If our customer needs the illustration for the other BTE function, please contact with RAiO's Sales or distributor.

Table 4-1 : BTE Operation Function

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

Table 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 Function S0: Source 0 Data, S1: Source 0 Data, D: Destination Data.
2. For pattern fill functions, the source data indicates the pattern data.

Example:

If ROP function setting Ch(1100b), then Destination Data = Source 0 Data
If ROP function setting Eh(1110b), then Destination Data = S0 + S1
If ROP function setting 2h(0010b), then Destination Data = ~S0・S1
If ROP function setting Ah(1010b), then Destination Data = Source 1 Data

BTE Access Memory Method

With the setting, The BTE memory source/destination data is treated as a block of display area .The below example shows source 0 / source 1 / destination address are defined as block access method.

BTE Chroma Key (Transparency Color) Compare

In BTE Chroma Key (Transparency color) function Enable, BTE process compare source 0 data and background color register data. If data equal then not change destination data otherwise write source 0 data to destination.

In source color depth = 256 color,
  Source 0 red color just only compare REG [D5h] Bit [7:5],
  Source 0 green color just only compare REG [D6h] Bit [7:5],
  Source 0 blue color just only compare REG [D7h] Bit [7:6]
In source color depth = 65k color,
  Source 0 red color just only compare REG [D5h] Bit [7:3],
  Source 0 green color just only compare REG [D6h] Bit [7:2],
  Source 0 blue color just only compare REG [D7h] Bit [7:3]
In source color depth = 16.7M color,
  Source 0 red color just only compare REG[D5h] Bit [7:0],
  Source 0 green color just only compare REG [D6h] Bit [7:0],
  Source 0 blue color just only compare REG [D7h] Bit [7:0]

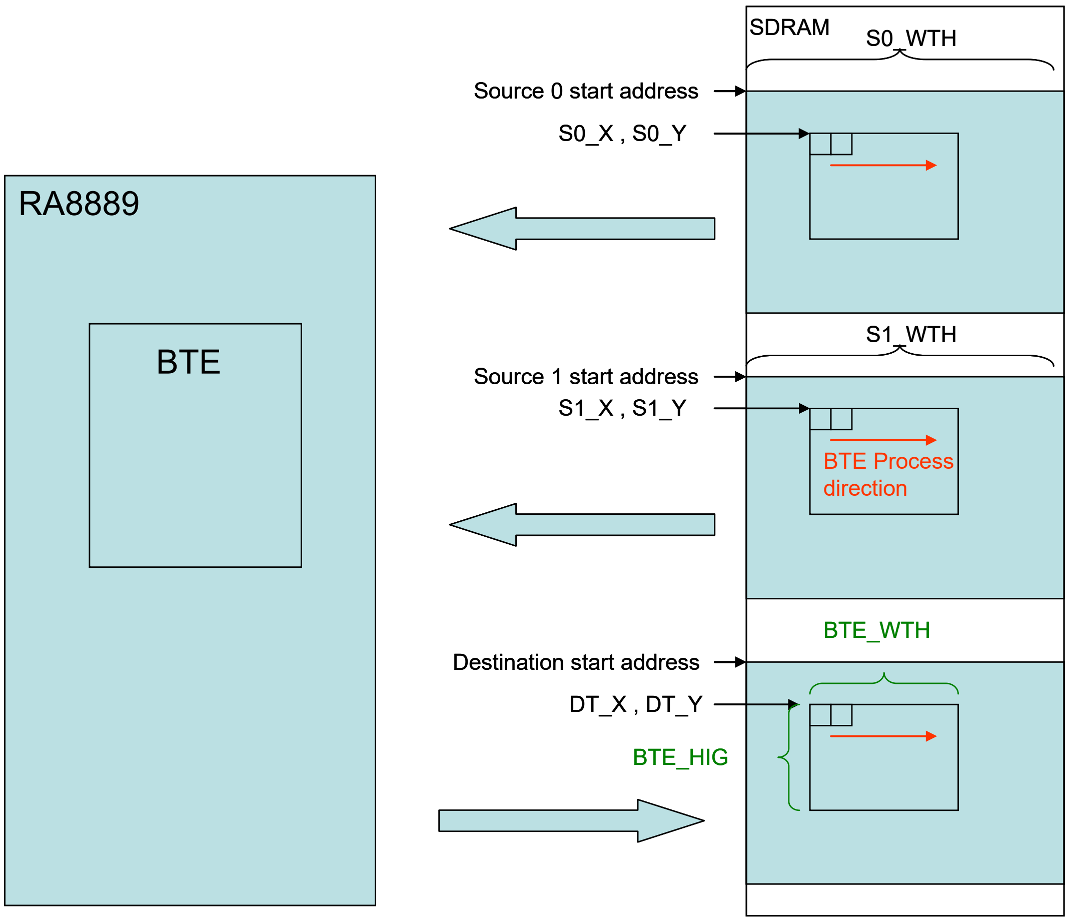


Figure 4-1 : Memory Access of BTE Function

4.1.1: Solid Fill

The Solid Fill BTE fills a rectangular area of the SDRAM with a solid color. This operation is used to paint large screen areas or to set areas of the SDRAM to be a given value. The color of Solid Fill is set by “BTE Foreground Color”.

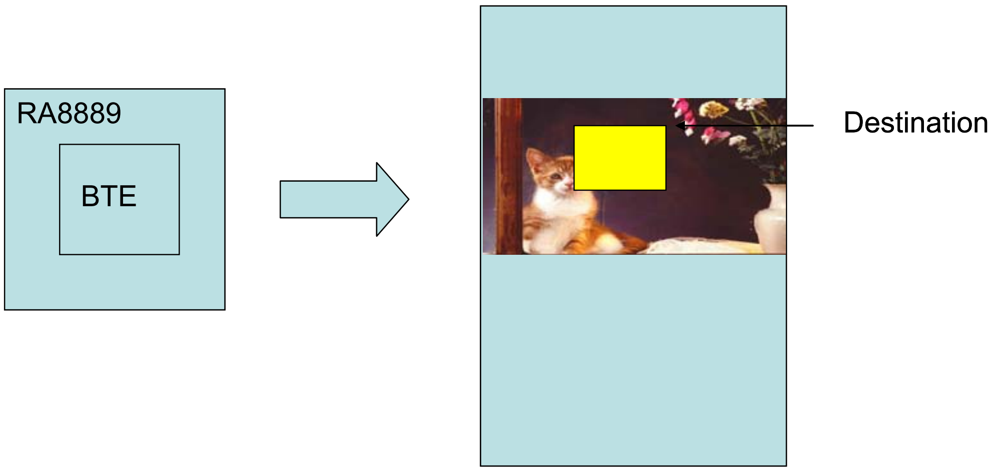


Figure 4-2 : Hardware Data Flow

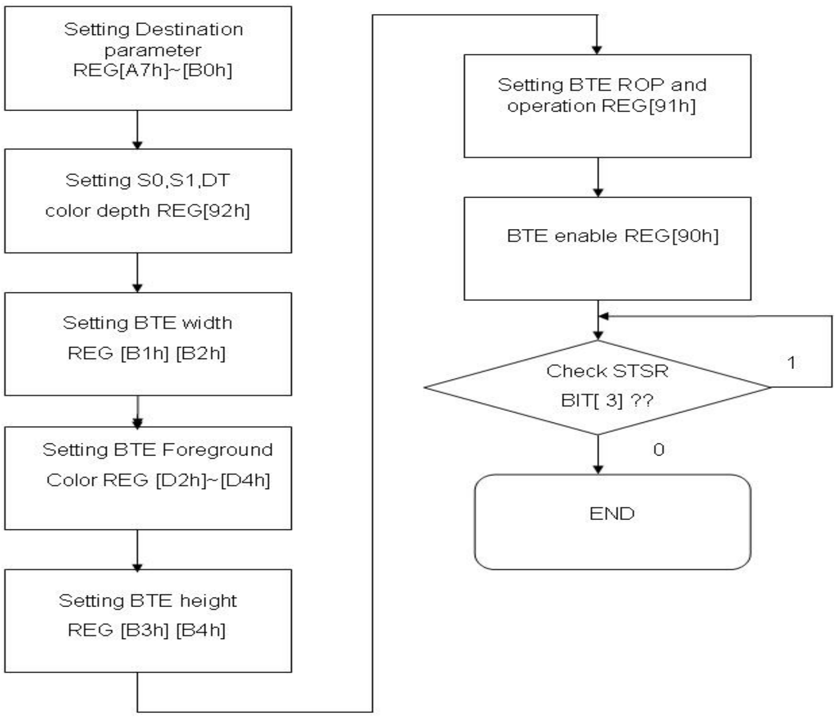

Setting Destination parameter REG[A7h]~[B0h]
Setting S0,S1,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE Foreground Color REG [D2h]~[D4h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Check STSR BIT[3] ??
1
0
END

Figure 4-3 : Flow Chart

4.1.2. Display results on LCD for the BTE Solid Fill Functions:

The following is the API program and examples illustrate, Figure 4-4 for the use of Solid Fill function fills a red square blocks.

```
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
)
```

Example:

```
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
```

Condition:

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

This program result:

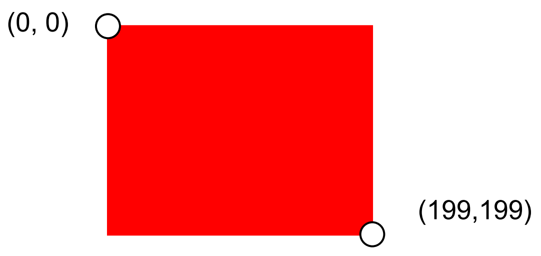


Figure 4-4 : Use Solid Fill to fill [(0, 0) ~ (199,199)] by red color

4.2.1: Pattern Fill with ROP

"Pattern Fill with ROP" operation fills a specified rectangular area of the SDRAM with a dedicated pattern repeatedly. The fill pattern is an array of 8x8/16x16 pixels stored in the SDRAM. The pattern can be logically combined with the destination using one of the 16 ROP codes. The operation can be used to speed up the application with duplicate pattern write into an area, such as background paste function.

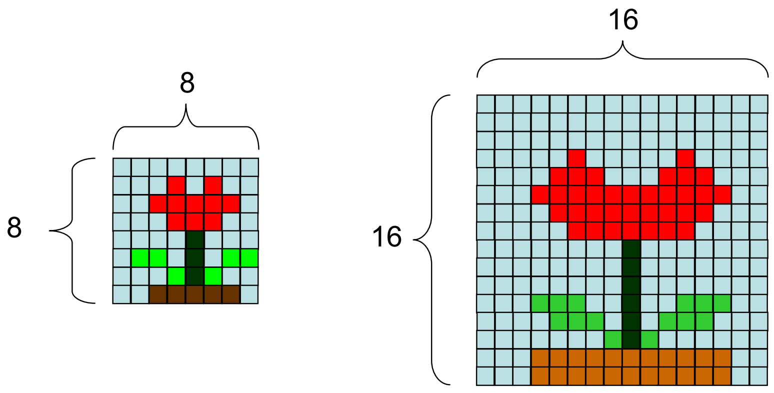


Figure 4-5 : Pattern Format

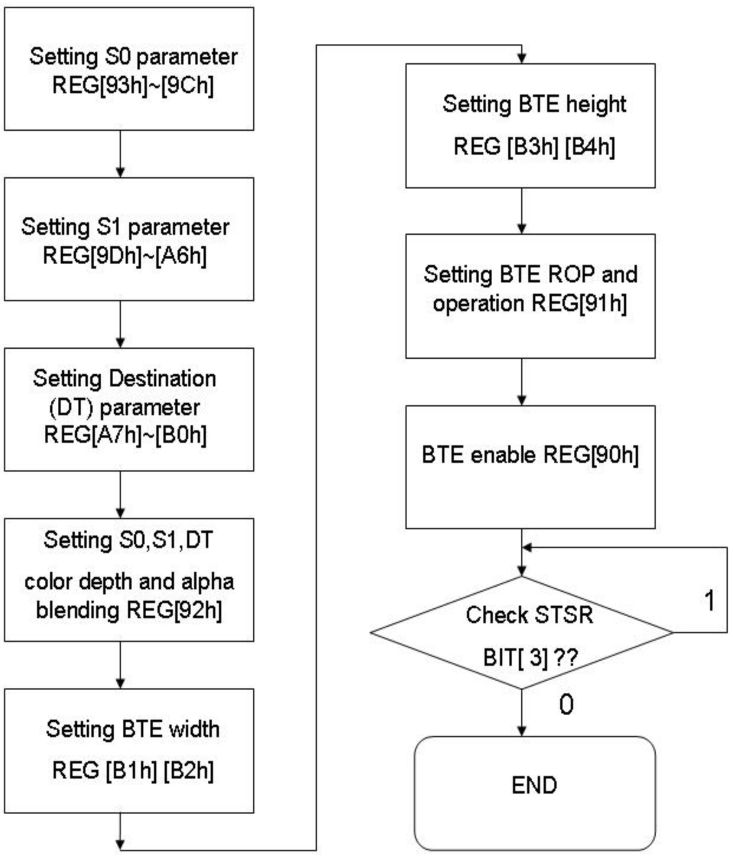

Setting S0 parameter REG[93h]~[9Ch]
Setting S1 parameter REG[9Dh]~[A6h]
Setting Destination (DT) parameter REG[A7h]~[B0h]
Setting S0,S1,DT color depth and alpha blending REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Check STSR BIT[3] ??
1
0
END

Figure 4-6 : Flow Chart

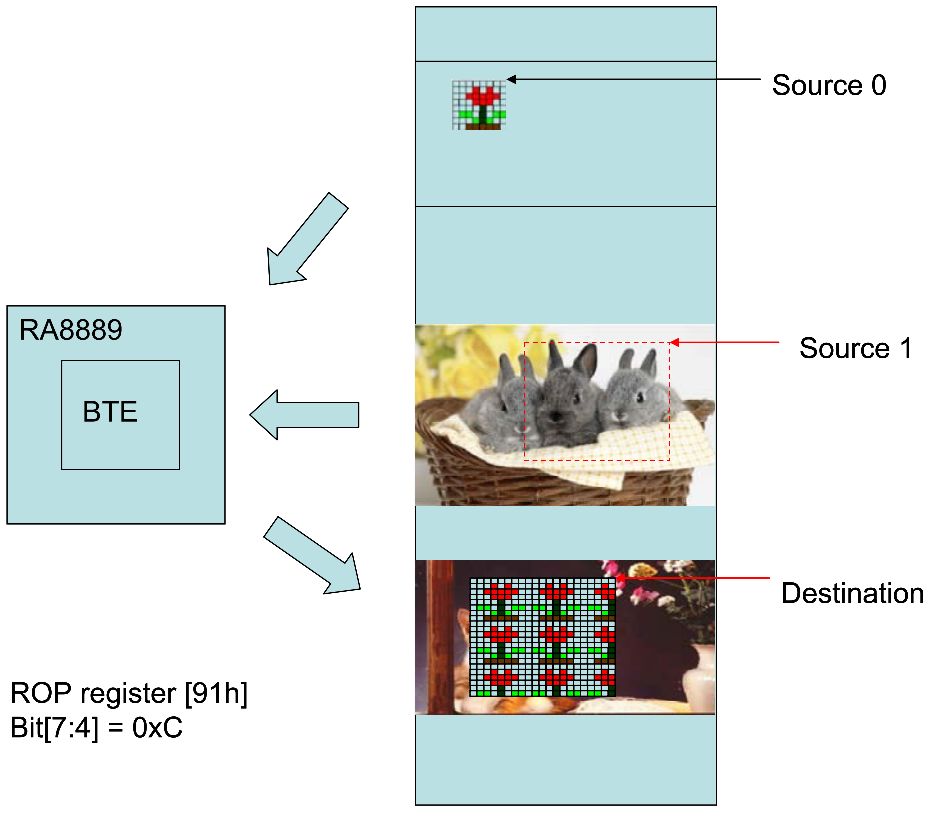


Figure 4-7 : Hardware Data Flow

4.2.2. Display results on LCD for the BTE Pattern Fill Function:

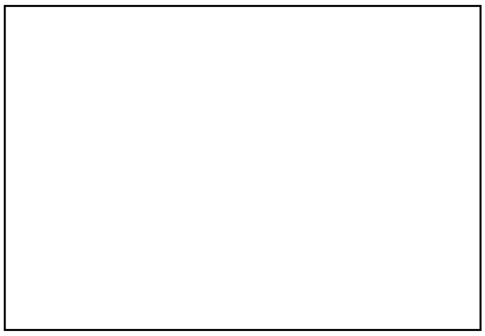


Figure 4-8: SDRAM current data in this example

Figure 4-9 (16x16) Icon

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
)
```

Example:

