

**RAiO**

**RA8871M**

**Application Programming Interface**

January 3, 2018

RAiO Technology Inc.

©Copyright RAiO Technology Inc. 2017

**CONTENTS**

<a href="#">1. INTRODUCTION.....</a>	<a href="#">3</a>
<a href="#">2. CHAPTER 1 : MCU WRITE DATA AND DISPLAY INPUT DATA FORMAT.....</a>	<a href="#">4</a>
<a href="#">3. CHAPTER 2 : DRAW FUNCTIONS .....</a>	<a href="#">17</a>
<a href="#">4. CHAPTER 3 : DMA IN BLOCK MODE .....</a>	<a href="#">38</a>
<a href="#">5. CHAPTER 4 : BLOCK TRANSFER ENGINE .....</a>	<a href="#">42</a>
<a href="#">6. CHAPTER 5 : PICTURE IN PICTURE FUNCTION.....</a>	<a href="#">90</a>
<a href="#">7. CHAPTER 6 : FONT .....</a>	<a href="#">95</a>
<a href="#">8. CHAPTER 7 : PWM(PULSE WIDTH MODULATION).....</a>	<a href="#">111</a>
<a href="#">9. CHAPTER 8 : KEY-SCAN UNIT .....</a>	<a href="#">114</a>
<a href="#">10. APPENDIX 1 : PLL INITIALIZATION.....</a>	<a href="#">119</a>
<a href="#">11. APPENDIX 2 : SDRAM INITIALIZATION .....</a>	<a href="#">121</a>
<a href="#">12. APPENDIX 3 : LCD INITIALIZATION .....</a>	<a href="#">122</a>
<a href="#">13. APPENDIX 4 : RAI0 USER DEFINE FONT.....</a>	<a href="#">127</a>

## Introduction

In this document, we will introduce 6 RA8871M functions. About MCU Write Data and Display Input Data Format, Draw, DMA, BTE, PIP, font and PWM function...etc, and provide 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, SDRAM initialization or LCD initialization, please refer appendix 1/2/3.

**MCU I/F Protocol** : RA8871M provide Parallel 8080、Parallel 6800、SPI – 3 wire、SPI – 4 wire、IIC for MCU Interface.

**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** : RA8871M 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 80MHz).

SDRAM CLK must > Core CLK.

In 16bpp color depth : Core CLK  $\geq 1.5 * \text{Pixel CLK}$

In 24bpp color depth : Core CLK  $\geq 2 * \text{Pixel CLK}$

## Chapter 1: MCU Write Data and Display Input Data Format

Overview:

Switch Layer Introduction

RA8871M provide 6 type 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.

**Switch Layer Introduction:**

RA8871M provides three layers, Layer1 address is 786432, Layer2 address is 1835008, Layer3 address is 2883584, here provides two sets of API, so that users can directly select the layers you want to write, and you want to display. The program as follows:

```
void Write_Layer(unsigned char Layer)
{
    unsigned long address;
    switch(Layer)
    {
        case 1:
            address = Layer1;    //Layer1 address define in userdef.h
            break;
        case 2:
            address = Layer2;    //Layer2 address define in userdef.h
            break;
        case 3:
            address = Layer3;    //Layer3 address define in userdef.h
            break;
    }
    Canvas_Image_Start_address(address);    //set data write address
    Goto_Pixel_XY(0,0);
}

void Show_Layer(unsigned char Layer)
{
    unsigned long address;
    switch(Layer)
    {
        case 1:
            address = Layer1;    //Layer1 address define in userdef.h
            break;
        case 2:
            address = Layer2;    //Layer2 address define in userdef.h
            break;
        case 3:
            address = Layer3;    //Layer3 address define in userdef.h
            break;
    }
}
```

```
Main_Image_Start_Address(address); //set display screen address
Main_Window_Start_XY(0,0);
}
```

**Example 1:**

```
Write_Layer(1);
```

```
Show_Layer(1);
```

Program Results: Set writing address is layer 1, and display screen address is layer 3.

**Example 2:**

```
Write_Layer(2);
```

```
Show_Layer(3);
```

Program Results: Set writing address is layer 2, and display screen address is layer 3.

**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 <sub>0</sub> <sup>r</sup>	R <sub>0</sub> <sup>b</sup>	R <sub>0</sub> <sup>b</sup>	G <sub>0</sub> <sup>r</sup>	G <sub>0</sub> <sup>b</sup>	G <sub>0</sub> <sup>b</sup>	B <sub>0</sub> <sup>r</sup>	B <sub>0</sub> <sup>b</sup>
2	R <sub>1</sub> <sup>r</sup>	R <sub>1</sub> <sup>b</sup>	R <sub>1</sub> <sup>b</sup>	G <sub>1</sub> <sup>r</sup>	G <sub>1</sub> <sup>b</sup>	G <sub>1</sub> <sup>b</sup>	B <sub>1</sub> <sup>r</sup>	B <sub>1</sub> <sup>b</sup>
3	R <sub>2</sub> <sup>r</sup>	R <sub>2</sub> <sup>b</sup>	R <sub>2</sub> <sup>b</sup>	G <sub>2</sub> <sup>r</sup>	G <sub>2</sub> <sup>b</sup>	G <sub>2</sub> <sup>b</sup>	B <sub>2</sub> <sup>r</sup>	B <sub>2</sub> <sup>b</sup>
4	R <sub>3</sub> <sup>r</sup>	R <sub>3</sub> <sup>b</sup>	R <sub>3</sub> <sup>b</sup>	G <sub>3</sub> <sup>r</sup>	G <sub>3</sub> <sup>b</sup>	G <sub>3</sub> <sup>b</sup>	B <sub>3</sub> <sup>r</sup>	B <sub>3</sub> <sup>b</sup>
5	R <sub>4</sub> <sup>r</sup>	R <sub>4</sub> <sup>b</sup>	R <sub>4</sub> <sup>b</sup>	G <sub>4</sub> <sup>r</sup>	G <sub>4</sub> <sup>b</sup>	G <sub>4</sub> <sup>b</sup>	B <sub>4</sub> <sup>r</sup>	B <sub>4</sub> <sup>b</sup>
6	R <sub>5</sub> <sup>r</sup>	R <sub>5</sub> <sup>b</sup>	R <sub>5</sub> <sup>b</sup>	G <sub>5</sub> <sup>r</sup>	G <sub>5</sub> <sup>b</sup>	G <sub>5</sub> <sup>b</sup>	B <sub>5</sub> <sup>r</sup>	B <sub>5</sub> <sup>b</sup>

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

API :

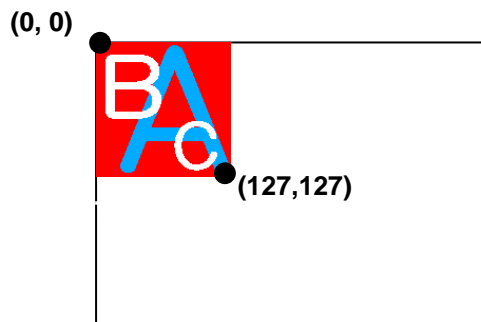
Using 8 bits MCU interface and 8bpp color depth. MCU write data into SDRAM.

```
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 :

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
MPU8_8bpp_Memory_Write(0,0,128,128 ,gImage_8);
//gImage_8 is 8bit array data ,8bpp color deepth ,size 128x128 picture data.
```

The example dispaly on LCD(Layer1):