```
    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);
```

Condition:

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

Step 1 : Execute DMA function copying a picture into SDRAM from external Serial Flash Memory

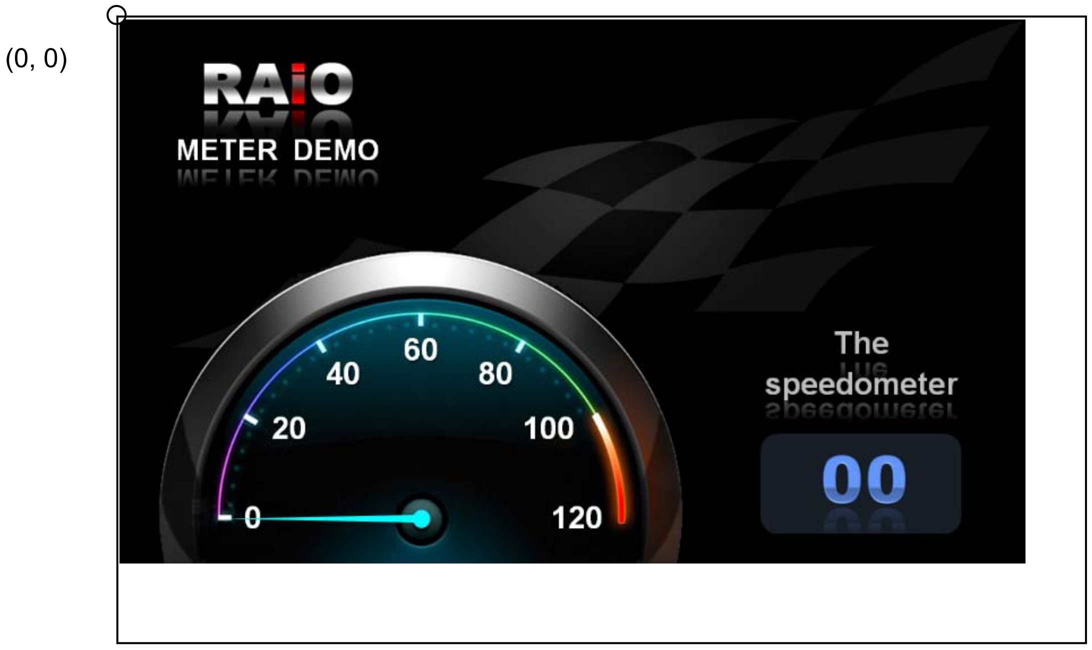


Figure 4-10: Using DMA function get a picture data from ext. Flash and write into SDRAM

Step 2 : Write a 16x16 icon into DDRAM (SDRAM)

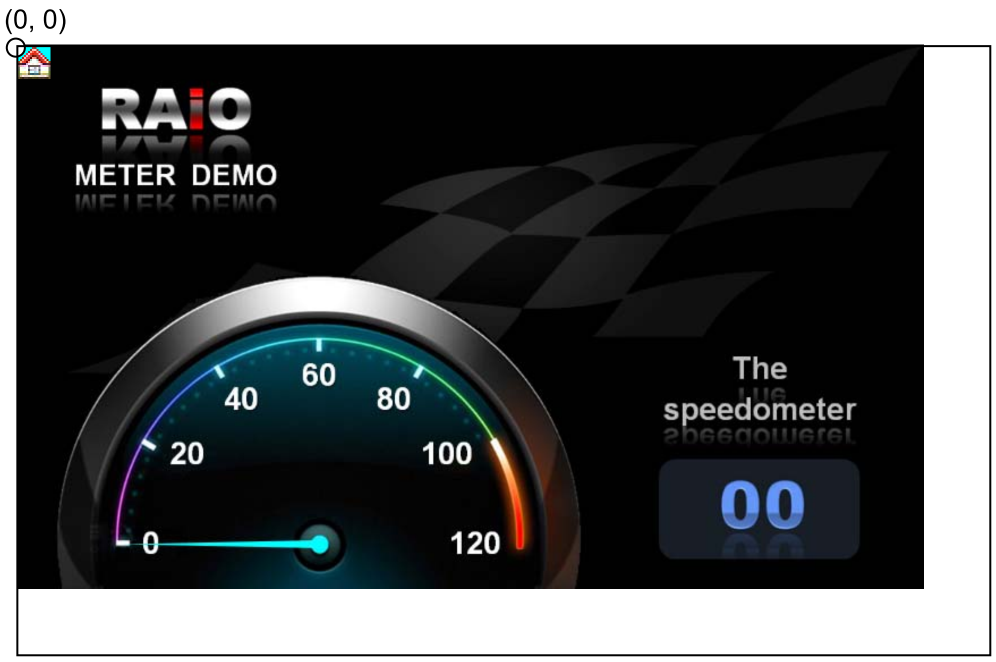


Figure 4-11: Write a 16x16 icon into SDRAM

Step 3 : Execute Pattern fill function

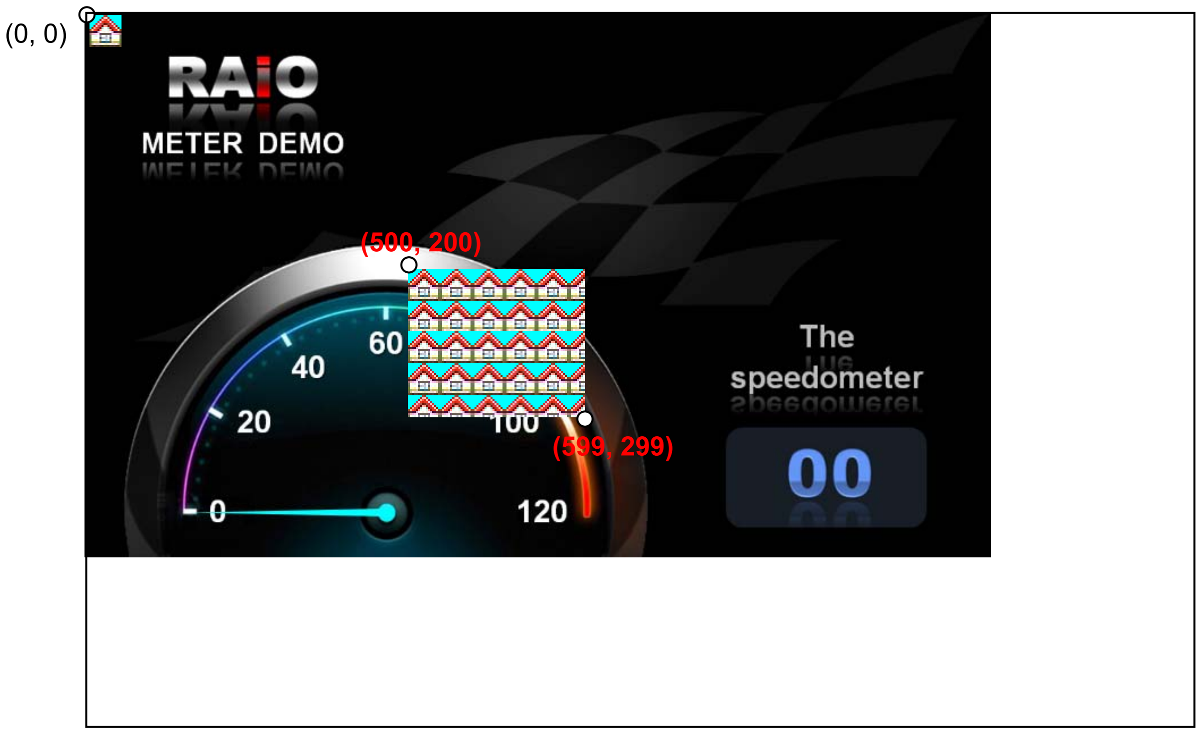


Figure 4-12: Accomplished pattern fill function, Source range is from (0, 0) to (15, 15), and Destination range is from (500,200) to (599,299)

4.2.3:Pattern Fill With Chroma Key

The Pattern Fill with chroma key fills a specified rectangular area of the SDRAM with a pattern. In the pattern fill operation, the Chroma key (transparent color) is ignored. The Chroma key setting in REG[D5h]~[D7h],When Pattern color is equal to Chroma set , then no change data in Destination.

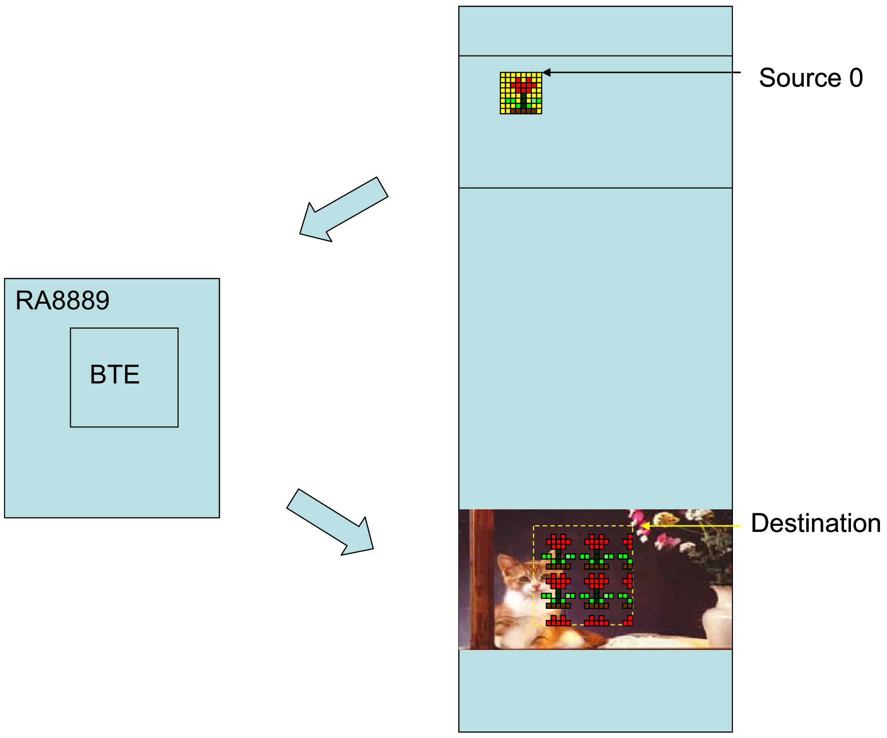


Figure 4-10 : Hardware Flow

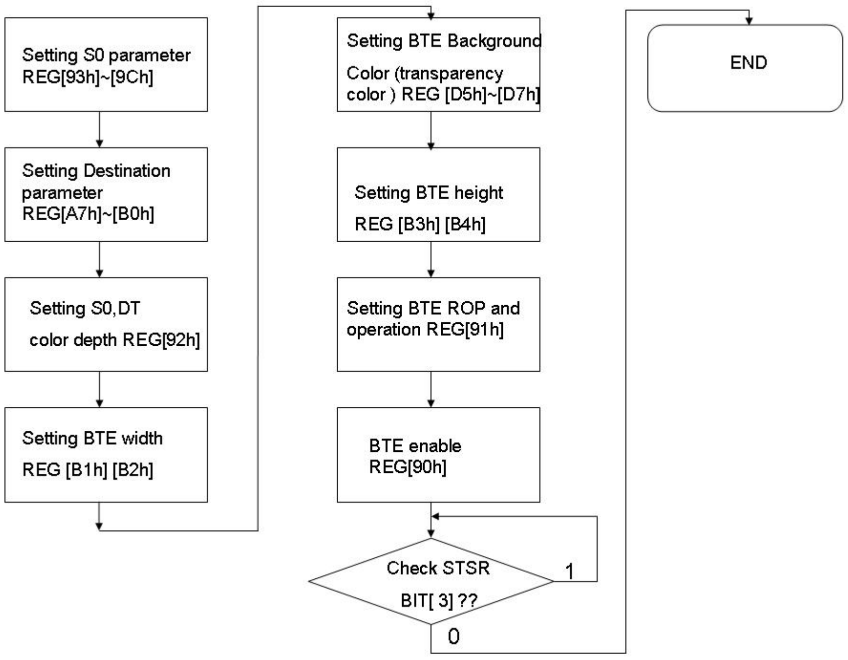

Setting S0 parameter REG[93h]~[9Ch]
Setting Destination parameter REG[A7h]~[B0h]
Setting S0,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE Background Color (transparency color) REG [D5h]~[D7h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Check STSR BIT[3] ??
1
0
END

Figure 4-11 : Flow Chart

4.2.4: Display results on LCD for the BTE Pattern Fill with Chroma key Function:

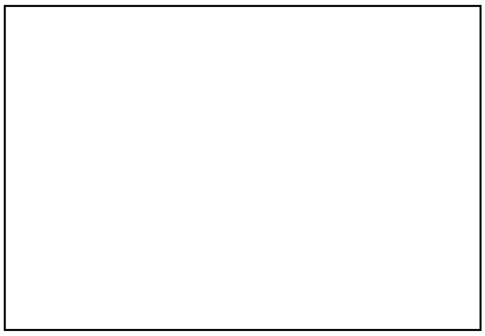


Figure 4-12: SDRAM current data in this example

Figure 4-13 : (16x16) Icon

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
)
```

Example:

```
    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);
```

Condition:

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

Step 1 : Execute DMA function copying a picture into SDRAM from external Serial Flash Memory

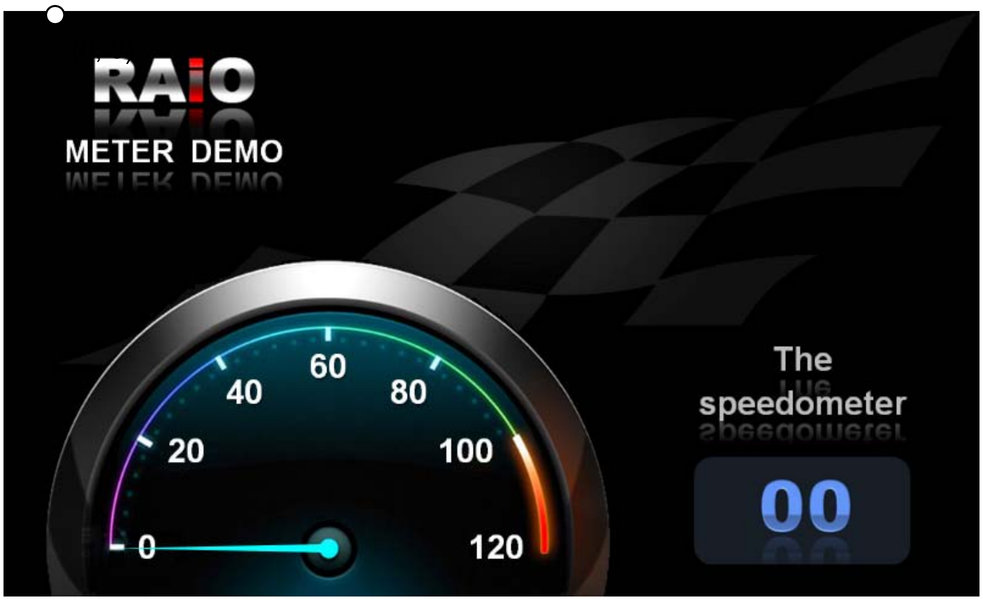


Figure 4-14: Execute DMA function copying a picture into SDRAM from external Serial Flash Memory

Step 2 : Write a 16x16 icon into DDRAM (SDRAM)

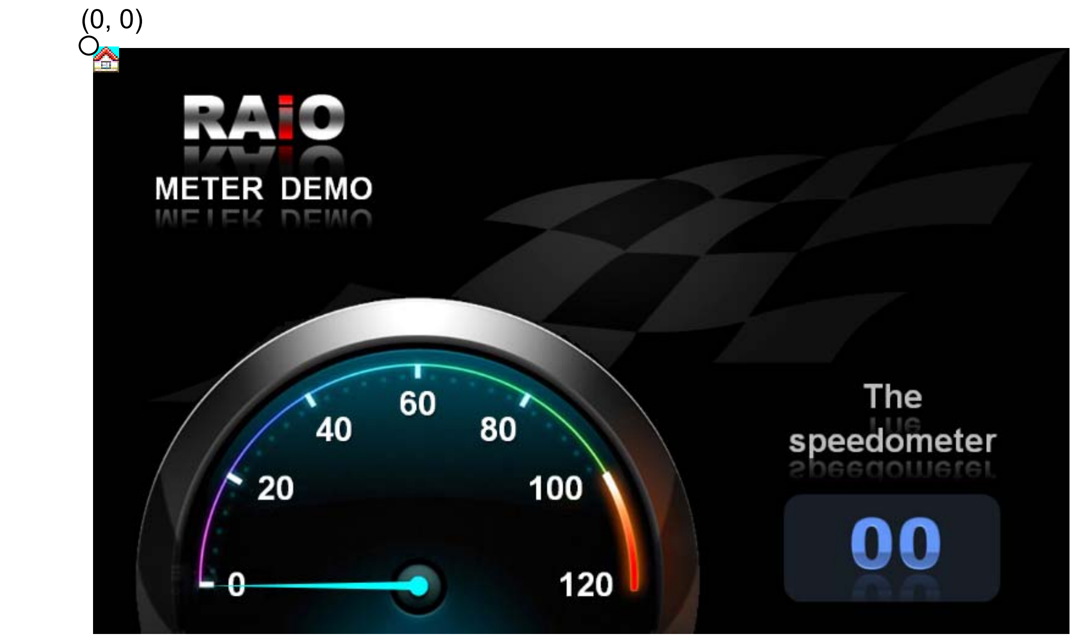


Figure 4-15: Write a 16x16 icon into SDRAM

Step 3 : Execute Pattern fill with chroma key function

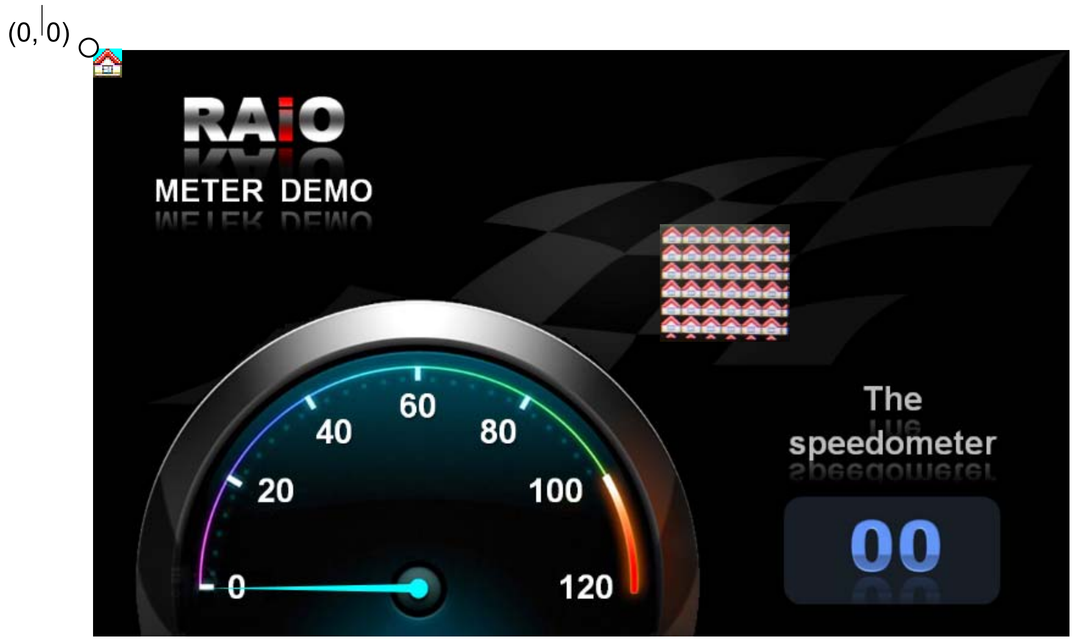


Figure 4-16: Accomplished pattern fill with chroma key function, Source range is from (0, 0) to (15, 15), and Destination range is from (500,200) to (599,299)

4.3.1: MCU Write with ROP

The Write BTE increases the speed of transferring data from MCU interface to the SDRAM. The Write BTE with ROP fills a specified area of the SDRAM with data supplied by the MCU. The Write BTE supports all 16 ROPs. The Write BTE requires the MCU to provide data.

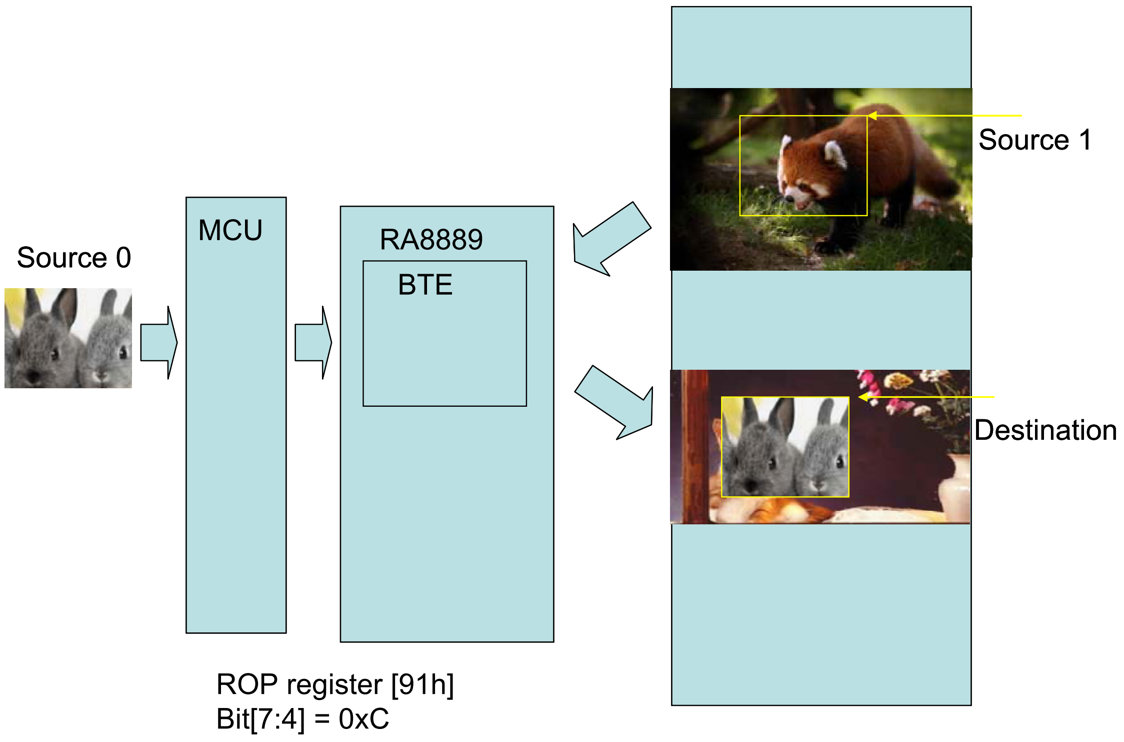


Figure 4-17 : Hardware Data Flow

The suggested programming steps and registers setting are listed as below.