**Figure 1.1 :Using 8 bits MCU interface and 8bpp color depth. MCU write data into SDRAM(Layer1).**

**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 <sub>0</sub> <sup>4</sup>	G <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>2</sup>	B <sub>0</sub> <sup>7</sup>	B <sub>0</sub> <sup>6</sup>	B <sub>0</sub> <sup>5</sup>	B <sub>0</sub> <sup>4</sup>	B <sub>0</sub> <sup>3</sup>
2	R <sub>0</sub> <sup>7</sup>	R <sub>0</sub> <sup>6</sup>	R <sub>0</sub> <sup>5</sup>	R <sub>0</sub> <sup>4</sup>	R <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>7</sup>	G <sub>0</sub> <sup>6</sup>	G <sub>0</sub> <sup>5</sup>
3	G <sub>1</sub> <sup>4</sup>	G <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>2</sup>	B <sub>1</sub> <sup>7</sup>	B <sub>1</sub> <sup>6</sup>	B <sub>1</sub> <sup>5</sup>	B <sub>1</sub> <sup>4</sup>	B <sub>1</sub> <sup>3</sup>
4	R <sub>1</sub> <sup>7</sup>	R <sub>1</sub> <sup>6</sup>	R <sub>1</sub> <sup>5</sup>	R <sub>1</sub> <sup>4</sup>	R <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>7</sup>	G <sub>1</sub> <sup>6</sup>	G <sub>1</sub> <sup>5</sup>
5	G <sub>2</sub> <sup>4</sup>	G <sub>2</sub> <sup>3</sup>	G <sub>2</sub> <sup>2</sup>	B <sub>2</sub> <sup>7</sup>	B <sub>2</sub> <sup>6</sup>	B <sub>2</sub> <sup>5</sup>	B <sub>2</sub> <sup>4</sup>	B <sub>2</sub> <sup>3</sup>
6	R <sub>2</sub> <sup>7</sup>	R <sub>2</sub> <sup>6</sup>	R <sub>2</sub> <sup>5</sup>	R <sub>2</sub> <sup>4</sup>	R <sub>2</sub> <sup>3</sup>	G <sub>2</sub> <sup>7</sup>	G <sub>2</sub> <sup>6</sup>	G <sub>2</sub> <sup>5</sup>

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

API :

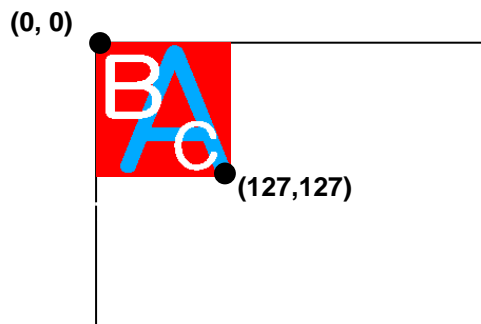
Using 8 bits MCU interface and 16bpp color depth. MCU write data into SDRAM.

```
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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1); //set LCD display layer. Reference Page.5~6
MPU8_16bpp_Memory_Write (0,0,128,128,gImage_16);
//gImage_16 is 8bit array data , 16bpp color deepth ,size 128x128 picture data.
```

The example dispaly on LCD(Layer1):



**Figure 1.2 : Using 8 bits MCU interface and 16bpp color depth. MCU write data into SDRAM(Layer1).**



**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 <sub>0</sub> <sup>7</sup>	B <sub>0</sub> <sup>6</sup>	B <sub>0</sub> <sup>5</sup>	B <sub>0</sub> <sup>4</sup>	B <sub>0</sub> <sup>3</sup>	B <sub>0</sub> <sup>2</sup>	B <sub>0</sub> <sup>1</sup>	B <sub>0</sub> <sup>0</sup>
2	G <sub>0</sub> <sup>7</sup>	G <sub>0</sub> <sup>6</sup>	G <sub>0</sub> <sup>5</sup>	G <sub>0</sub> <sup>4</sup>	G <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>2</sup>	G <sub>0</sub> <sup>1</sup>	G <sub>0</sub> <sup>0</sup>
3	R <sub>0</sub> <sup>7</sup>	R <sub>0</sub> <sup>6</sup>	R <sub>0</sub> <sup>5</sup>	R <sub>0</sub> <sup>4</sup>	R <sub>0</sub> <sup>3</sup>	R <sub>0</sub> <sup>2</sup>	R <sub>0</sub> <sup>1</sup>	R <sub>0</sub> <sup>0</sup>
4	B <sub>1</sub> <sup>7</sup>	B <sub>1</sub> <sup>6</sup>	B <sub>1</sub> <sup>5</sup>	B <sub>1</sub> <sup>4</sup>	B <sub>1</sub> <sup>3</sup>	B <sub>1</sub> <sup>2</sup>	B <sub>1</sub> <sup>1</sup>	B <sub>1</sub> <sup>0</sup>
5	G <sub>1</sub> <sup>7</sup>	G <sub>1</sub> <sup>6</sup>	G <sub>1</sub> <sup>5</sup>	G <sub>1</sub> <sup>4</sup>	G <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>2</sup>	G <sub>1</sub> <sup>1</sup>	G <sub>1</sub> <sup>0</sup>
6	R <sub>1</sub> <sup>7</sup>	R <sub>1</sub> <sup>6</sup>	R <sub>1</sub> <sup>5</sup>	R <sub>1</sub> <sup>4</sup>	R <sub>1</sub> <sup>3</sup>	R <sub>1</sub> <sup>2</sup>	R <sub>1</sub> <sup>1</sup>	R <sub>1</sub> <sup>0</sup>