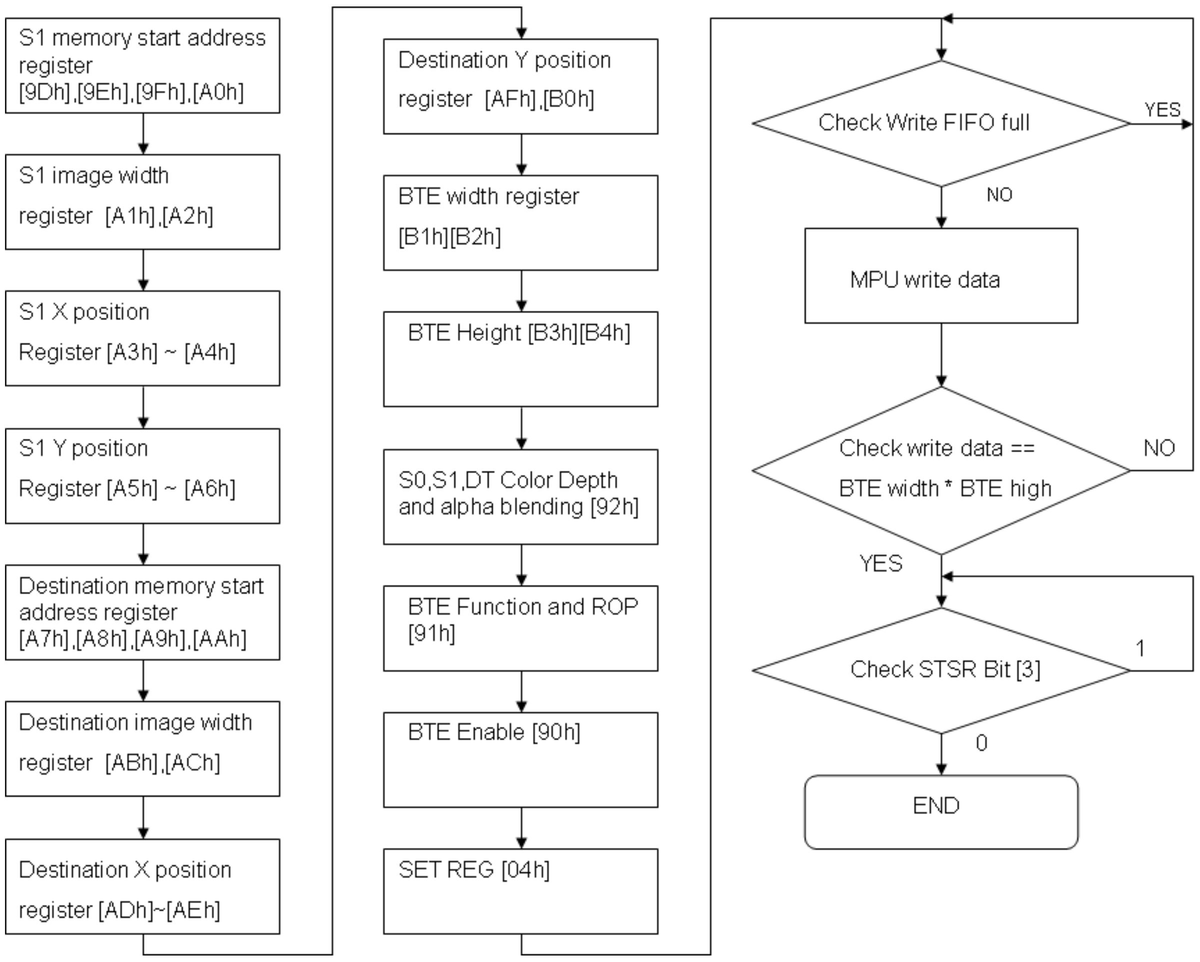

S1 memory start address register [9Dh],[9Eh],[9Fh],[A0h]
S1 image width register [A1h],[A2h]
S1 X position Register [A3h] ~ [A4h]
S1 Y position Register [A5h] ~ [A6h]
Destination memory start address register [A7h],[A8h],[A9h],[AAh]
Destination image width register [ABh],[ACh]
Destination X position register [ADh]~[AEh]
Destination Y position register [AFh],[B0h]
BTE width register [B1h][B2h]
BTE Height [B3h][B4h]
S0,S1,DT Color Depth and alpha blending [92h]
BTE Function and ROP [91h]
BTE Enable [90h]
SET REG [04h]
Check Write FIFO full
YES
NO
MPU write data
Check write data == BTE width * BTE high
NO
YES
Check STSR Bit [3]
1
0
END

Figure 4-18 : Flow Chart

4.3.2: Display results on LCD for the BTE MCU Write with ROP

In BTE MCU Write with ROP function, we provide for each MCU 8bit and 16bit an API for users, Figure 4-19 for SDRAM information, Figure 4-20 for the map data from MCU interface, using BTE MCU Write with ROP function write SDRAM, the map data can be used with ROP parameters do logic operations. The following is API program and Commentary:

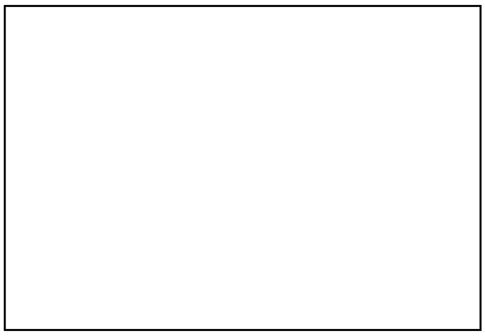


Figure 4-19: SDRAM current data in this example

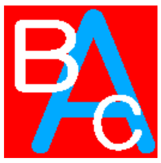


Figure 4-20 : picture data from MCU(128x128)

```
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
)
```

Example:

```
//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 .

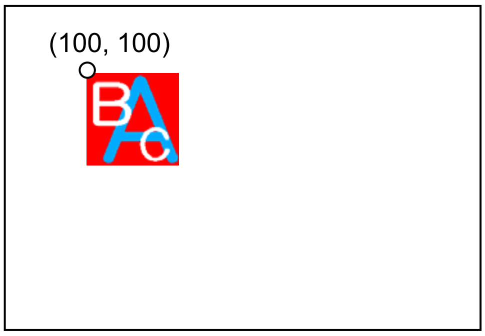


Figure 4-21: BTE MCU Write picture into SDRAM. The destination address is from (100,100) to (227,227)]

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

The MCU Write With chroma key function increases the speed of transferring data from MCU interface to the SDRAM. Once the function begins, the BTE engine remains active until all pixels have been written.

Unlike “Write BTE” operation, the “MCU Write with chroma key” will ignore the operation of a dedicated color of MCU data that is set as “Chroma key Color (Transparent Color)”.If MCU data is equal to dedicated color then BTE write is from S1 data to destination. In RA8889, the “Chroma key color (Transparent Color)” is set as “BTE background Color“. For example, considering a source image has a yellow circle on a red background. By selecting the blue color as the transparent color and using the MCU Write with chroma key on the whole rectangles, the effect is a BTE of the red circle only.

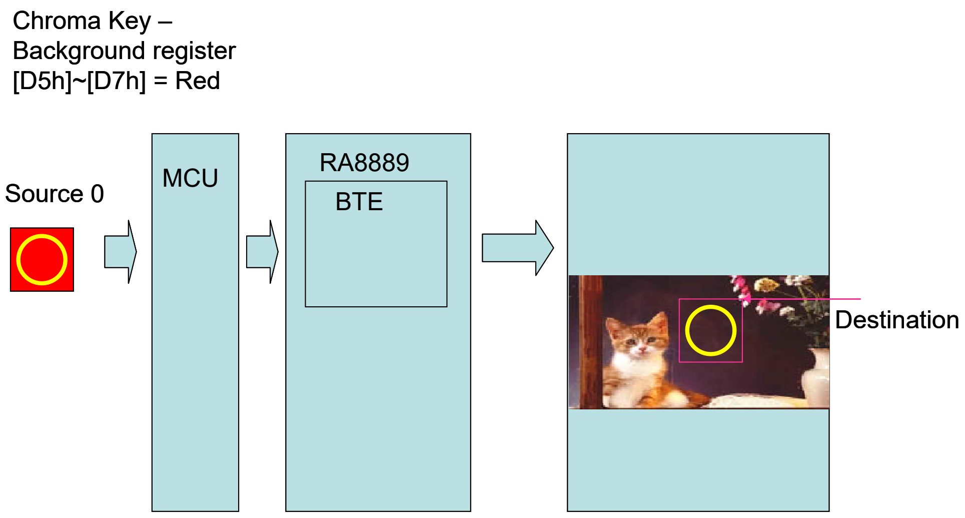


Figure 4-22 : Hardware Data Flow

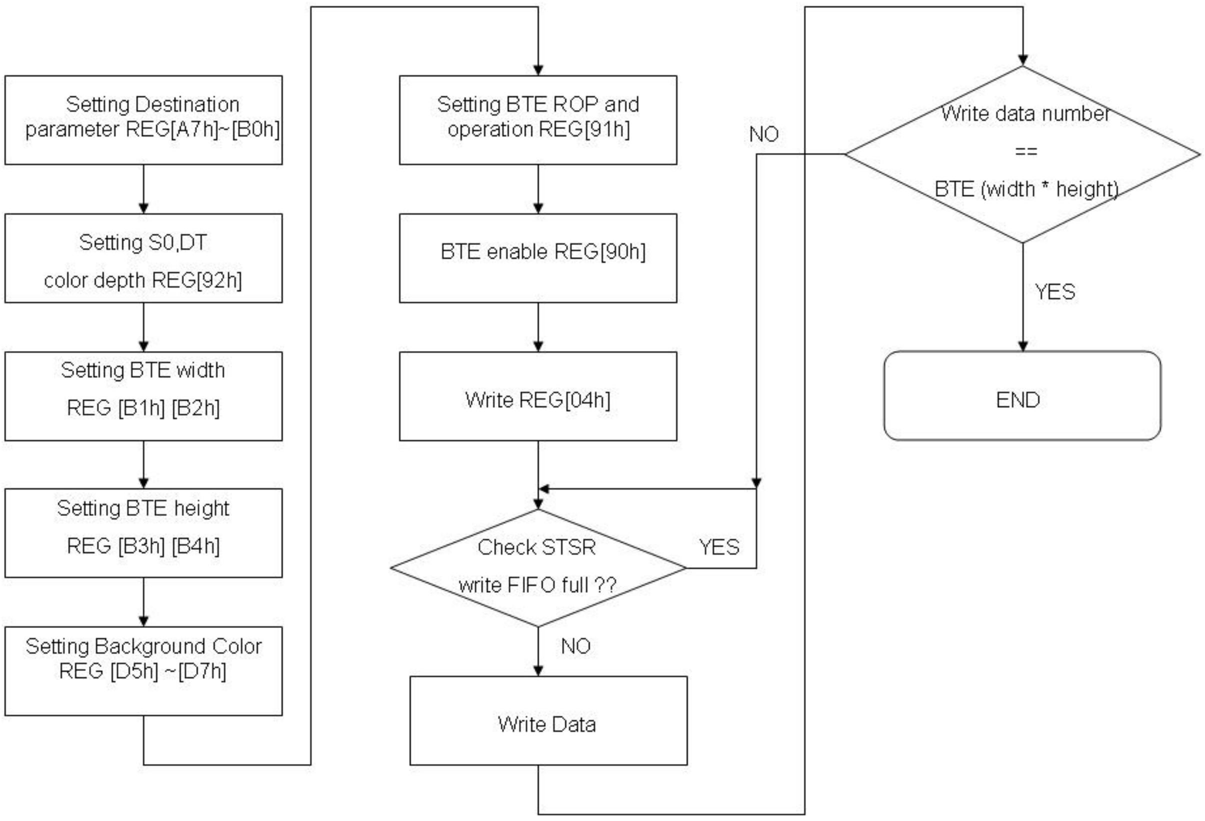
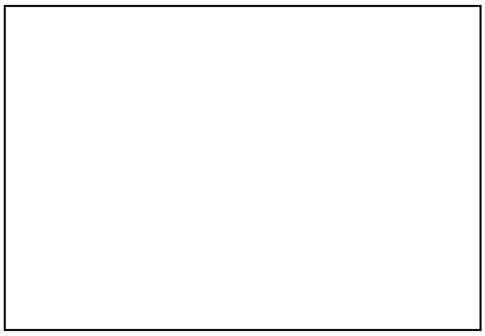

Setting Destination parameter REG[A7h]~[B0h]
Setting S0,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting Background Color REG [D5h] ~[D7h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Write REG[04h]
Check STSR write FIFO full ??
YES
NO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

Figure 4-23 : Flow Chart

4.3.4: Display results on LCD for BTE MCU Write With Chroma Key:

The MCU Write With chroma key function. Increasing the speed of transferring data is from MCU interface to the SDRAM. Once the function begins, the BTE engine remains active until all pixels have been written. Unlike “Write BTE” operation, the “MCU Write with chroma key” will ignore the operation of a dedicated color of MCU data that is set as “Chroma key Color (Transparent Color)”.If MCU data is equal to dedicated color then BTE write is from S1 data to destination. In RA8889, the “Chroma key color (Transparent Color)” is set as “BTE background Color“. For example, considering a source image has a blue color letter “A” and two color white letters “B”, “C” on a red background. By selecting the red color as the transparency color and using the MCU Write with chroma key on the whole rectangles, the effect is a BTE of the letters only.



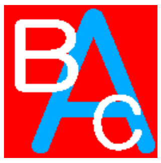


Figure 4-25 picture data from MCU(128x128)

Figure 4-24: SDRAM current data in this example

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
)
```

Example:

```
//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,pic16
241);
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,pic16
24);
```

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.

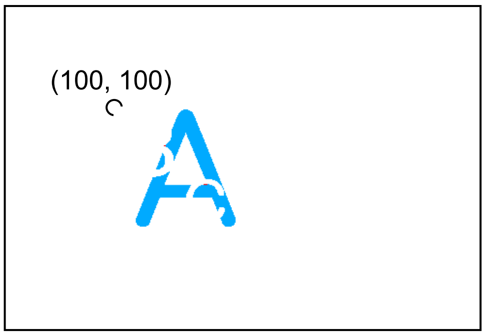


Figure 4-26: BTE MCU Write With Choma Key , Transparency color = red color.
The destination address is from (100,100) to (227,227).

4.4.1: Memory Copy with ROP

The Memory Copy moves a specific area of the SDRAM to a different area of the SDRAM. This operation can speed up the data copy operation from one block to another and save a lot of MCU processing time and loading.

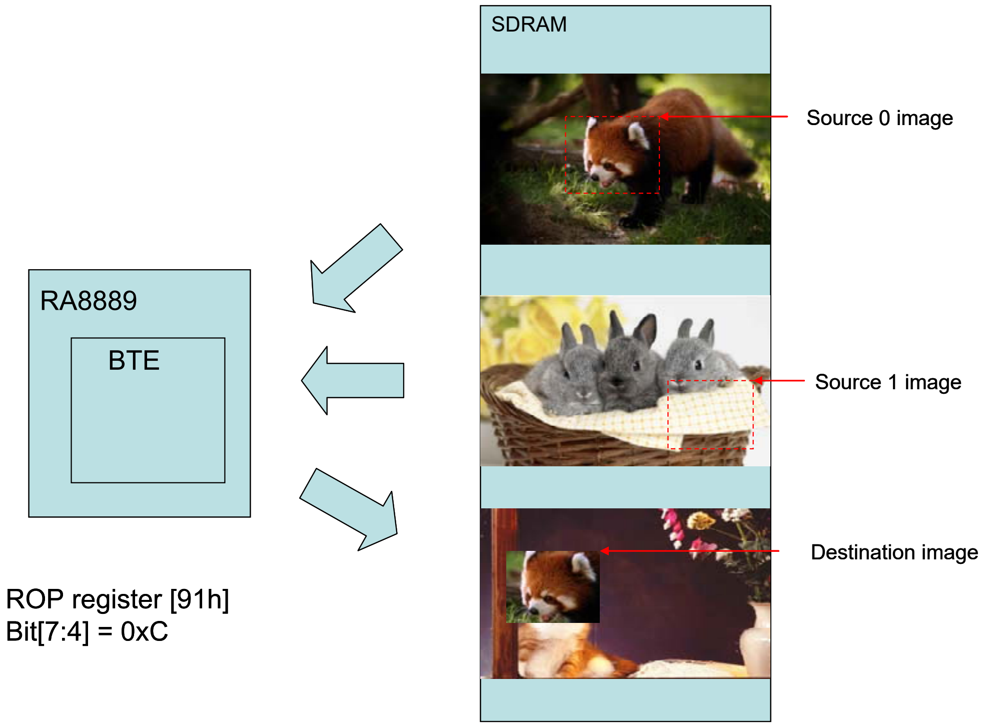


Figure 4-27 : Hardware Data Flow

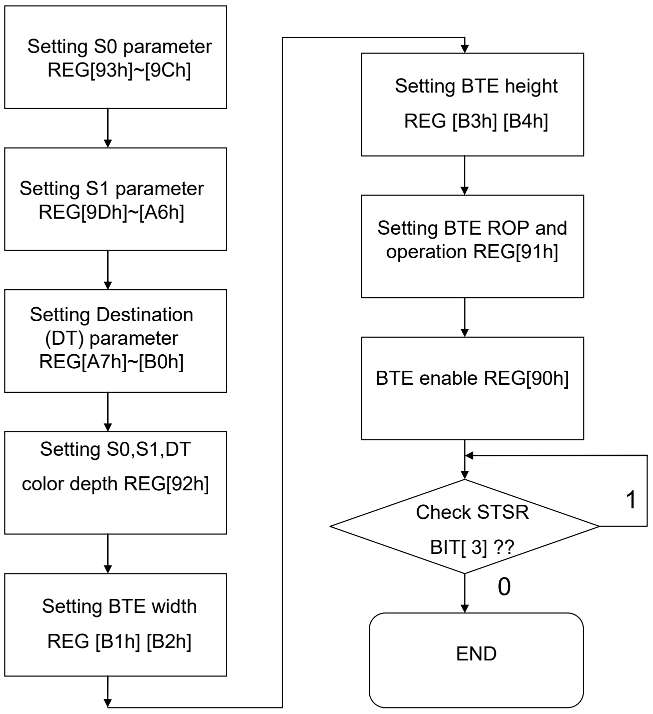

Setting S0 parameter REG[93h]~[9Ch]
Setting S1 parameter REG[9Dh]~[A6h]
Setting Destination (DT) parameter REG[A7h]~[B0h]
Setting S0,S1,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Check STSR BIT[3] ??
1
0
END

Figure 4-28 : Flow Chart

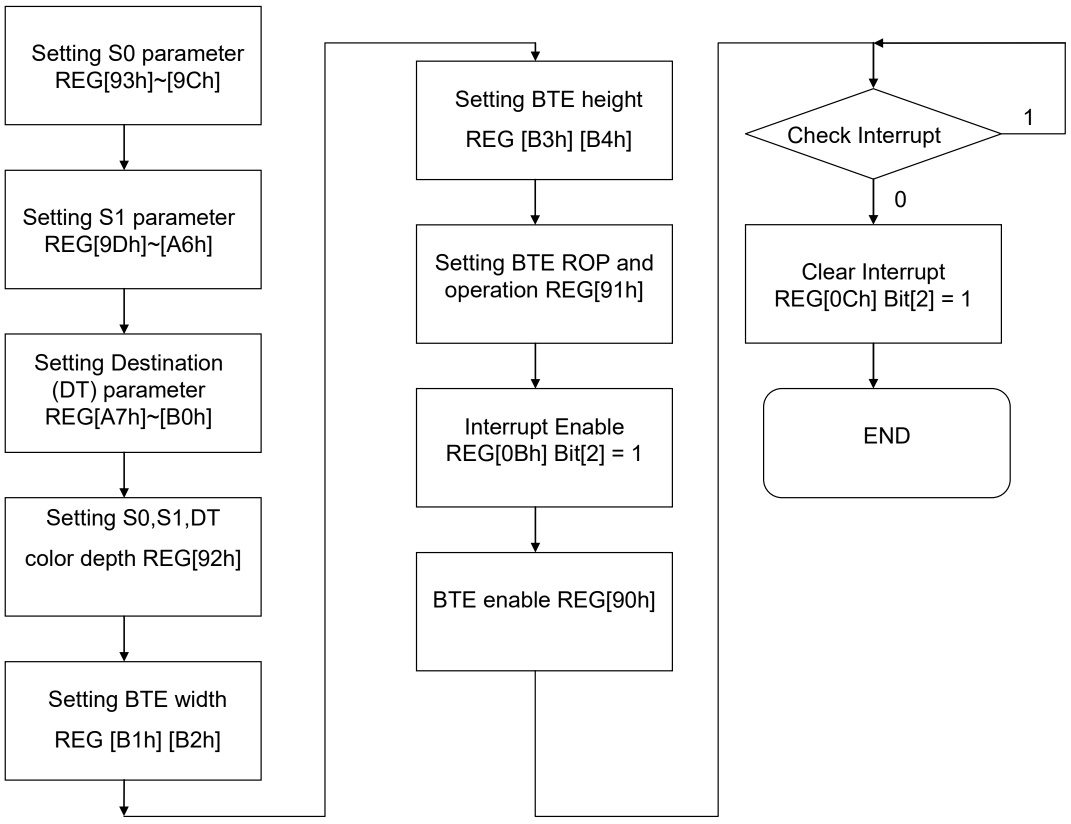

Setting S0 parameter REG[93h]~[9Ch]
Setting S1 parameter REG[9Dh]~[A6h]
Setting Destination (DT) parameter REG[A7h]~[B0h]
Setting S0,S1,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
Interrupt Enable REG[0Bh] Bit[2] = 1
BTE enable REG[90h]
Check Interrupt
1
0
Clear Interrupt REG[0Ch] Bit[2] = 1
END

Figure 4-29 : Flow Chart – Check Int

4.4.2: Display results on LCD for BTE Memory Copy with ROP:

The following is the explanation and examples BTE Memory Copy with ROP API function, first use BTE Solid Fill function to draw a square filled with red and circle filled with yellow by draw function(Figure 4-30) , then through BTE Memory Copy to copy a same pattern (Figure 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
)
```

Example:

```
/*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);
```

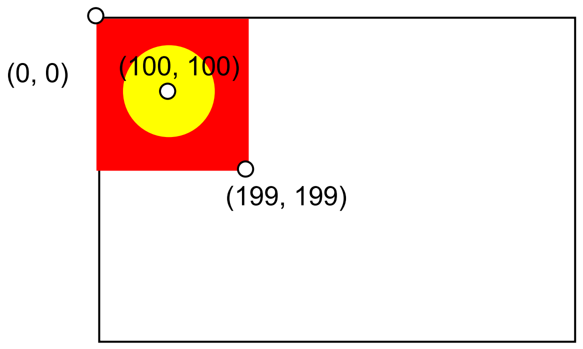


Figure 4-30: draw a square filled with red and circle filled with yellow

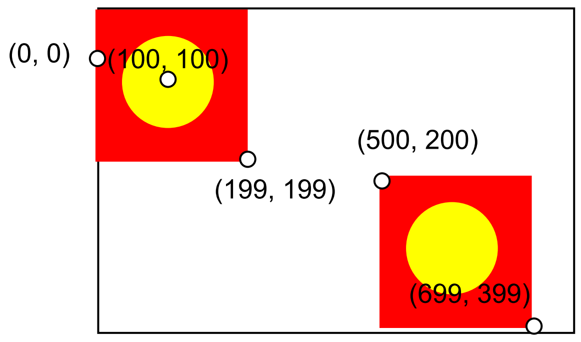


Figure 4-31 : copy a same pattern by BTE Memory Copy with ROP

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

"Memory copy with chroma key" moves a specified area of the SDRAM to the different specified area of the same SDRAM with ignoring the "choma key (Transparent Color)". The chroma key (transparency color) setting in BTE background color register. If chroma key (transparency color) meets, then not change destination data. The source 0, source 1 / destination data are in the memory.

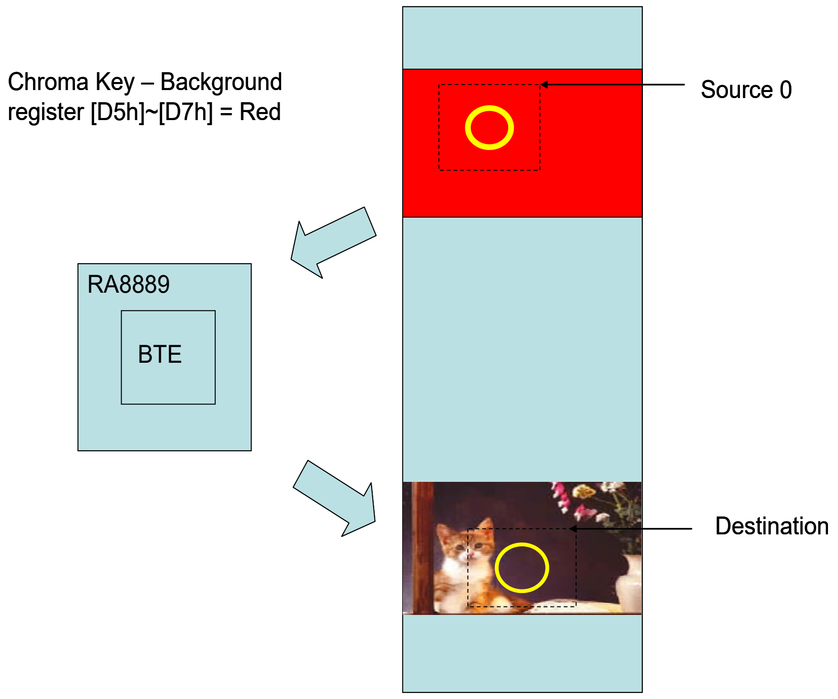


Figure 4-32 : Hardware Data Flow

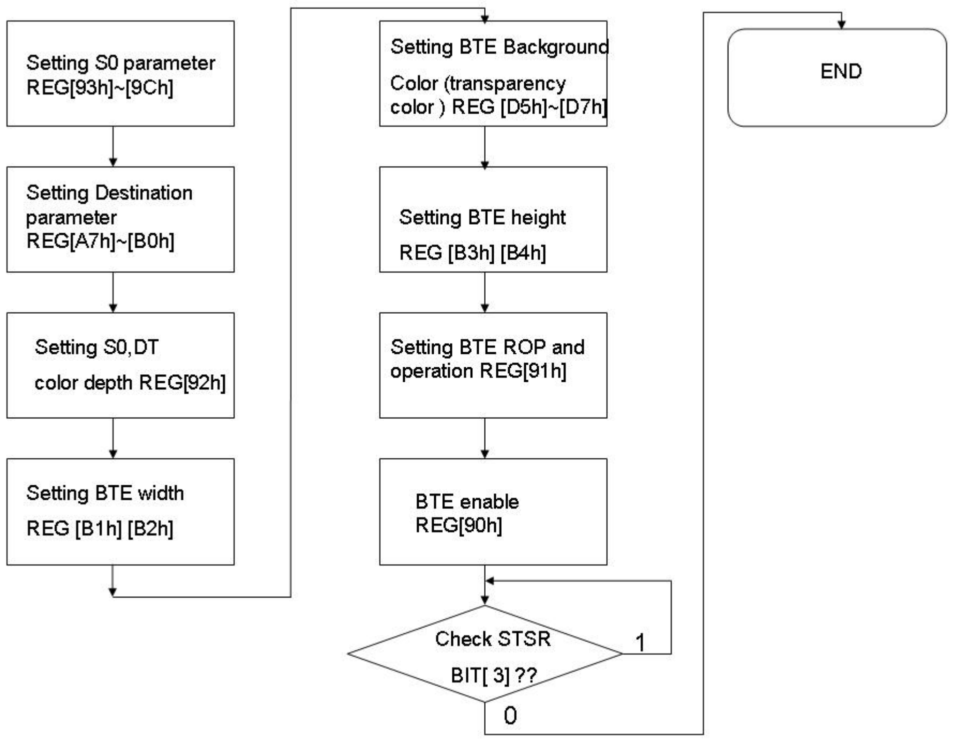

Setting S0 parameter REG[93h]~[9Ch]
Setting Destination parameter REG[A7h]~[B0h]
Setting S0,DT color depth REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE Background Color (transparency color) REG [D5h]~[D7h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
BTE enable REG[90h]
Check STSR BIT[3] ??
1
0
END

Figure 4-33 : Flow Chart

4.4.4 : Display results on LCD for BTE Memory Copy with Chroma key:

The following is the commentary and example for BTE Memory Copy with Chroma key (w/o ROP) API function, first use BTE Solid Fill function to draw a square filled with red and circle filled with yellow by draw function (Figure 4-34), then through BTE Memory Copy with Chroma key (w/o ROP) to copy a portion of the red filtered pattern (Figure 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
)
```

Example:

```
/*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);

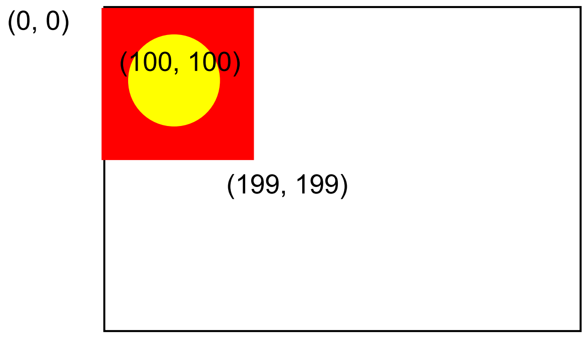


Figure 4-34: draw a square filled with red and circle filled with yellow

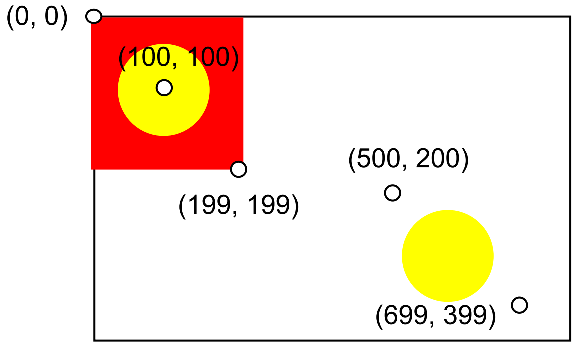


Figure 4-35 : example for BTE Memory Copy with Chroma key

4.5.1:MCU Write With Color Expansion

"MCU Write With Color Expansion" is a useful operation to translate monochrome data of MCU interface to be colorful one. In the operation, the source data will be treated as a monochrome bit-map. The bit-wise data is translated to multi-bits per pixel color data by the setting of "BTE Foreground Color" and "BTE Background Color". If the data on the source is equal to logical "1", then it will be translated to "BTE Foreground Color". If the data on the source is equal to logical "0", then it will be translated to "BTE Background Color". This function can largely reduce the effort of system translation from mono system to color system. When the end of the line is reached, any unused bits will be discarded. The data for the next line will be taken from the next data package. Each bit is serially expanded to the destination data starting from MSB to LSB. If MCU interface set 16bit , then ROP (start bit) - 15:0 valid., If MCU interface set 8bit, then ROP (start bit) – 7:0 valid.. Source 0 color depth REG [92h] Bit[7:6] don't care.

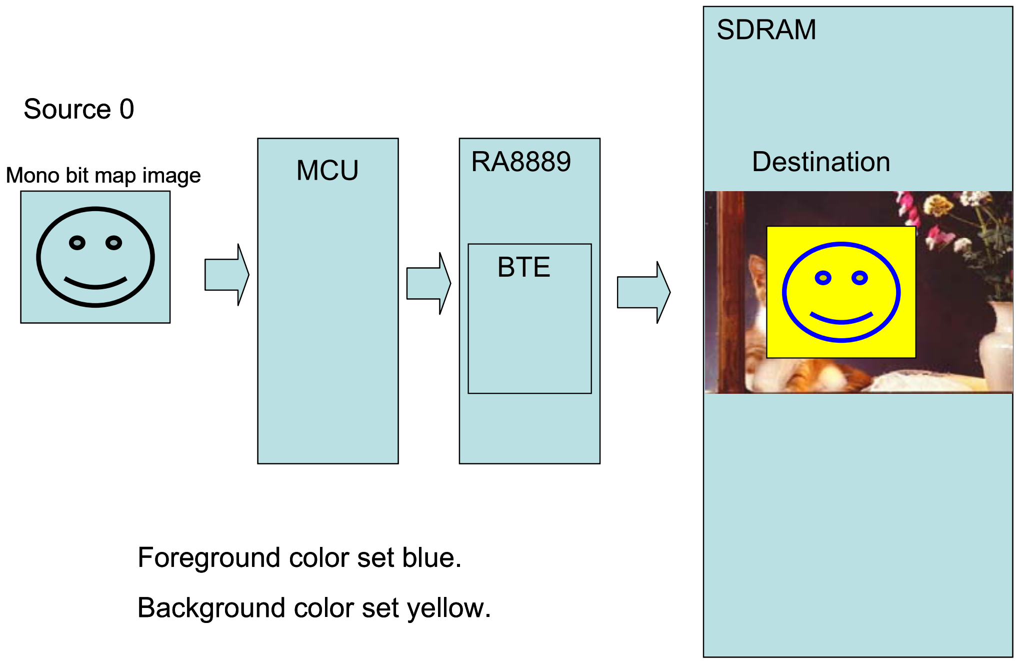


Figure 4-36 : Hardware Data Flow

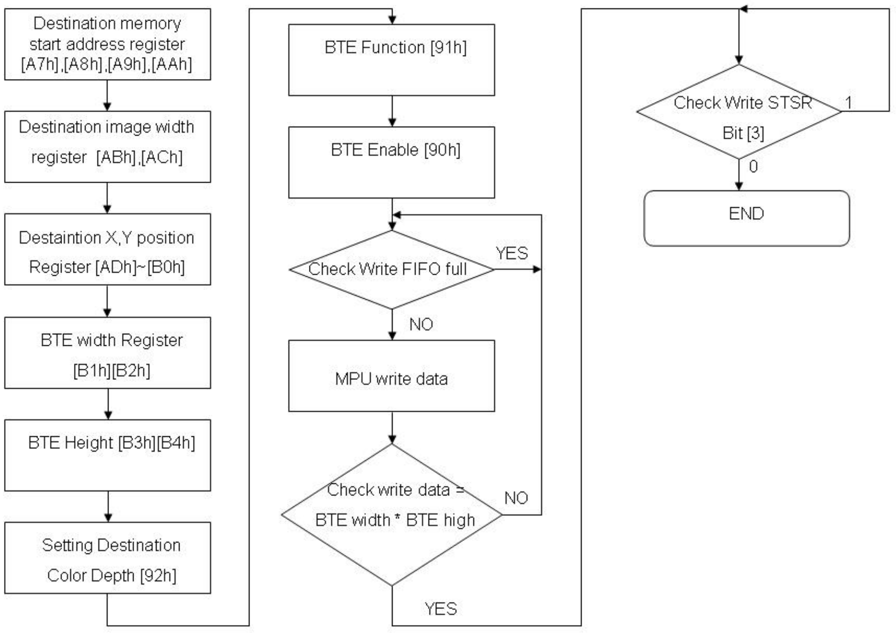

Destination memory start address register [A7h],[A8h],[A9h],[AAh]
Destination image width register [ABh],[ACh]
Destaintion X,Y position Register [ADh]~[B0h]
BTE width Register [B1h][B2h]
BTE Height [B3h][B4h]
Setting Destination Color Depth [92h]
BTE Function [91h]
BTE Enable [90h]
Check Write FIFO full
YES
NO
MPU write data
Check write data = BTE width * BTE high
NO
YES
Check Write STSR Bit [3]
1
0
END

Figure 4-37 : Flow Chart

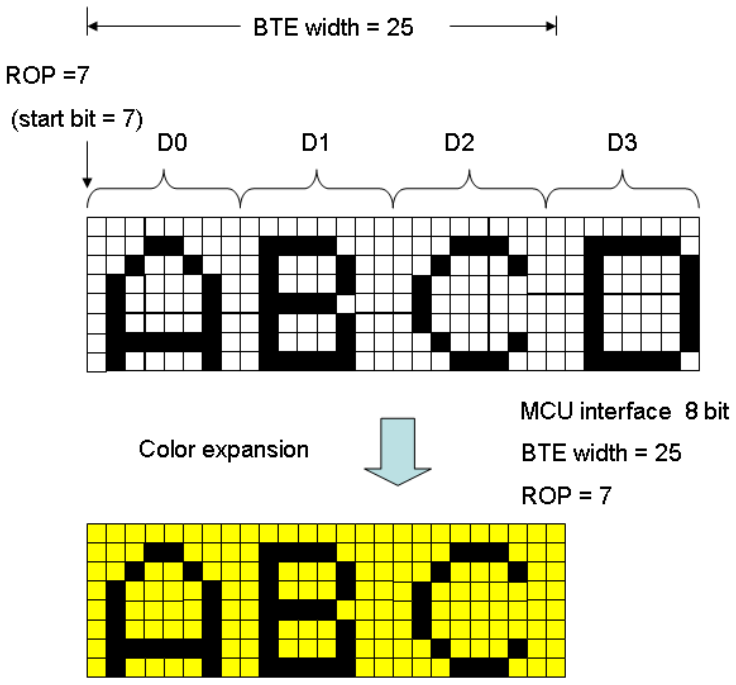


Figure 4-38 :Start Bit Example 1

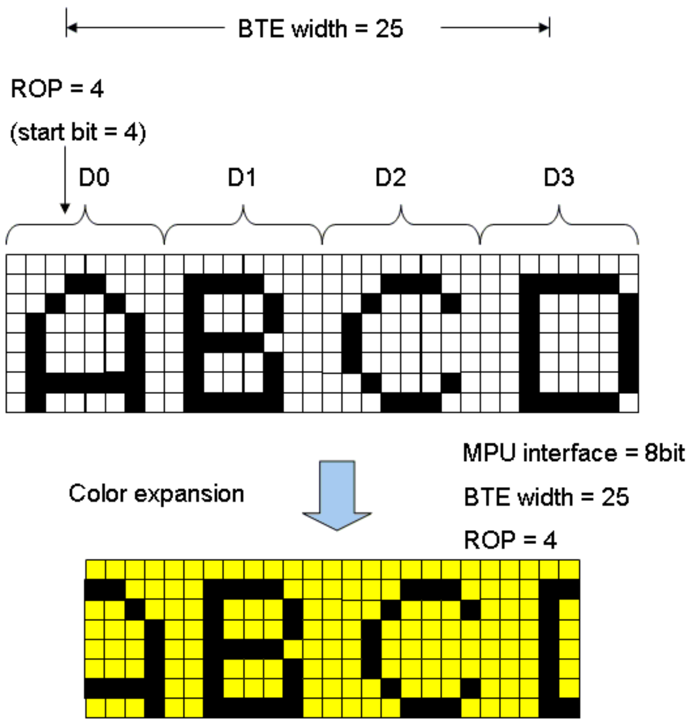


Figure 4-39 : Start bit Exapmle 2

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

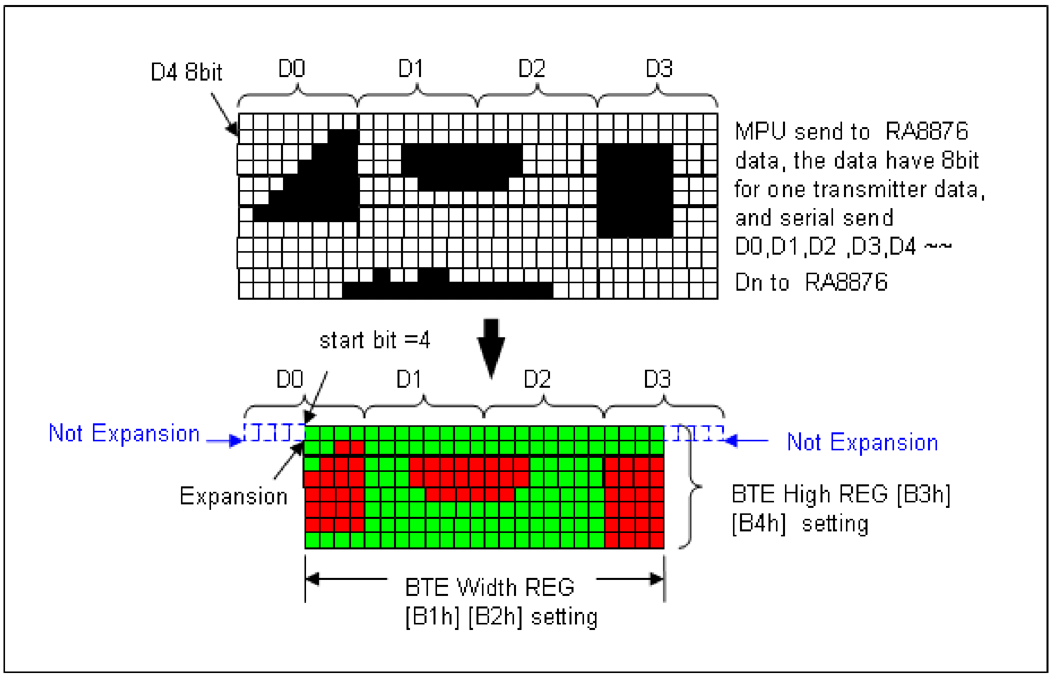

D4 8bit
D0
D1
D2
D3
MPU send to RA8876 data, the data have 8bit for one transmitter data, and serial send D0,D1,D2 ,D3,D4 ~~ Dn to RA8876
start bit =4
D0
D1
D2
D3
Not Expansion
Not Expansion
Expansion
BTE High REG [B3h] [B4h] setting
BTE Width REG [B1h] [B2h] setting

Figure 4-40 : Color Expansion Data Diagram

4.5.2:Display results on LCD for illustrating the BTE MCU Write with Color Expansion:

Figure 4-41 is a 128x128 monochrome image, assuming that the foreground color is set to green, the background color is set to blue, use the BTE MCU Write with Color Expansion function, it will turn out like the pattern in Figure 4-42 the same. Below we provide a set of API and each illustrated with examples for reference for the MCU 8bit and 16bit.

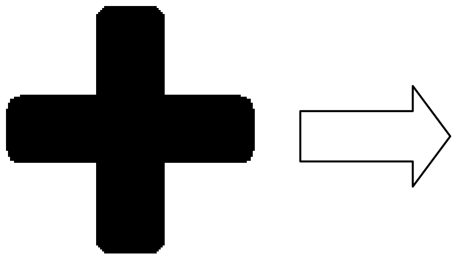


Figure 4-41 :
Original pattern for monochrome 128x128 picture (1-bit bpp)

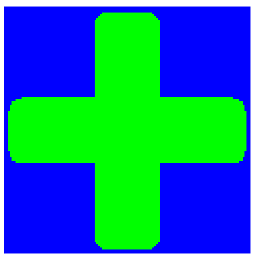


Figure 4-42 :
BTE MCU Write with color expansion

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
)
```

Example :

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

//MCU_8bit_ColorDepth_8bpp                          //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_8bit(0,canvas_image_width,0,0,128,128,0x03,0x1c,gImage_1);
or
//MCU_8bit_ColorDepth_16bpp                         //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_MCU_8bit(0,canvas_image_width,0,0,128,128,0x001f,0x07e0,gImage
_1);
or
//MCU_8bit_ColorDepth_24bpp                         //setting in UserDef.h
```