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

API :

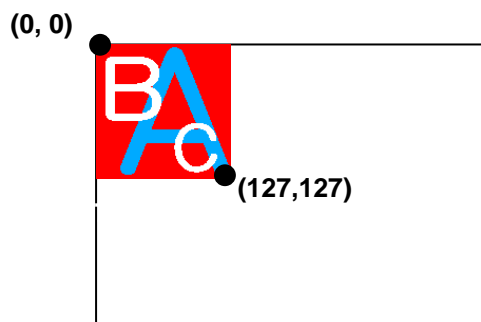
Using 8 bits MCU interface and 24bpp color depth. MCU write data into SDRAM.

```
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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1); //set LCD display layer. Reference Page.5~6
MPU8_24bpp_Memory_Write (0,0,128,128 ,gImage_24);
//gImage_24 is 8bit array data , 24bpp color deepth ,size 128x128 picture data.
```

The example display on LCD(Layer1):



**Figure 1.3 : Using 8 bits MCU interface and 24bpp color depth. MCU write data into SDRAM(Layer1)**

### 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 <sub>0</sub> <sup>7</sup>	R <sub>0</sub> <sup>6</sup>	R <sub>0</sub> <sup>5</sup>	R <sub>0</sub> <sup>4</sup>	R <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>7</sup>	G <sub>0</sub> <sup>6</sup>	G <sub>0</sub> <sup>5</sup>	G <sub>0</sub> <sup>4</sup>	G <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>2</sup>	B <sub>0</sub> <sup>7</sup>	B <sub>0</sub> <sup>6</sup>	B <sub>0</sub> <sup>5</sup>	B <sub>0</sub> <sup>4</sup>	B <sub>0</sub> <sup>3</sup>
2	R <sub>1</sub> <sup>7</sup>	R <sub>1</sub> <sup>6</sup>	R <sub>1</sub> <sup>5</sup>	R <sub>1</sub> <sup>4</sup>	R <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>7</sup>	G <sub>1</sub> <sup>6</sup>	G <sub>1</sub> <sup>5</sup>	G <sub>1</sub> <sup>4</sup>	G <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>2</sup>	B <sub>1</sub> <sup>7</sup>	B <sub>1</sub> <sup>6</sup>	B <sub>1</sub> <sup>5</sup>	B <sub>1</sub> <sup>4</sup>	B <sub>1</sub> <sup>3</sup>
3	R <sub>2</sub> <sup>7</sup>	R <sub>2</sub> <sup>6</sup>	R <sub>2</sub> <sup>5</sup>	R <sub>2</sub> <sup>4</sup>	R <sub>2</sub> <sup>3</sup>	G <sub>2</sub> <sup>7</sup>	G <sub>2</sub> <sup>6</sup>	G <sub>2</sub> <sup>5</sup>	G <sub>2</sub> <sup>4</sup>	G <sub>2</sub> <sup>3</sup>	G <sub>2</sub> <sup>2</sup>	B <sub>2</sub> <sup>7</sup>	B <sub>2</sub> <sup>6</sup>	B <sub>2</sub> <sup>5</sup>	B <sub>2</sub> <sup>4</sup>	B <sub>2</sub> <sup>3</sup>
4	R <sub>3</sub> <sup>7</sup>	R <sub>3</sub> <sup>6</sup>	R <sub>3</sub> <sup>5</sup>	R <sub>3</sub> <sup>4</sup>	R <sub>3</sub> <sup>3</sup>	G <sub>3</sub> <sup>7</sup>	G <sub>3</sub> <sup>6</sup>	G <sub>3</sub> <sup>5</sup>	G <sub>3</sub> <sup>4</sup>	G <sub>3</sub> <sup>3</sup>	G <sub>3</sub> <sup>2</sup>	B <sub>3</sub> <sup>7</sup>	B <sub>3</sub> <sup>6</sup>	B <sub>3</sub> <sup>5</sup>	B <sub>3</sub> <sup>4</sup>	B <sub>3</sub> <sup>3</sup>
5	R <sub>4</sub> <sup>7</sup>	R <sub>4</sub> <sup>6</sup>	R <sub>4</sub> <sup>5</sup>	R <sub>4</sub> <sup>4</sup>	R <sub>4</sub> <sup>3</sup>	G <sub>4</sub> <sup>7</sup>	G <sub>4</sub> <sup>6</sup>	G <sub>4</sub> <sup>5</sup>	G <sub>4</sub> <sup>4</sup>	G <sub>4</sub> <sup>3</sup>	G <sub>4</sub> <sup>2</sup>	B <sub>4</sub> <sup>7</sup>	B <sub>4</sub> <sup>6</sup>	B <sub>4</sub> <sup>5</sup>	B <sub>4</sub> <sup>4</sup>	B <sub>4</sub> <sup>3</sup>
6	R <sub>5</sub> <sup>7</sup>	R <sub>5</sub> <sup>6</sup>	R <sub>5</sub> <sup>5</sup>	R <sub>5</sub> <sup>4</sup>	R <sub>5</sub> <sup>3</sup>	G <sub>5</sub> <sup>7</sup>	G <sub>5</sub> <sup>6</sup>	G <sub>5</sub> <sup>5</sup>	G <sub>5</sub> <sup>4</sup>	G <sub>5</sub> <sup>3</sup>	G <sub>5</sub> <sup>2</sup>	B <sub>5</sub> <sup>7</sup>	B <sub>5</sub> <sup>6</sup>	B <sub>5</sub> <sup>5</sup>	B <sub>5</sub> <sup>4</sup>	B <sub>5</sub> <sup>3</sup>