```
BTE_MCU_Write_ColorExpansion_MCU_8bit(0,canvas_image_width,0,0,128,128,0x0000ff,0x00ff00,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);
```

Virtual display on LCD:

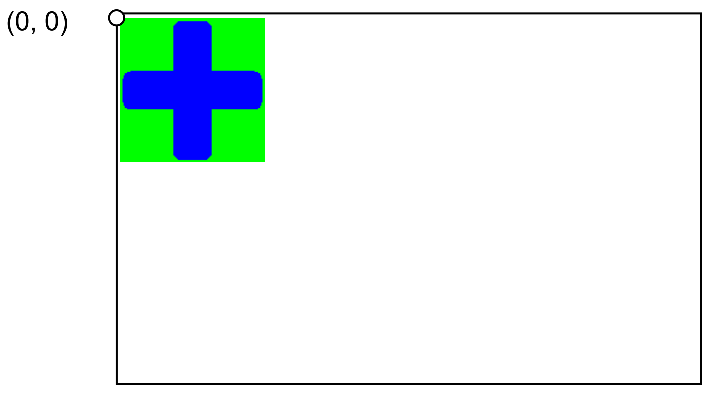


Figure 4-43 : BTE MCU Write with Color Expansion

4.5.3: MCU Write with Color Expansion and Chroma key

This BTE operation is virtually identical to the Color Expand BTE, except the background color is completely ignored. All bits set to 1 in the source monochrome bitmap are color expanded to the "BTE Foreground Color". All bits set to 0 in source monochrome bitmap that would be expanded to the "BTE Background Color" are not expanded at all.

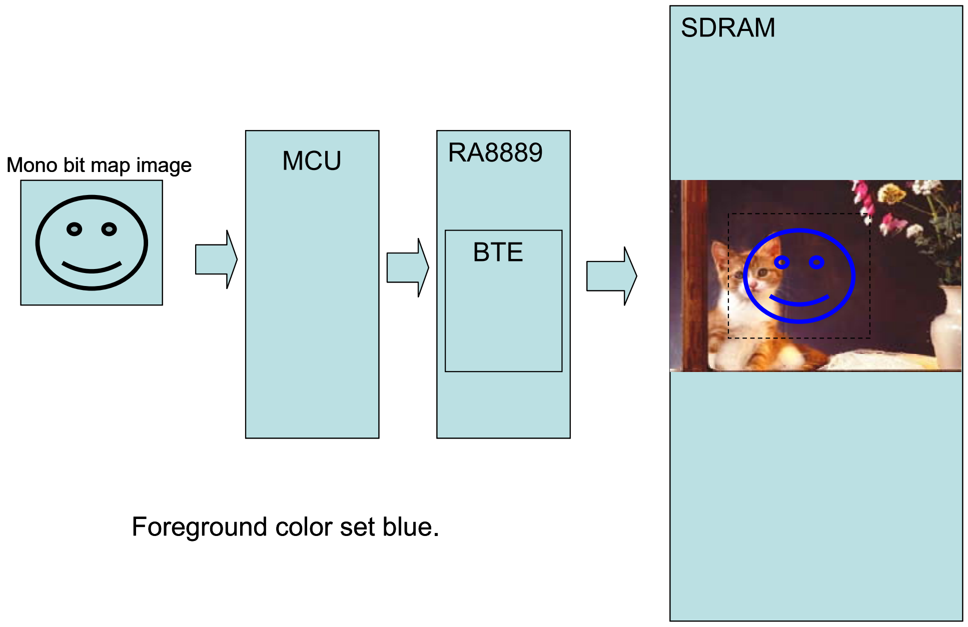


Figure 4-44 : Hardware Data Flow

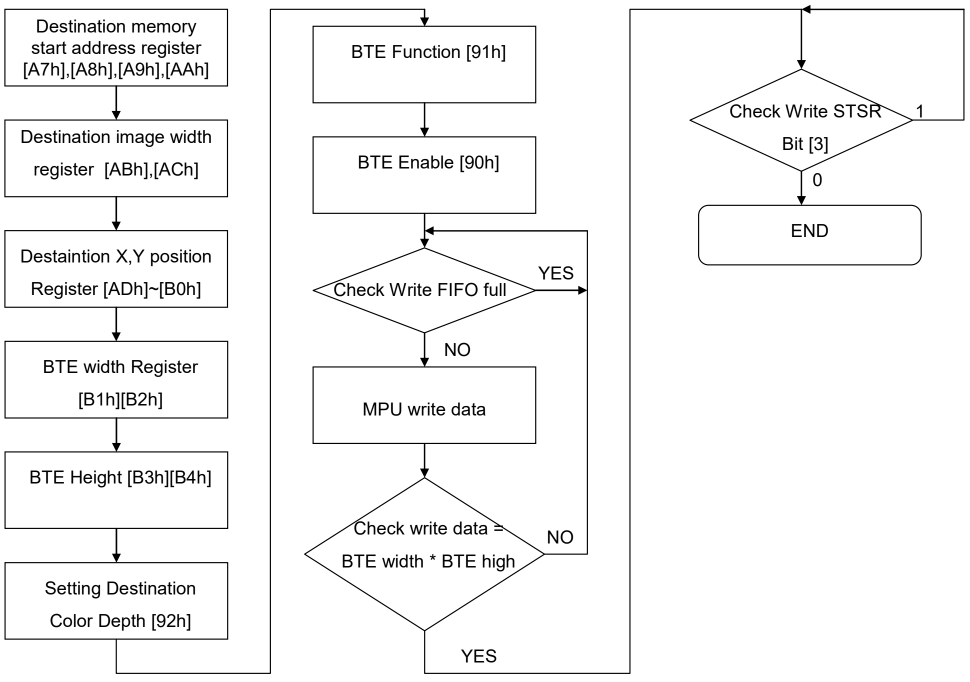

Destination memory start address register [A7h],[A8h],[A9h],[AAh]
Destination image width register [ABh],[ACh]
Destaintion X,Y position Register [ADh]~[B0h]
BTE width Register [B1h][B2h]
BTE Height [B3h][B4h]
Setting Destination Color Depth [92h]
BTE Function [91h]
BTE Enable [90h]
Check Write FIFO full
YES
NO
MPU write data
Check write data = BTE width * BTE high
NO
YES
Check Write STSR Bit [3]
1
0
END

Figure 4-45 : Flow Chart

4.5.4:Display results on LCD for illustrating BTE MCU Write With Color Expansion and Chroma key API:

Figure 4-46 is a 128x128 monochrome image, assuming that the foreground color is set to green, use the BTE MCU Write with Color Expansion and Chroma key function, it will turn out like the pattern in Figure 4-47 the same. Below we provide a set of API and each illustrated with examples for reference for the MCU 8bit and 16bit.

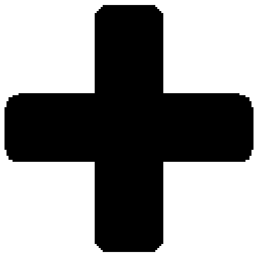
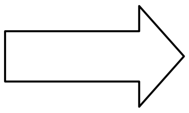


Figure 4-46 :

Original pattern for monochrome 128x128 picture(1-bit bpp)



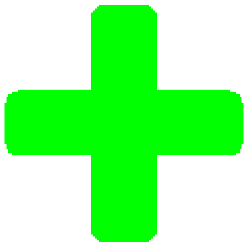


Figure 4-47 :

BTE MCU Write 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
)
```

Example:

```
/*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,gIm
age_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,gI
mage_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,
gImage_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,0,0,128,128,0xff000
0,Test);
or
```

```
//MCU_16bit_ColorDepth_24bpp_Mode_2                    //setting in UserDef.h
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit(0,canvas_image_width,0,0,128,128,0xff000
0,Test);
```

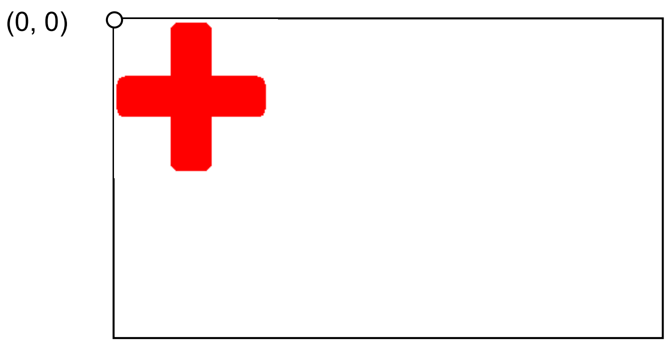


Figure 4-48 : BTE MCU Write With Color Expansion and Chroma key

4.6 : BTE Alpha Blending

Overview:

In so many display applications, alpha blending is the process of combining an image with the other layer display data in order to show translucency on the display. RA8889 also provide the Alpha Blending function through hardware design, user can get more fancy sense of sight on the display but it does not need to occupy so much system resources. This application note aims to help our users take advantage of RA8889.

4.6.1: Memory Copy with Alpha Blending

The "Memory Copy with opacity" function can mix two different images (e.g. source 0 image and source 1 image) into a new image via alpha value and paste the result on a destination image. This function provides two modes for use, e.g. Picture mode and Pixel mode. Picture mode can be operated in 8 bpp/16bpp/24bpp image source and there is only one opacity value (alpha Level) for whole picture. The opacity value is specified on REG[B5h]. Pixel mode is operated in 8bpp/16bpp/32bpp image and each pixel has its own opacity value. In pixel mode, bit [15:12] represents its alpha value for 16bpp source 1 image and the other bits are color data; for 8bpp image format, bit [7:6] reperents alpha level and the other bit (bit [5:0]) is use to map palette color data into real image data. According to 32bpp image in pixel mode, S1 color depth must be set to 16bpp, and S1 width must be set to the same width of the original image (width). In 32-bit pixel mode, bit[31:24] of S1 image reperents its alpha value and bit[23:0] represents pixel data. Figure 13-32 shows the flow about how to write RGB image data into SDRAM via MPU interface.

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

Pixel mode 32bpp - Destination data = (Source 0 * (1 - alpha Level)) + (Source 1 [24:0] * alpha Level)

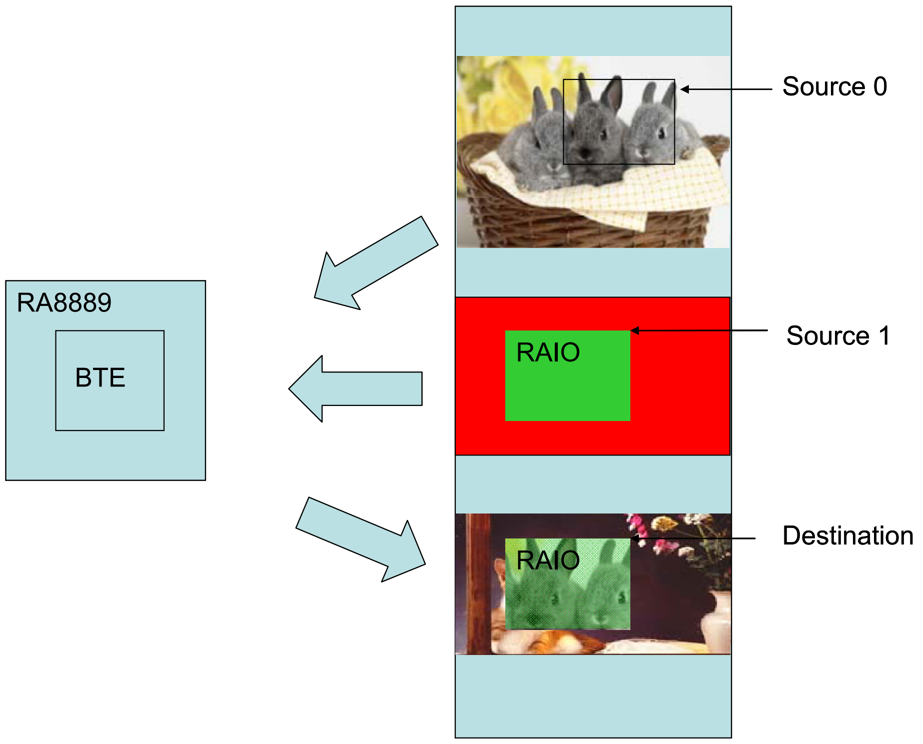


Figure 4-49 : Picture Mode Hardware Data Flow

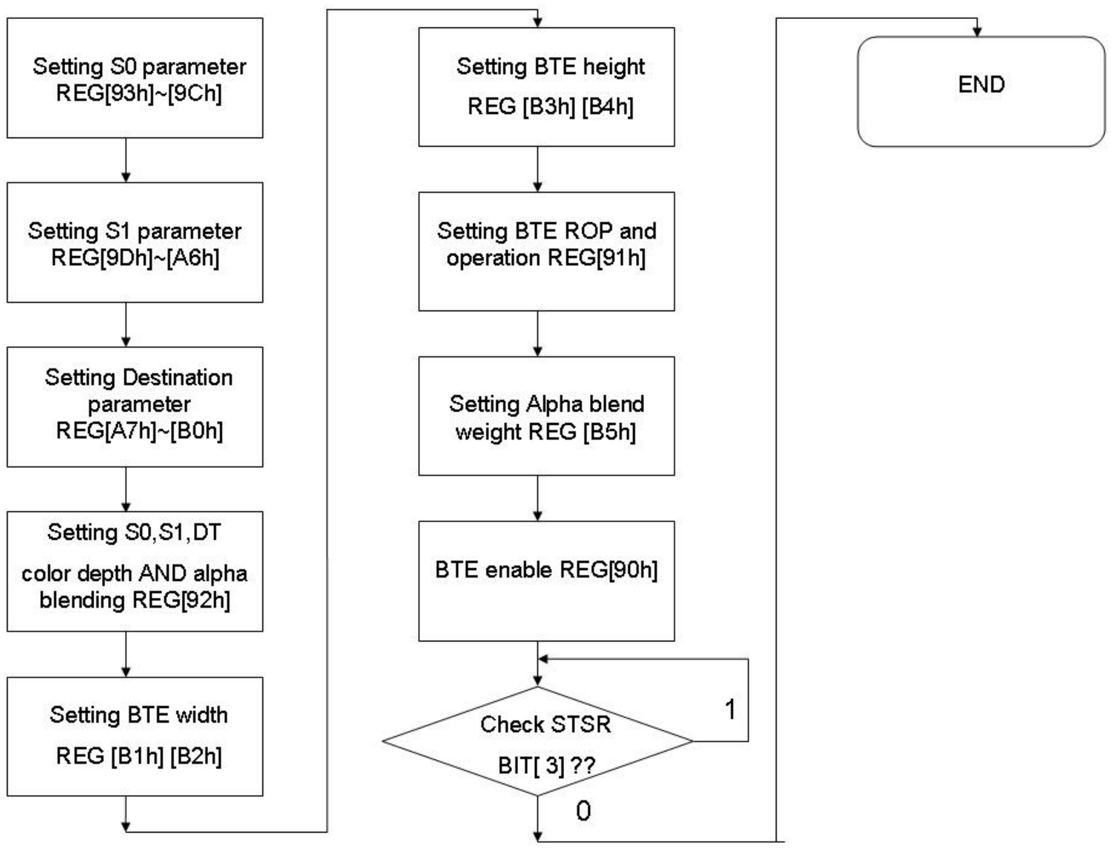

Setting S0 parameter REG[93h]~[9Ch]
Setting S1 parameter REG[9Dh]~[A6h]
Setting Destination parameter REG[A7h]~[B0h]
Setting S0,S1,DT color depth AND alpha blending REG[92h]
Setting BTE width REG [B1h] [B2h]
Setting BTE height REG [B3h] [B4h]
Setting BTE ROP and operation REG[91h]
Setting Alpha blend weight REG [B5h]
BTE enable REG[90h]
Check STSR BIT[3] ??
1
0
END

Figure 4-50 : Picture Mode Flow Chart

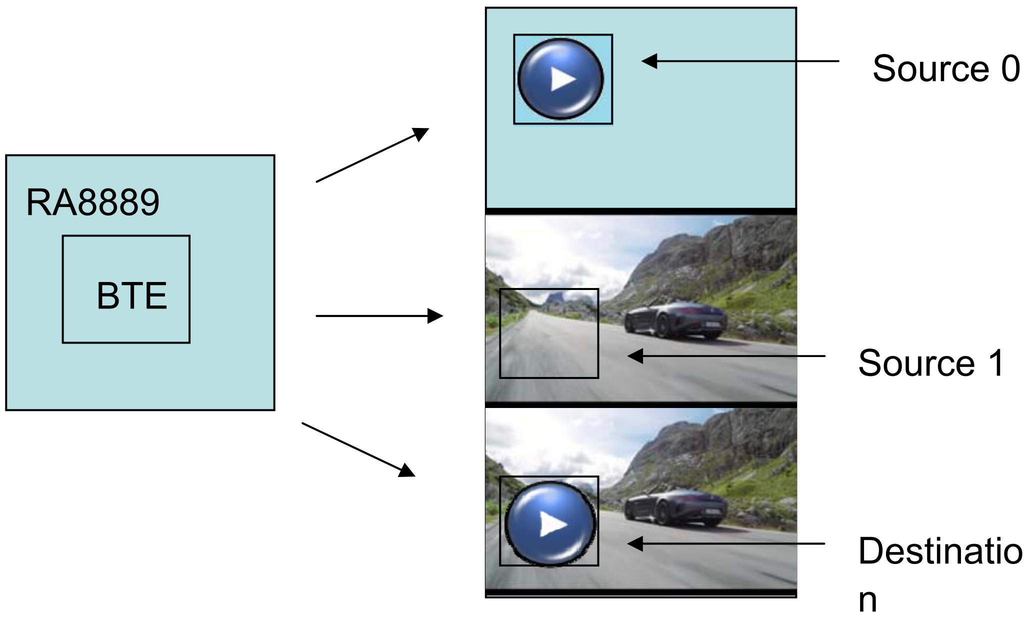


Figure 4-50 : Pixel Mode Hardware Data Flow

4.6.2: Display results on LCD for illustrating BTE Memory Copy with Alpha Blending in Picture mode

The “Memory Copy with Alpha blending” function can mix source 0 data and source 1 blending to Destination. That is having 2 mode - Picture mode and Pixel mode. In this case, we use picture mode.

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

Example:

```
/*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);
```

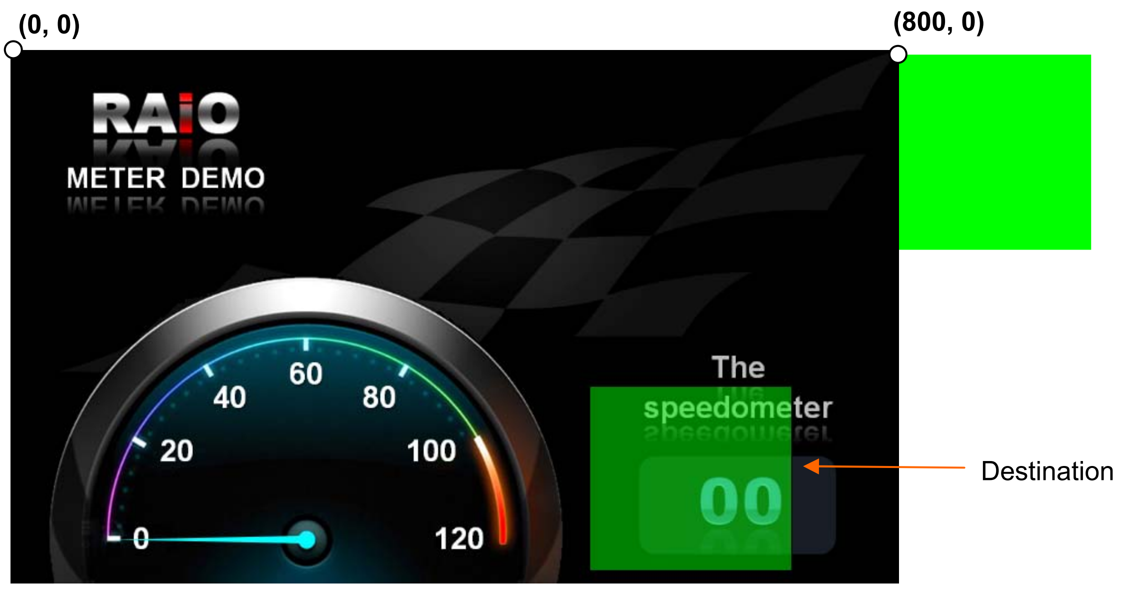


Figure 4-51 : Source0[(500,300) ~ (699,499)] ,Source1[(800,0) ~ (999,199)] executes alpha blending and showing on Destination[(500,300) ~ (699,499)], alpha = 16.

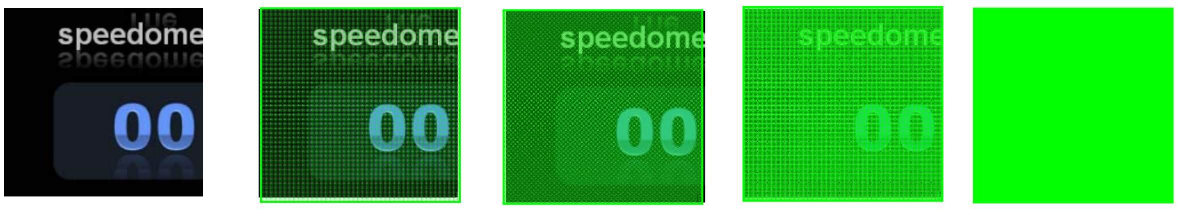


Figure 4-52: Destination in different alpha/transparent level.

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

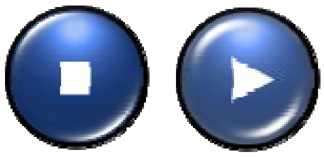
4.6.2: Display results on LCD for illustrating BTE Memory Copy with Alpha Blending in Pixel mode

```
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
)
```

Example:

Step 1:Write PNG(32bit ARGB) pic to SDRAM

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

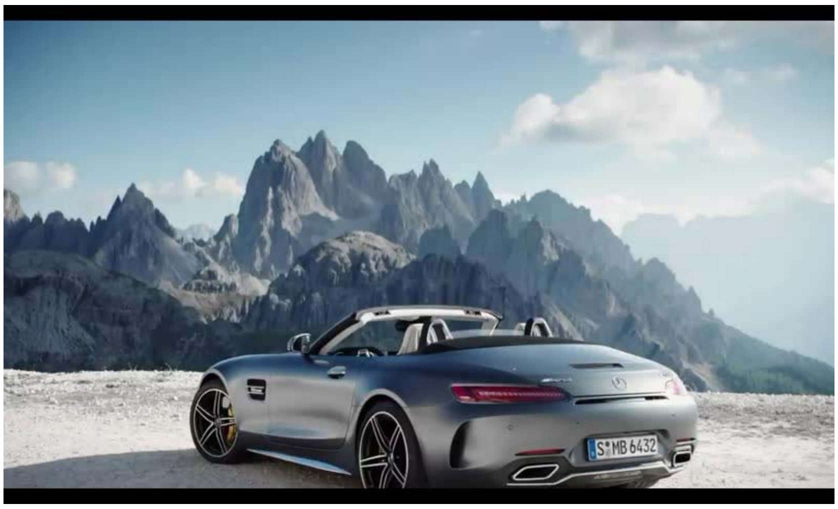


PNG Pic

Step 2:Write JPG pic to SDRAM

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

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

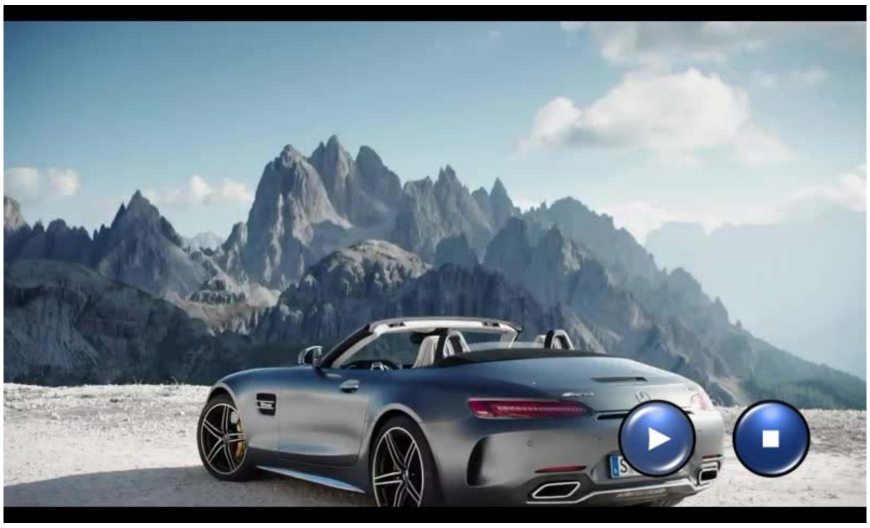


JPG Pic

Step 3:Source 0 = JPG PIC , Source 1 = ARGB Data , do 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);



To do Alpha blending pixel mode

Chapter 5: Picture In Picture Function

5.1: PIP Window

RA8889 supports two PIP windows that can be used with main display window. PIP windows do not support transparent overlays, it just provides users that can enable or disable without overwriting the main display window image data. If the PIP1 and PIP2 windows are overlapped, the PIP1 window is displayed over PIP2 window.

The size and position of PIP windows are specified using registers from REG[2Ah] to REG[3Bh] and REG[11h]. PIP1 and PIP2 window are sharing the same set of registers, and according REG[10h] Bit[4] to select REG [2Ah ~ 3Bh] as PIP1 or PIP2 window's parameters. Function bit will be configured for relative PIP window. PIP windows sizes and star positions are specified in 4 pixel resolution (horizontal) and 1 line resolution.

5.2: PIP Windows Settings

A PIP window position and size is specified using PIP image start address, PIP image Width, PIP Display X/Y coordination, PIP Image X/Y coordination, PIP windows color depth, PIP window width and PIP window height registers. PIP1 and PIP2 window are sharing the same set of registers, and according REG[10h] Bit[4] to select REG [2Ah ~ 3Bh] as PIP1 or PIP2 window's parameters.

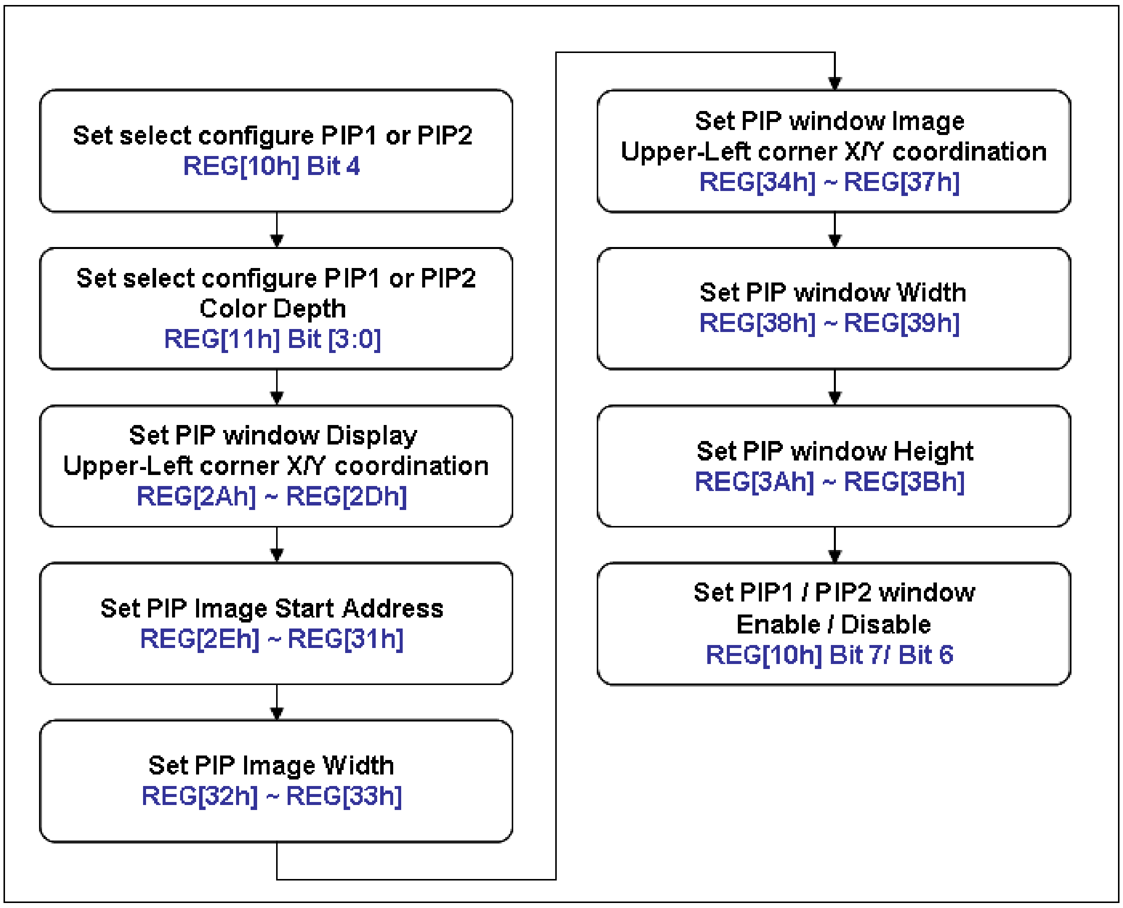

Set select configure PIP1 or PIP2
REG[10h] Bit 4
Set select configure PIP1 or PIP2 Color Depth
REG[11h] Bit [3:0]
Set PIP window Display Upper-Left corner X/Y coordination
REG[2Ah] ~ REG[2Dh]
Set PIP Image Start Address
REG[2Eh] ~ REG[31h]
Set PIP Image Width
REG[32h] ~ REG[33h]
Set PIP window Image Upper-Left corner X/Y coordination
REG[34h] ~ REG[37h]
Set PIP window Width
REG[38h] ~ REG[39h]
Set PIP window Height
REG[3Ah] ~ REG[3Bh]
Set PIP1 / PIP2 window Enable / Disable
REG[10h] Bit 7/ Bit 6

Figure 5-1 Flow Char

5.3: Diagrammatic explanation for PIP function

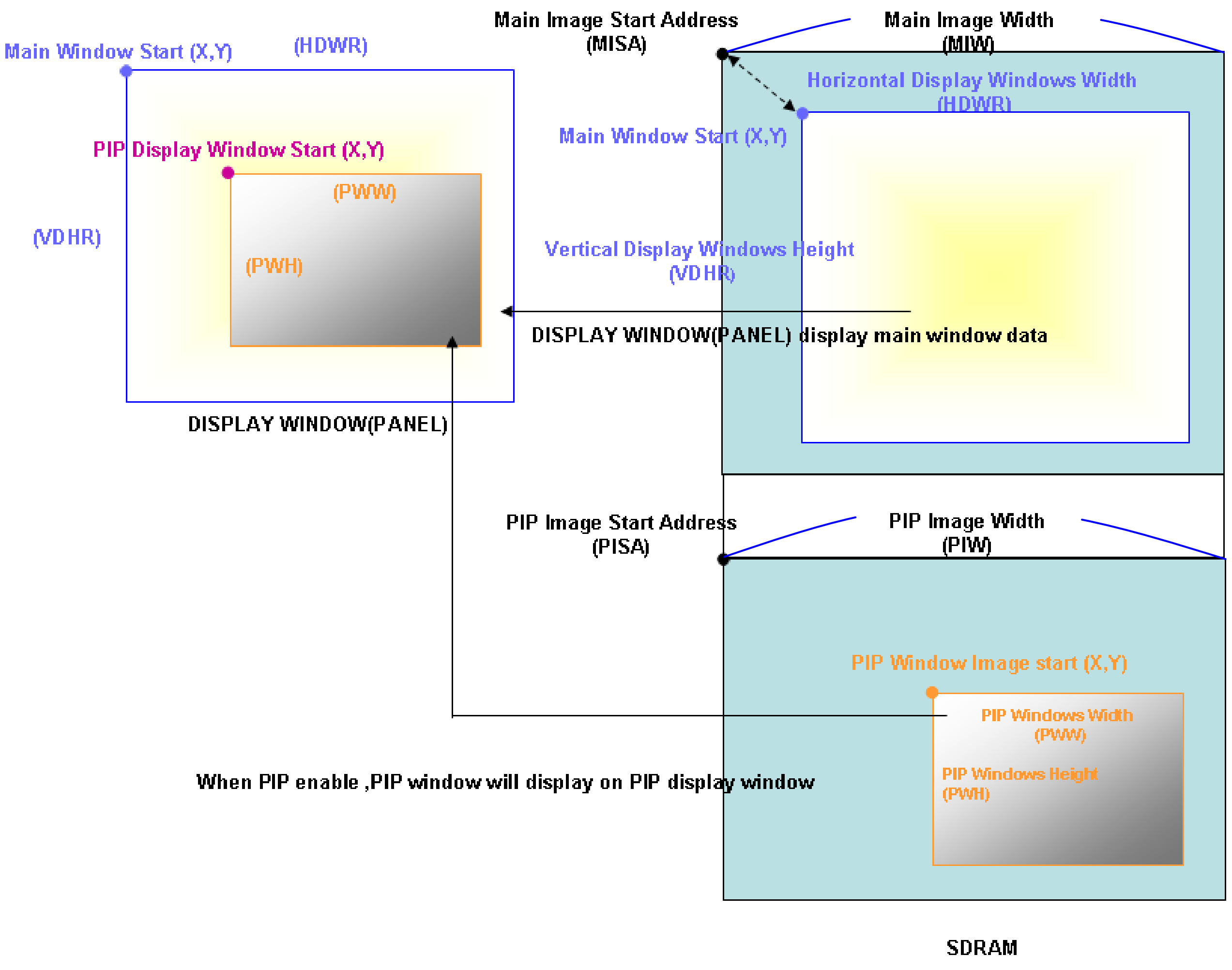


Figure 5-2

5.4:Display results on LCD for illustrating PIP Function API:

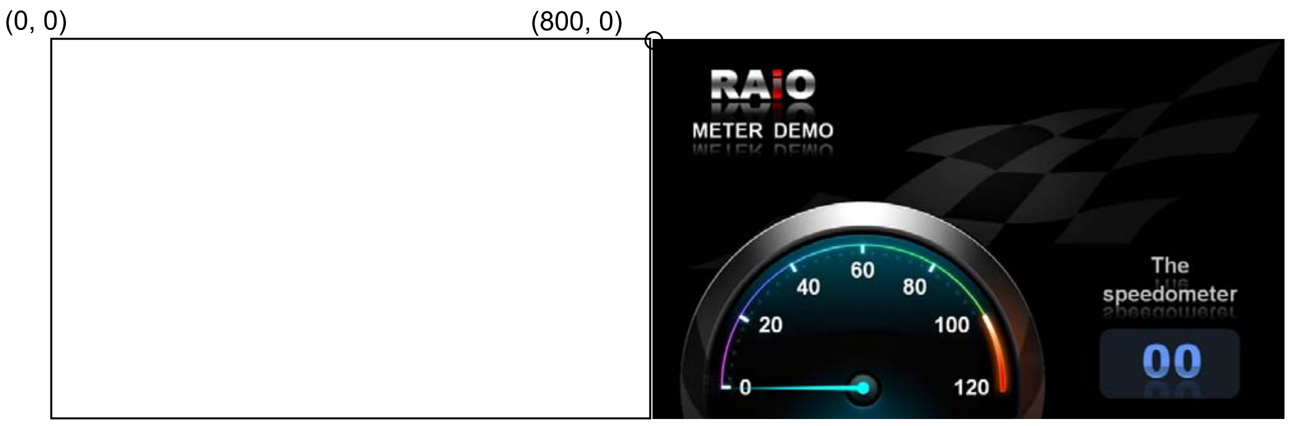
RA8889 supports two PIP windows that can be used with main display window. PIP windows do not support transparent overlays, it just provides users that can enable or disable without overwriting the main display window image data. If the PIP1 and PIP2 windows are overlapped, the PIP1 window is displayed over PIP2 window. This API will help user easy to use picture in picture function.

```
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.
)
```

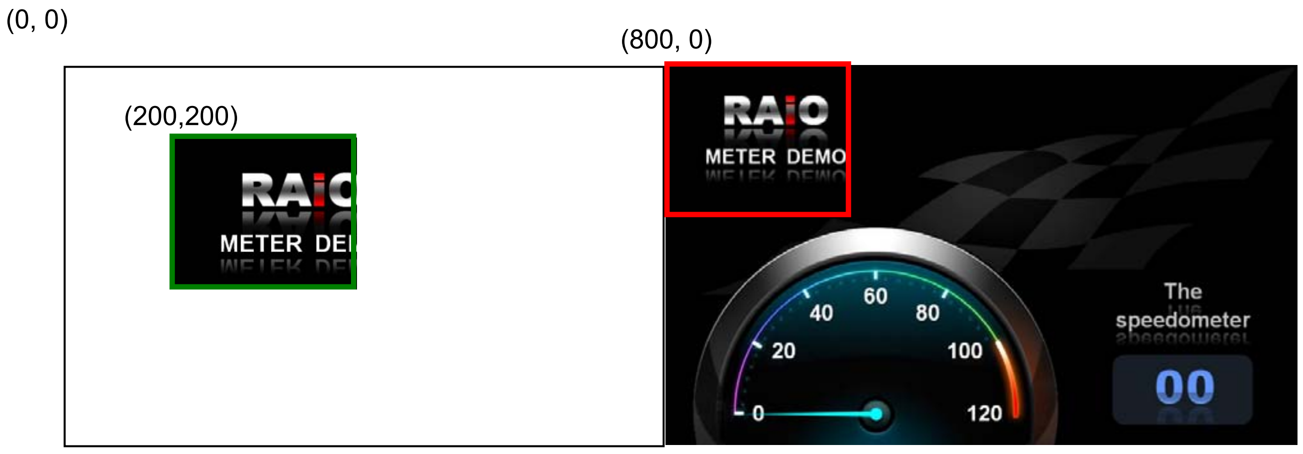
Example: (DMA Reference Chapter 3)

```
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);
```

Step 1 : Write display data into the SDRAM by DMA function



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



Red square is PIP source, Green square is PIP display window

Chapter 6 : Font

RA8889 can support two kinds of font sources into the built-in fonts and external fonts, built-in font support ISO / IEC 8859-1 / 2/4/5 and other fonts. External font set with Shanghai Genitop company (Genitop Inc) part of the serial font ROM, can support multinational fonts or font standard display, such as ASIC, GB12345, GB18030, GB2312 Special, BIG5, UNI-jpn, JIS0208, Latin , Greek, Cyrillic, Arabic, UNICODE, Hebrew, Thai, ISO-8859 and GB2312 Extension etc.

Text input RA8889 can be classified with two sources:

TEXT INPUT

1. Embedded Characters .
2. External Character ROM .

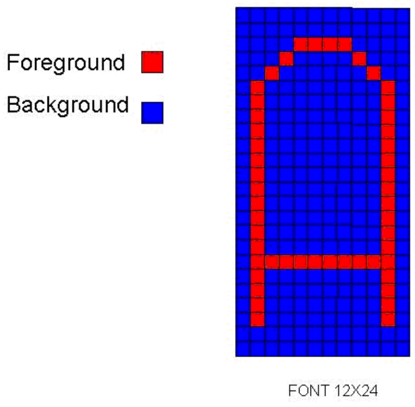


Figure 6-1 : Font Example

The Text have some parameter EX:REG[CCh]~REG[DEh],When you change any font parameter. You can reference below flow chart. The Font color set in Foreground and Background REG[D2]~[D7h].
Ex : We write 64 characters as font1 and 64 characters as font2 to RA8889

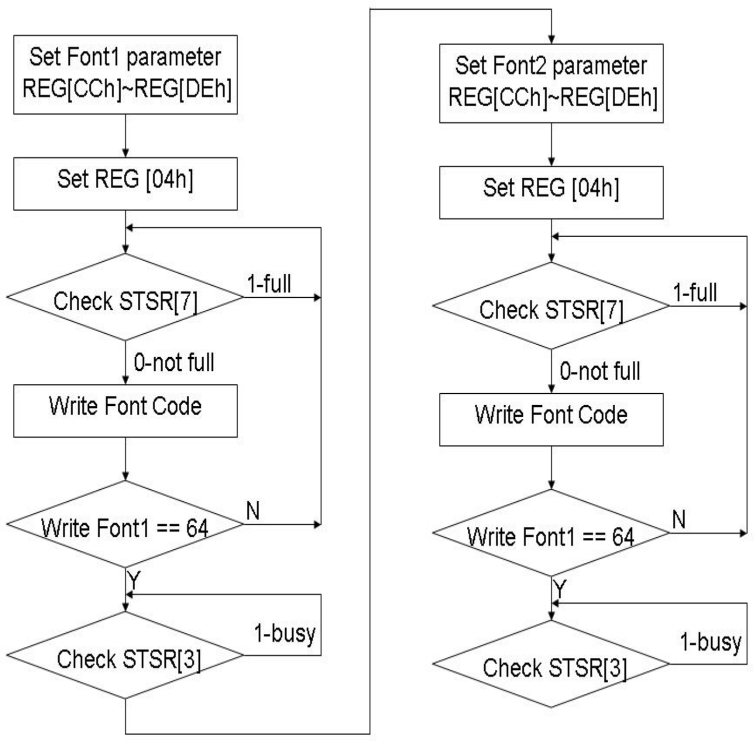


Figure 6-2 : Font display flow chart

Embedded Characters

The RA8889 embedded 12x24 dots ASCII Characters ROM that provides user a convenient way to input characters by ASCII code. The embedded character set supports ISO/IEC 8859-1/2/4/5 coding standards. Besides, user can choose the character foreground color by setting the REG[D2h~D4h] and background color by setting the REG[D5h~D7h]. For the procedure of characters writing please refers to below figure:

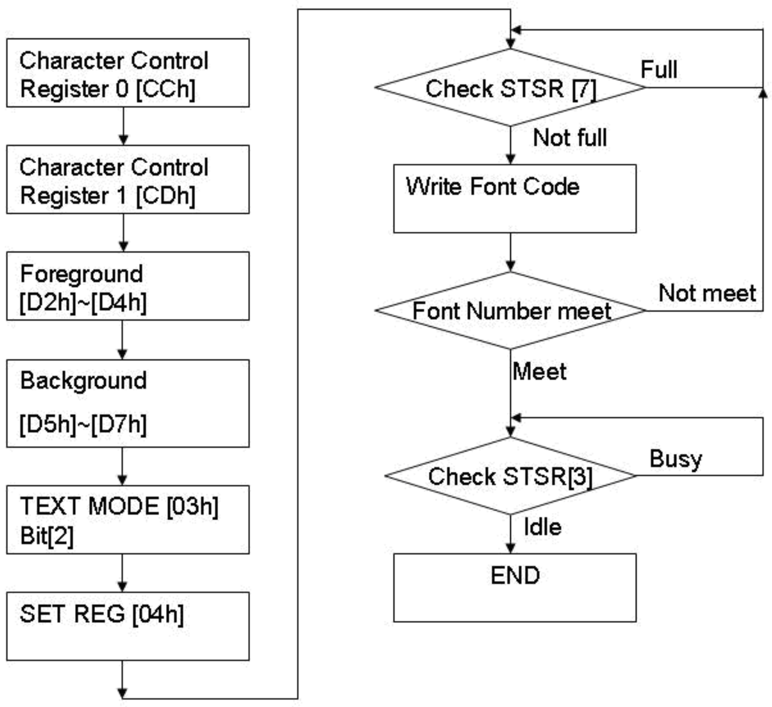


Figure 6-3 : ASCII Character ROM Programming Procedure

Table 6-1 shows the standard character encoding of ISO/IEC 8859-1. ISO means International Organization for Standardization. The ISO/IEC 8859-1, generally called "Latin-1", is the first 8-bit coded character sets that developed by the ISO. It refers to ASCII that consisting of 192 characters from the Latin script in range 0xA0-0xFF. This character encoding is used throughout Western Europe, includes Albanian, Afrikaans, Breton, Danish, Faroese, Frisian, Galician, German, Greenlandic, Icelandic, Irish, Italian, Latin, Luxembourgish, Norwegian, Portuguese, Rhaeto-Romanic, Scottish Gaelic, Spanish, Swedish. English letters with no accent marks also can use ISO/IEC 8859-1. In addition, it is also commonly used in many languages outside Europe, such as Swahili, Indonesian, Malaysian and Tagalong.