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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1); //set LCD display layer. Reference Page.5~6
MPU16_16bpp_Memory_Write (0,0,128,128,pic1616);
//pic1616 is 16bit array data , 16bpp color depth ,size 128x128 picture data.
```

The example display on LCD(Layer1):

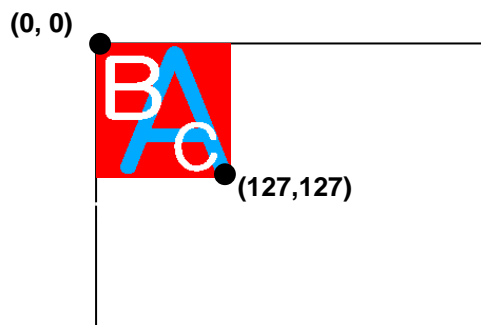


Figure 1.4 : Using 16 bits MCU interface and 16bpp color depth. MCU write data into SDRAM(Layer1).

**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 <sub>0</sub> <sup>7</sup>	G <sub>0</sub> <sup>6</sup>	G <sub>0</sub> <sup>5</sup>	G <sub>0</sub> <sup>4</sup>	G <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>2</sup>	G <sub>0</sub> <sup>1</sup>	G <sub>0</sub> <sup>0</sup>	B <sub>0</sub> <sup>7</sup>	B <sub>0</sub> <sup>6</sup>	B <sub>0</sub> <sup>5</sup>	B <sub>0</sub> <sup>4</sup>	B <sub>0</sub> <sup>3</sup>	B <sub>0</sub> <sup>2</sup>	B <sub>0</sub> <sup>1</sup>	B <sub>0</sub> <sup>0</sup>
2	B <sub>1</sub> <sup>7</sup>	B <sub>1</sub> <sup>6</sup>	B <sub>1</sub> <sup>5</sup>	B <sub>1</sub> <sup>4</sup>	B <sub>1</sub> <sup>3</sup>	B <sub>1</sub> <sup>2</sup>	B <sub>1</sub> <sup>1</sup>	B <sub>1</sub> <sup>0</sup>	R <sub>0</sub> <sup>7</sup>	R <sub>0</sub> <sup>6</sup>	R <sub>0</sub> <sup>5</sup>	R <sub>0</sub> <sup>4</sup>	R <sub>0</sub> <sup>3</sup>	R <sub>0</sub> <sup>2</sup>	R <sub>0</sub> <sup>1</sup>	R <sub>0</sub> <sup>0</sup>
3	R <sub>1</sub> <sup>7</sup>	R <sub>1</sub> <sup>6</sup>	R <sub>1</sub> <sup>5</sup>	R <sub>1</sub> <sup>4</sup>	R <sub>1</sub> <sup>3</sup>	R <sub>1</sub> <sup>2</sup>	R <sub>1</sub> <sup>1</sup>	R <sub>1</sub> <sup>0</sup>	G <sub>1</sub> <sup>7</sup>	G <sub>1</sub> <sup>6</sup>	G <sub>1</sub> <sup>5</sup>	G <sub>1</sub> <sup>4</sup>	G <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>2</sup>	G <sub>1</sub> <sup>1</sup>	G <sub>1</sub> <sup>0</sup>