In the table, character codes 0x80-0x9F are defined by Microsoft windows, also called CP1252 (WinLatin1).

Table 6-1 : ASCII Block 1(ISO/IEC 8859-1)

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

Table 6-2 shows the standard characters of ISO/IEC 8859-2. ISO/IEC 8859-2 also cited as Latin-2 is the part 2 of the 8-bit coded character sets developed by ISO/IEC 8859. These code values can be used in almost any data interchange system to communicate in the following European languages: Croatian, Czech, Hungarian, Polish, Slovak, Slovenian, and Upper Sorbian. The Serbian, English, German, Latin can use ISO/IEC 8859-2 as well. Furthermore it is suitable to represent some western European languages like Finnish (with the exception of å used in Swedish and Finnish)

Table 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	˙

Table 6-3 shows the standard characters of ISO/IEC 8859-4. ISO/IEC 8859-4 is known as Latin-4 or "North European" is the forth part of the ISO/IEC 8859 8-bit character encoding. It was designed originally to cover Estonian, Greenlandic, Latvian, Lithuanian, and Sami. This character set also supports Danish, English, Finnish, German, Latin, Norwegian, Slovenian, and Swedish.

Table 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	˙

Table 6-4 shows the standard characters of ISO/IEC 8859-5. ISO/IEC 8859-5 is known as is the five part of the ISO/IEC 8859 8-bit character encoding. It was designed originally to cover Bulgarian , Belarusian, Russian, Serbian and Macedonian.

Table 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	Ѕ	І	Ї	Ј	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	ѕ	і	ї	ј	љ	њ	ћ	ќ	§	ў	џ

External Character ROM

RA8889 use External serial ROM interface to provide more characters set for different applications. It is compatible with Character ROM of Genitop Inc., which is a professional Character sets ROM vendor. The supporting product numbers are GT21L16T1W, GT30L16U2W, GT30L24T3Y, GT30L24M1Z, and GT30L32S4W, GT23L24F6Y, GT23L24S1W. According to different product, there are different character's size including 16x16, 24x24, 32x32, and variable width character size in them.

Detailed instructions listed below:

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:Display results on LCD for showing Fonts

6.2.1: Internal Font :

The RA8889 embedded 12x24 dots ASCII Characters ROM that provides user a convenient way to input characters by ASCII code. The embedded character set supports ISO/IEC 8859-1/2/4/5 coding standards.

```
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
)
```

Example :

```
/*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");
```

Virtual display on LCD :

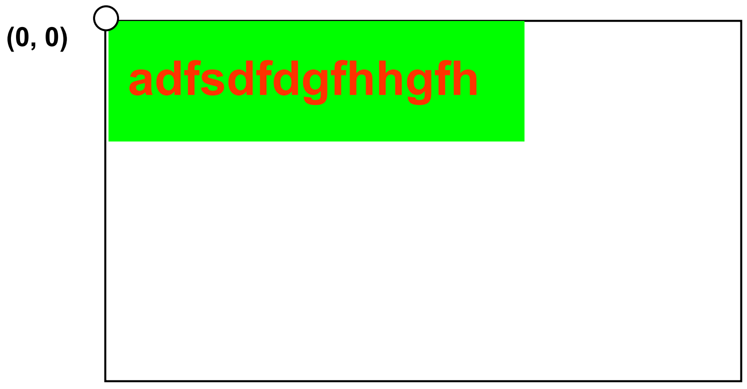


Figure 6-4 Internal Font Example

6.2.2:Print Big5 String by external font rom :

RA8889 use External serial ROM interface to provide more characters set for different applications. It is compatible with Character ROM of Genitop Inc., which is a professional Character sets ROM vendor. The supporting product numbers are GT21L16T1W, GT30L16U2W, GT30L24T3Y, GT30L24M1Z, and GT30L32S4W, GT23L24F6Y, GT23L24S1W. According to different product, there are different character's size including 16x16, 24x24, 32x32, and variable width character size in them.

This API using RA8889 external serial font rom to print Big5 string.

```
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
)
```

Example :

```
/*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
```

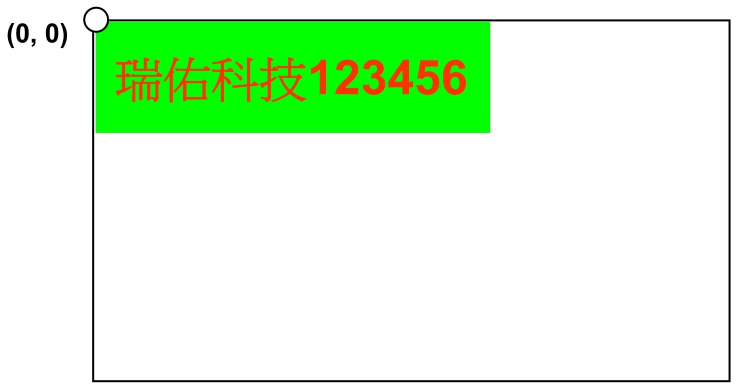


Figure 6-5 Print Big5 Font Example by External Font ROM

6.2.3:Print GB2312 String by external font rom :

RA8889 use External serial ROM interface to provide more characters set for different applications. It is compatible with Character ROM of Genitop Inc., which is a professional Character sets ROM vendor. The supporting product numbers are GT21L16T1W, GT30L16U2W, GT30L24T3Y, GT30L24M1Z, and GT30L32S4W, GT23L24F6Y, GT23L24S1W. According to different product, there are different character's size including 16x16, 24x24, 32x32, and variable width character size in them.
This API using RA8889 external serial font rom to print GB2312 string.

```
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
)
```

Example :

```
/*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
```

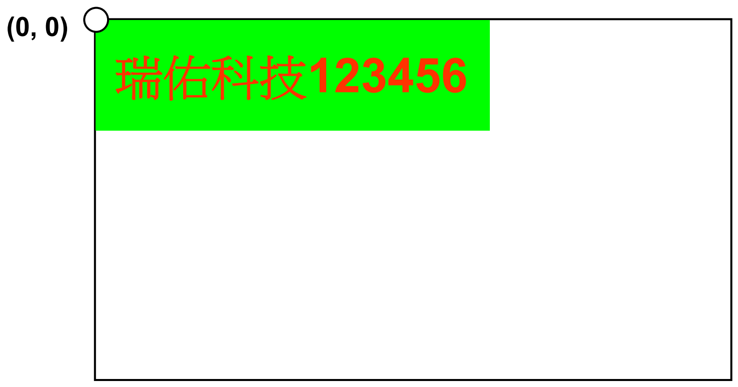


Figure 6-6 Print GB2312 Font Example by External Font ROM

6.3.4:Print GB12345 String by external font rom:

RA8889 use External serial ROM interface to provide more characters set for different applications. It is compatible with Character ROM of Genitop Inc., which is a professional Character sets ROM vendor. The supporting product numbers are GT21L16T1W, GT30L16U2W, GT30L24T3Y, GT30L24M1Z, and GT30L32S4W, GT23L24F6Y, GT23L24S1W. According to different product, there are different character's size including 16x16, 24x24, 32x32, and variable width character size in them.

This API using RA8889 external serial font rom to print GB12345 string.

```
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
)
```

Example :

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

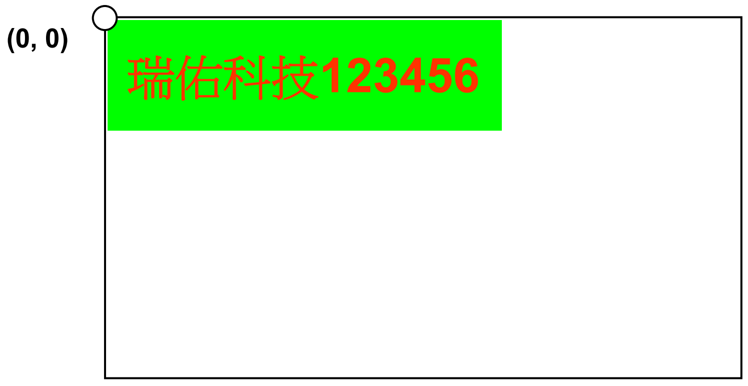


Figure 6-7 Print GB12345 Font Example by External Font ROM

6.3.5:Print Unicode String by external font rom:

RA8889 use External serial ROM interface to provide more characters set for different applications. It is compatible with Character ROM of Genitop Inc., which is a professional Character sets ROM vendor. The supporting product numbers are GT21L16T1W, GT30L16U2W, GT30L24T3Y, GT30L24M1Z, and GT30L32S4W, GT23L24F6Y, GT23L24S1W. According to different product, there are different character's size including 16x16, 24x24, 32x32, and variable width character size in them.
This API using RA8889 external serial font rom to print Unicode string.

```
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)*/
)
```

Example :

```
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");
```

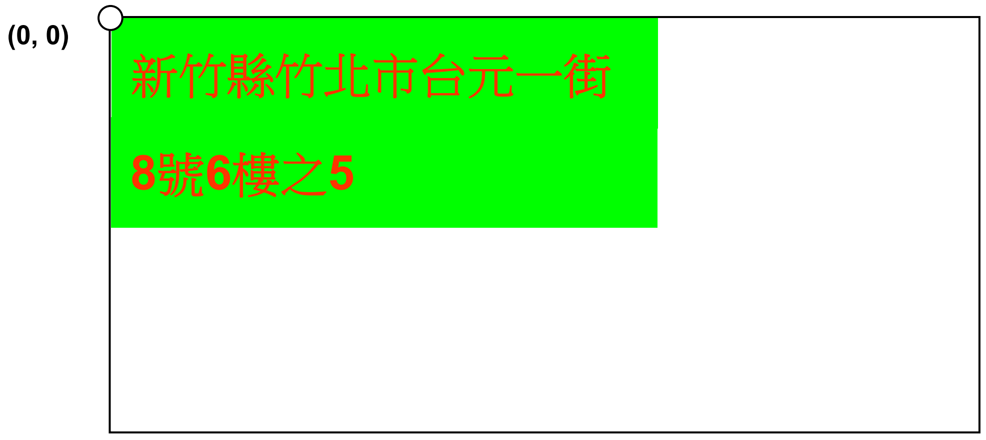


Figure 6-8 : Print Unicode Font Example by External Font ROM

Chapter 7 : PWM (Pulse Width Modulation)

The RA8889 has two 16-bit timers. Timers 0 and 1 have Pulse Width Modulation (PWM) function. The timer 0 has a dead-zone generator, which is used with a large current device.

The timer 0 and 1 share an 8-bit pre-scalar. Each timer has a clock divider, which generates 4 different divided signals (1, 1/2, 1/4 & 1/8).

Finally, through the timer count buffer (TCNTBn) adjusting the PWM output cycle time and the use of timer compare buffer (TCMPBn) values for pulse width modulation (PWM), to produce a stable pulse waveform. Detailed formulas and diagrams such as the following description:

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

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

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

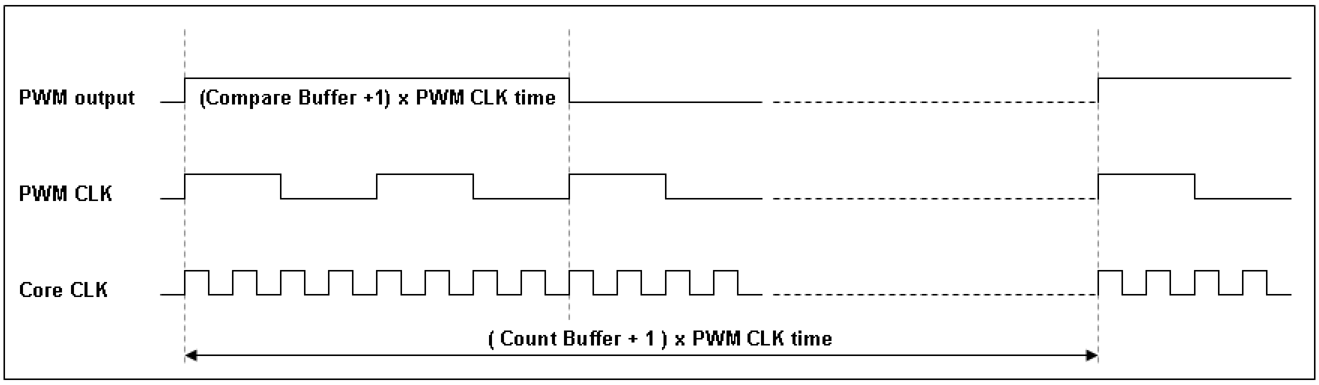

PWM output
(Compare Buffer +1) x PWM CLK time
PWM CLK
Core CLK
(Count Buffer + 1) x PWM CLK time

Figure 7-1 : PWM Timing Diagram

PWM Function 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 */
)
```

Example:

```
When Core CLK = 60MHz
PWM0(1,3,100,1,0);
PWM1(1,3,100,4,3);
```

PWM0 Output:

/*On_off = 1, Enable PWM.

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

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

Oscilloscope waveform:

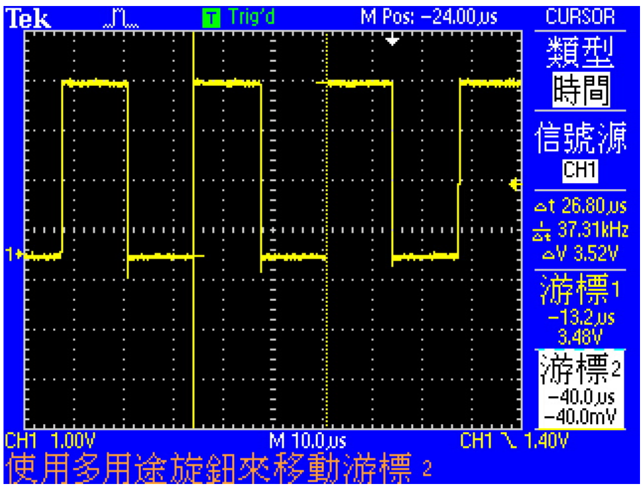


Figure 7-2 : PWM0 output period waveform

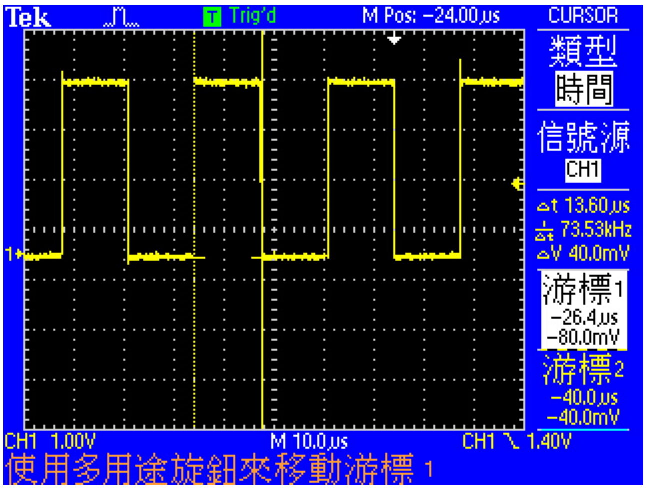


Figure 7-3 : PWM0 output high level time

PWM1 Output:

/*On_off = 1, Enable PWM.

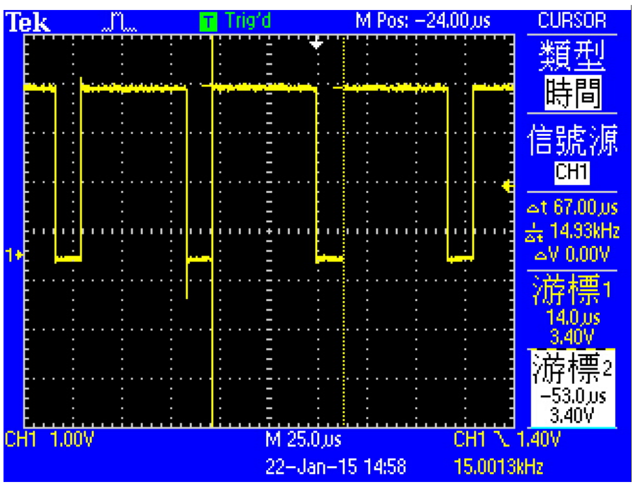
Clock Divided = 3, Prescalar = 100, Count_Buffer = 4, Compare_Buffer = 3.

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

Oscilloscope waveform:



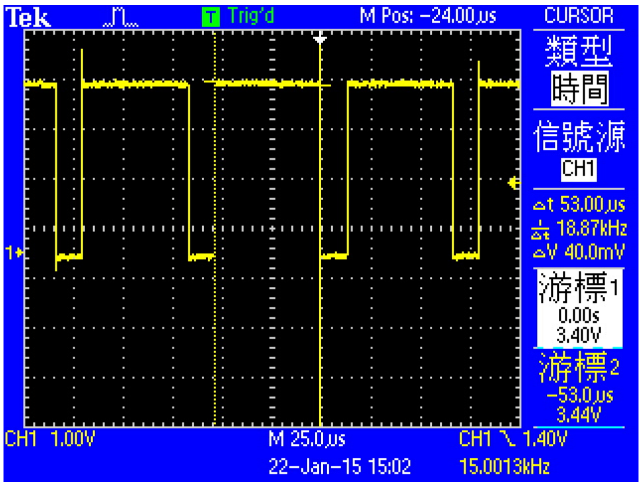


Figure 7-4 : PWM1 output period waveform **Figure 7-5 : PWM1 output high level time**

Chapter 8 : KEY-SCAN Unit

The key-scan interface scan reads the switch data automatically by hardware scanning the key matrix switch. It will help to integrate the system circuit that includes keyboard application. Figure 8-1 shows the basic application circuit of Key-Pad on digital panel package type. RA8889 already built-in pull-up resistors in the pins "XKIN[4:0]" so no external circuit is needed.

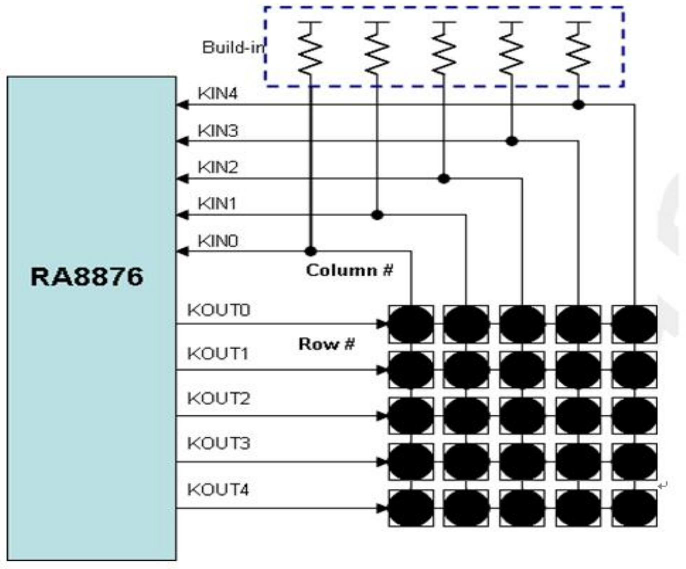


Figure 8-1 : Key-Pad Application

8.1 Operation

The RA8889 Key-Scan controller features are given as below:

1. Supporting with up-to 5x5 Key-Scan Matrix
2. Programmable setting of sampling times and scan frequency of Key-Scan
3. Adjustable long key-press timing
4. Multi-Key is available
   Note: Up-to 2 keys at the same time & restricted 3 keys at the same time (3 keys cannot form 90°)
5. The function of "Key stroke to wake-up the system"

KSCR is the KEYSCAN control and status register, it is used to configure the options for KEYSCAN, such as data sample time, sample clock frequency or long key function enable etc. When key-press is active, user can sense it from the interrupt of KEYSCAN. The status bit of KSCR2 bit1~0 will update the number of current key press. Then user can get the key code directly from KSDR.

Note : "Normal key" means a key press that qualified by the sample time of RA8889. "Long Key" means a key press that keeps "pressed" for a specified long time period. That is, a "Long Key" must be a "Normal Key" first. Sometimes they need to be separated for some applications.

Table 8-1 is the key code mapping to key-pad matrix for normal press. The key code will be stored in KSDR0~2 when key was pressed. If it was a long time press, then the key code is show as Table 8-2.

Table 8-1
Note : "Normal key" means a key press that qualified by the sample time of RA8889. "Long Key" means a key press that keeps "pressed" for a specified long time period. That is, a "Long Key" must be a "Normal Key" first. Sometimes they need to be separated for some applications.

Table 8-1 is the key code mapping to key-pad matrix for normal press. The key code will be stored in KSDR0~2 when key was pressed. If it was a long time press, then the key code is show as Table 8-2.

Table 8-1: Key Code Mapping Table (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

Table 8-2: Key Code Mapping Table (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

When the multi-key function is applied, the up to 3 pressed keys data will be saved in the registers (KSDR0, KSDR1 and KSDR2). Note that the order of keys saving is determined by the position (or key code) of the keys, not the order of keys being pressed; please refer to the following example:

Press the key-code in turn of 0x34, 0x00 and 0x22, press multi-key at the same time, the key-code will be saved in KSDR0~2:

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

The basic features of above Key-Scan settings are introduced as follows:

Table 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]

Enabling the Key-Scan functions, programmer can use following methods to check keystroke.

1) Software check method: to know the key be pressed from keeping check the status of Key-Scan.
2) Hardware check method: to know the key be pressed from external interrupt signal.

Please be aware that when key-scan interrupt enable bit(INTEN bit[3]) set as “1” and key event of interrupt happens, the interrupt status of Key-Scan (bit[3] of INTF) is always set to “1”, no matter which method is used, programmer have to clear the status bit to 0 after reading the correct Key Code, otherwise the interrupt will be kept and no more interrupt will be generated again.

Besides, RA8889 allows the “Key-stroke wakeup function” for power saving mode. By setting the function on, any legal key-stroke event can wakeup RA8889 from sleep mode. To sense the wakeup event, RA8889 can assert hardware interrupt for MPU which can do software polling from RA8889.

The flowchart of register settings for above applications are shown as following:

1. Software Method

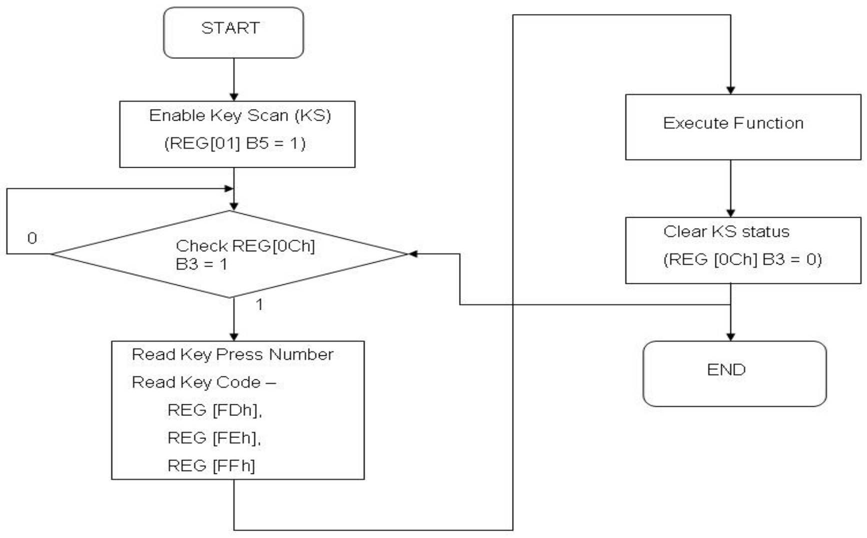


Figure 8-2 : Key-Scan Flowchart for Software Polling

2. Hardware Method

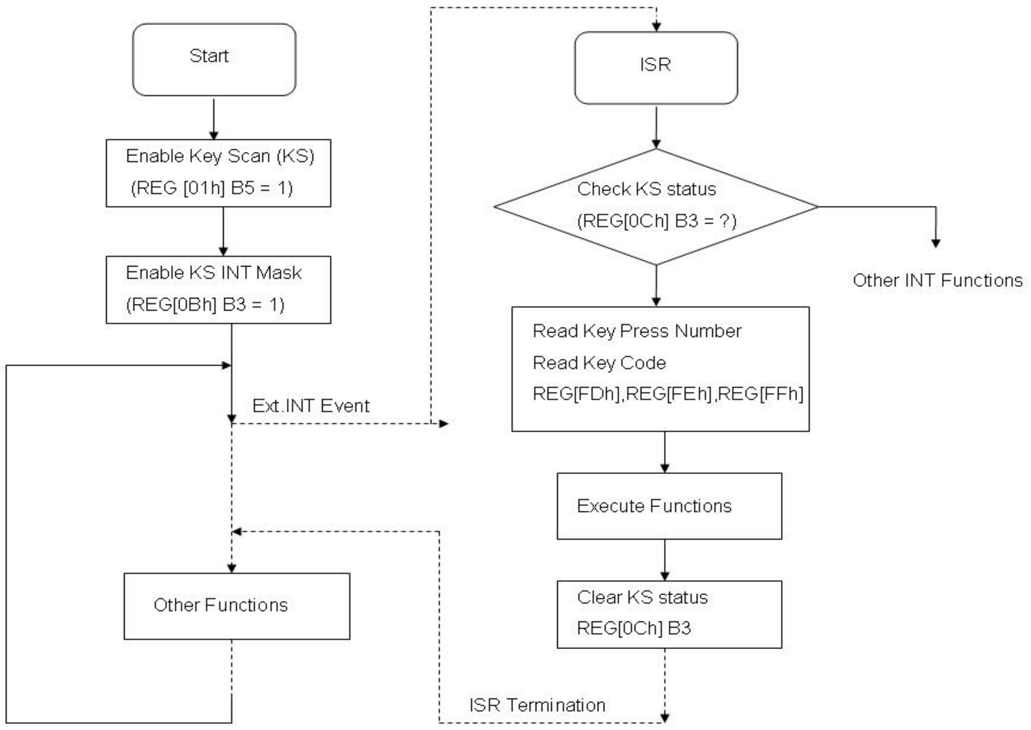


Figure 8-3 : Key-Scan for Hardware Interrupt

KEY SCAN function API:

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

Excute:

```
    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
```

Condition:

x: Displays the X coordinate of the Key Code = 0

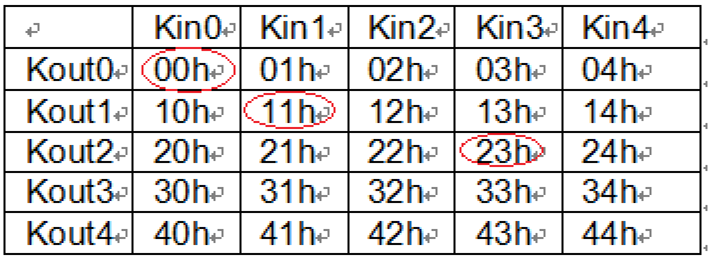
y: Displays the Y coordinate of the Key Code = 0

FontColor: Keycode color = block

BackGroundColor: Keycode background color = white

Example:

If you press the following three buttons



	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 Display:

```
001123
```

Appendix 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);
```

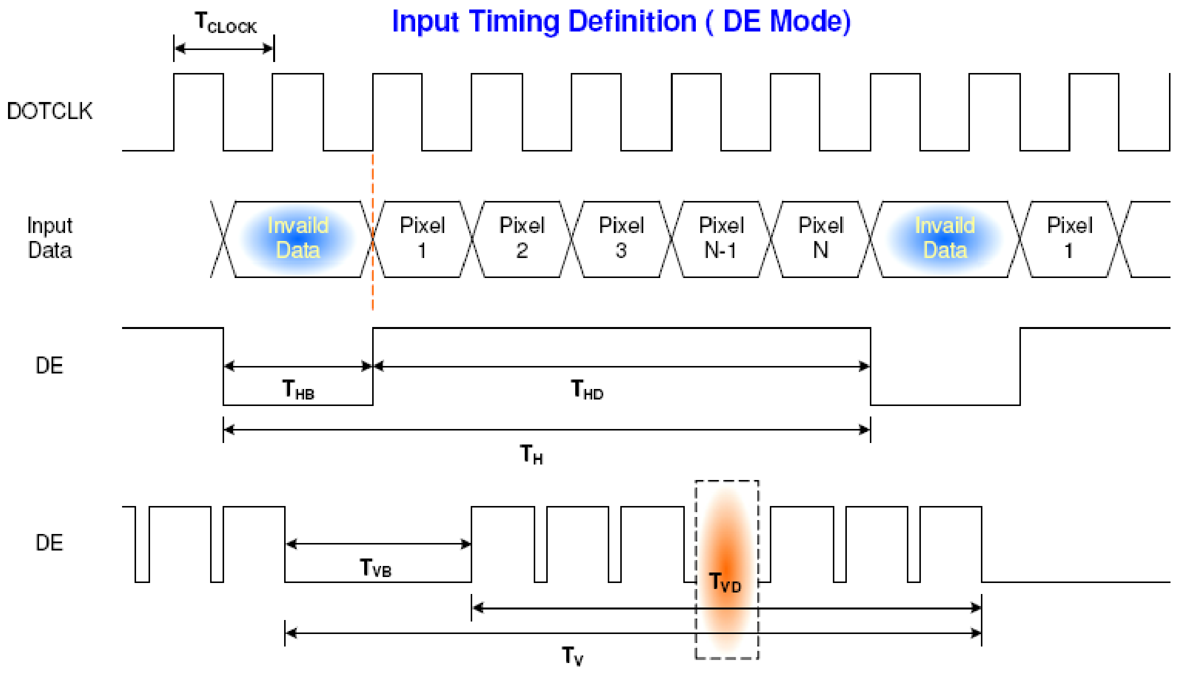
```
            }

            Enable_PLL();
            delay_ms(100);

}
```

Appendix 2 : LCD initialization:

B116XW03 (Resolution 1366X768):

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

B116XW03 Timing



Clock Frequency = SCLK = 70MHz

T_{HD} = Horizontal display width = 1366

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

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

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

void Initial_AUO1366x768(void)

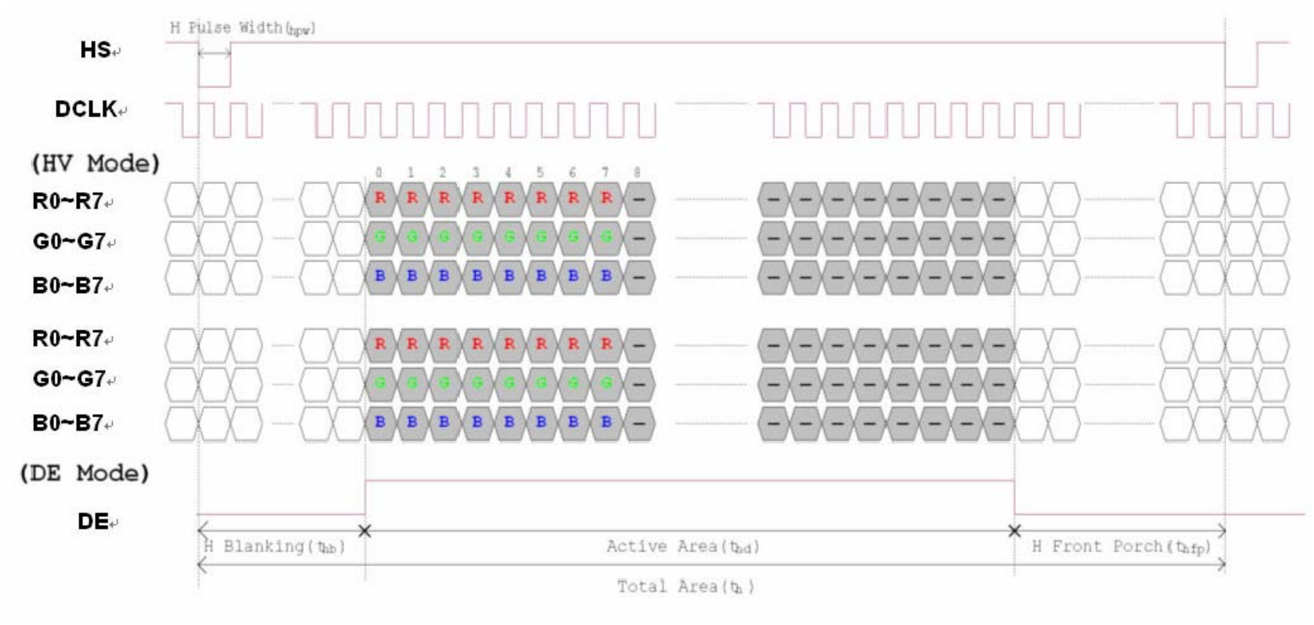
```
{
DE_PCLK_Rising();               //PDAT, DE, HSYNC etc. Panel fetches PDAT at the rising edge of PCLK
PDATA_Set_RGB();                //Parallel XPDAT[23:0] Output Sequence is set as RGB
DE_High_Active();               //Set XDE Polarity worked as High active.

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



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

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

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

//Vertical Display Height(Line) = VDHR + 1 = (512 + 255 ) +1 =768
LCD_CmdWrite(0x1A);                   //VDHR0 //Vertical Display Height Bit[7:0]
LCD_DataWrite(0xff);                  //Vertical Display Height(Line) = VDHR + 1    =  255
Delay_us(10);
LCD_CmdWrite(0x1B);                    //VDHR1 //Vertical Display Height Bit[10:8] = 512
LCD_DataWrite(0x02);                    //Vertical Display Height(Line) = VDHR + 1
Delay_us(100);

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

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

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

AT070TN92 Timing :

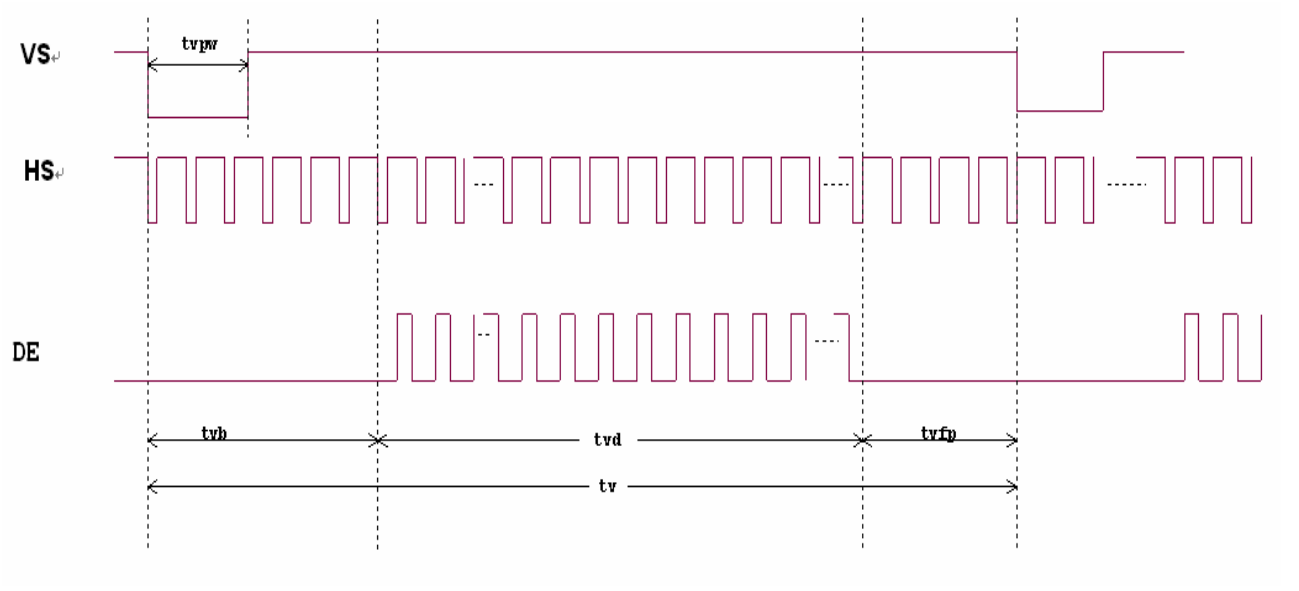
Item	symbol	value			Unit	Remark
		Min.	Typ.	Max.		
Horizontal Display Area	thd		800		DCLK	
DCLK Frequency(PCLK)	fclk	26.4	33.3	46.8	MHz	
One Horizontal Line	th	862	1056	1200	DCLK	
HS pulse width	thpw	1	-	40	DCLK	
HS Blanking(Back Porch)	thb	46	46	46	DCLK	
HS Front Porch	thfp	16	210	354	DCLK	

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

VS Front Porch	tvfp	7	22	147	TH	



Horizontal Input Timing Diagram



Vertical Input Timing Diagram

```
DE_PCLK_Falling();
PDATA_Set_RGB();
LCD_CmdWrite(0x13);          // de --> low active
LCD_DataWrite(0x00);
delay_us(100);
```

```
//Horizontal display width(pixels) = (HDWR + 1) * 8 + HDWFTR Maximum value cannot over 2048
//Horizontal display width(pixels) = (0x63+1) * 8 + 0 = 800
LCD_CmdWrite(0x14);        //HDWR//Horizontal Display Width Setting Bit[6:0]
LCD_DataWrite(0x63);       //Horizontal display width(pixels) = (HDWR + 1)*8
delay_us(100);
LCD_CmdWrite(0x15);       //Horizontal Display Width Fine Tuning (HDWFT) [3:0]
LCD_DataWrite(0x00);
delay_us(100);

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

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

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

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

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

```
LCD_CmdWrite(0x1D);        //VNDR1 //Vertical Non-Display Period Bit [8]
LCD_DataWrite(0x00);       //Vertical Non-Display area = (VNDR + 1)
delay_us(100);

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

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

Appendix 3 : RAIO self-made character

RAIO self-made character, ASCII fonts available in three sizes, respectively 8x12,16x24,32x48, the API support 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 in this chapter, but does not support 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
)
```

Example:

```
//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");
```

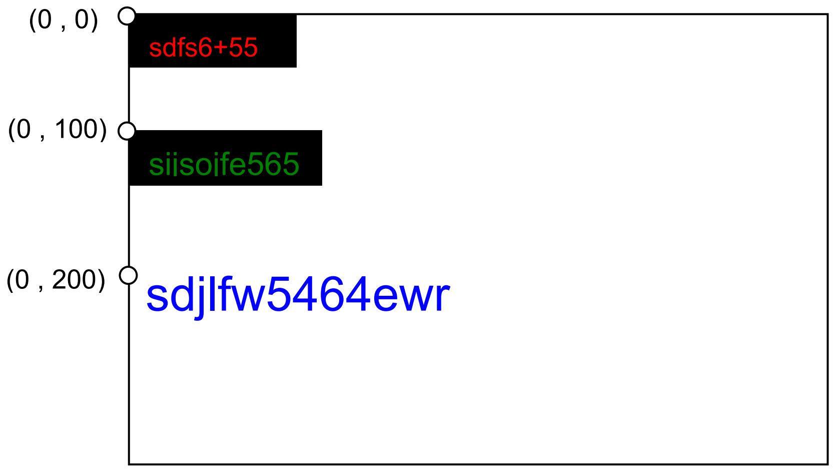


Figure 8-1 : self-made character