4	G <sub>2</sub> <sup>7</sup>	G <sub>2</sub> <sup>6</sup>	G <sub>2</sub> <sup>5</sup>	G <sub>2</sub> <sup>4</sup>	G <sub>2</sub> <sup>3</sup>	G <sub>2</sub> <sup>2</sup>	G <sub>2</sub> <sup>1</sup>	G <sub>2</sub> <sup>0</sup>	B <sub>2</sub> <sup>7</sup>	B <sub>2</sub> <sup>6</sup>	B <sub>2</sub> <sup>5</sup>	B <sub>2</sub> <sup>4</sup>	B <sub>2</sub> <sup>3</sup>	B <sub>2</sub> <sup>2</sup>	B <sub>2</sub> <sup>1</sup>	B <sub>2</sub> <sup>0</sup>
5	B <sub>3</sub> <sup>7</sup>	B <sub>3</sub> <sup>6</sup>	B <sub>3</sub> <sup>5</sup>	B <sub>3</sub> <sup>4</sup>	B <sub>3</sub> <sup>3</sup>	B <sub>3</sub> <sup>2</sup>	B <sub>3</sub> <sup>1</sup>	B <sub>3</sub> <sup>0</sup>	R <sub>2</sub> <sup>7</sup>	R <sub>2</sub> <sup>6</sup>	R <sub>2</sub> <sup>5</sup>	R <sub>2</sub> <sup>4</sup>	R <sub>2</sub> <sup>3</sup>	R <sub>2</sub> <sup>2</sup>	R <sub>2</sub> <sup>1</sup>	R <sub>2</sub> <sup>0</sup>
6	R <sub>3</sub> <sup>7</sup>	R <sub>3</sub> <sup>6</sup>	R <sub>3</sub> <sup>5</sup>	R <sub>3</sub> <sup>4</sup>	R <sub>3</sub> <sup>3</sup>	R <sub>3</sub> <sup>2</sup>	R <sub>3</sub> <sup>1</sup>	R <sub>3</sub> <sup>0</sup>	G <sub>3</sub> <sup>7</sup>	G <sub>3</sub> <sup>6</sup>	G <sub>3</sub> <sup>5</sup>	G <sub>3</sub> <sup>4</sup>	G <sub>3</sub> <sup>3</sup>	G <sub>3</sub> <sup>2</sup>	G <sub>3</sub> <sup>1</sup>	G <sub>3</sub> <sup>0</sup>

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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
MPU16_24bpp_Mode1_Memory_Write(0,0,128,128,pic16241);
//pic16241 is 16bit array data , 24bpp color deepth mode 1,size 128x128 picture data.
```

The example dispaly on LCD(Layer1):

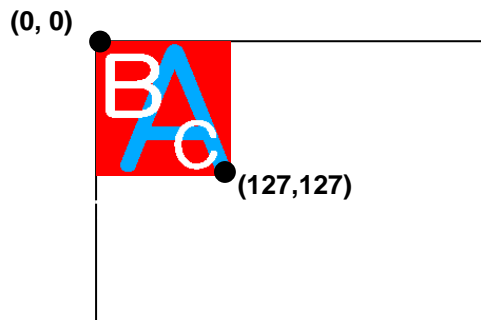


Figure 1.5 : Using 16 bits MCU interface and 24bpp color depth mode 1. MCU write data into SDRAM(Layer1).

**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 <sub>0</sub> <sup>7</sup>	G <sub>0</sub> <sup>6</sup>	G <sub>0</sub> <sup>5</sup>	G <sub>0</sub> <sup>4</sup>	G <sub>0</sub> <sup>3</sup>	G <sub>0</sub> <sup>2</sup>	G <sub>0</sub> <sup>1</sup>	G <sub>0</sub> <sup>0</sup>	B <sub>0</sub> <sup>7</sup>	B <sub>0</sub> <sup>6</sup>	B <sub>0</sub> <sup>5</sup>	B <sub>0</sub> <sup>4</sup>	B <sub>0</sub> <sup>3</sup>	B <sub>0</sub> <sup>2</sup>	B <sub>0</sub> <sup>1</sup>	B <sub>0</sub> <sup>0</sup>
2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	R <sub>0</sub> <sup>7</sup>	R <sub>0</sub> <sup>6</sup>	R <sub>0</sub> <sup>5</sup>	R <sub>0</sub> <sup>4</sup>	R <sub>0</sub> <sup>3</sup>	R <sub>0</sub> <sup>2</sup>	R <sub>0</sub> <sup>1</sup>	R <sub>0</sub> <sup>0</sup>
3	G <sub>1</sub> <sup>7</sup>	G <sub>1</sub> <sup>6</sup>	G <sub>1</sub> <sup>5</sup>	G <sub>1</sub> <sup>4</sup>	G <sub>1</sub> <sup>3</sup>	G <sub>1</sub> <sup>2</sup>	G <sub>1</sub> <sup>1</sup>	G <sub>1</sub> <sup>0</sup>	B <sub>1</sub> <sup>7</sup>	B <sub>1</sub> <sup>6</sup>	B <sub>1</sub> <sup>5</sup>	B <sub>1</sub> <sup>4</sup>	B <sub>1</sub> <sup>3</sup>	B <sub>1</sub> <sup>2</sup>	B <sub>1</sub> <sup>1</sup>	B <sub>1</sub> <sup>0</sup>
4	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	R <sub>1</sub> <sup>7</sup>	R <sub>1</sub> <sup>6</sup>	R <sub>1</sub> <sup>5</sup>	R <sub>1</sub> <sup>4</sup>	R <sub>1</sub> <sup>3</sup>	R <sub>1</sub> <sup>2</sup>	R <sub>1</sub> <sup>1</sup>	R <sub>1</sub> <sup>0</sup>
5	G <sub>2</sub> <sup>7</sup>	G <sub>2</sub> <sup>6</sup>	G <sub>2</sub> <sup>5</sup>	G <sub>2</sub> <sup>4</sup>	G <sub>2</sub> <sup>3</sup>	G <sub>2</sub> <sup>2</sup>	G <sub>2</sub> <sup>1</sup>	G <sub>2</sub> <sup>0</sup>	B <sub>2</sub> <sup>7</sup>	B <sub>2</sub> <sup>6</sup>	B <sub>2</sub> <sup>5</sup>	B <sub>2</sub> <sup>4</sup>	B <sub>2</sub> <sup>3</sup>	B <sub>2</sub> <sup>2</sup>	B <sub>2</sub> <sup>1</sup>	B <sub>2</sub> <sup>0</sup>
6	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	R <sub>2</sub> <sup>7</sup>	R <sub>2</sub> <sup>6</sup>	R <sub>2</sub> <sup>5</sup>	R <sub>2</sub> <sup>4</sup>	R <sub>2</sub> <sup>3</sup>	R <sub>2</sub> <sup>2</sup>	R <sub>2</sub> <sup>1</sup>	R <sub>2</sub> <sup>0</sup>

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

API :

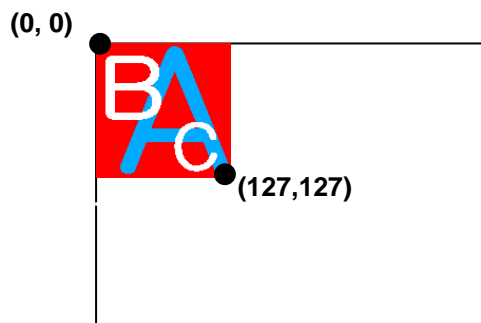
Using 16 bits MCU interface and 24bpp color depth mode 2. MCU write data into SDRAM.

```
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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1); //set LCD display layer. Reference Page.5~6
MPU16_24bpp_Mode2_Memory_Write(0,0,128,128,pic1624);
//pic1624 is 16bit array data , 24bpp color depth mode 2,size 128x128 picture data.
```

The example display on LCD(Layer1):



**Figure 1.6 : 16 bits MCU and 24bpp color depth mode2 MCU Write data to SDRAM(Layer1)**

## 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. RA8871M 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 RA8871M is.

### 2.1: Ellipse/Circle Input

RA8871M 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, RA8871M 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:

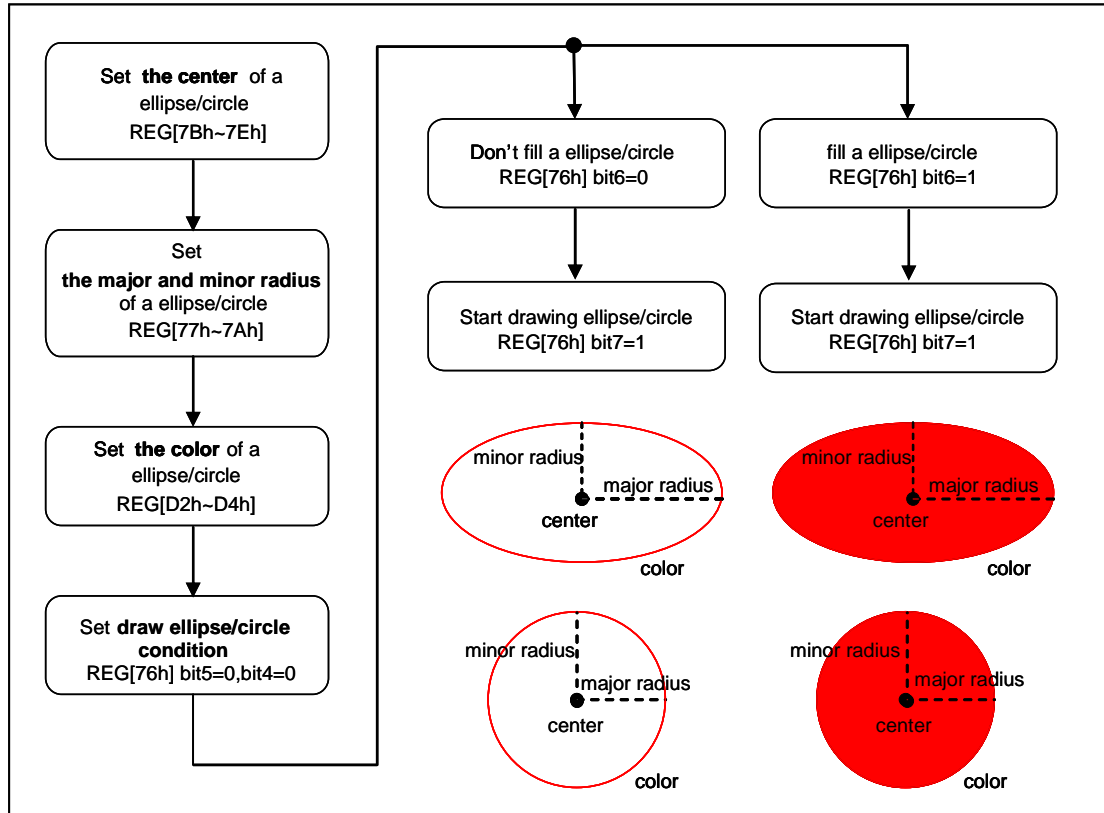


Figure 2.1

## 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:**

```

Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//When color depth = 8bpp
Draw_Circle(0xfc,100,50,50);
Draw_Circle_Fill(0xfc,250,50,50);
Or
//When color depth = 16bpp
Draw_Circle(0xfe0,100,50,50);
Draw_Circle_Fill(0xfe0,250,50,50);
Or
//When color depth = 24bpp
Draw_Circle(0xffff00,100,50,50);
Draw_Circle_Fill(0xffff00,250,50,50);
    
```

LCD display screen(Layer1):

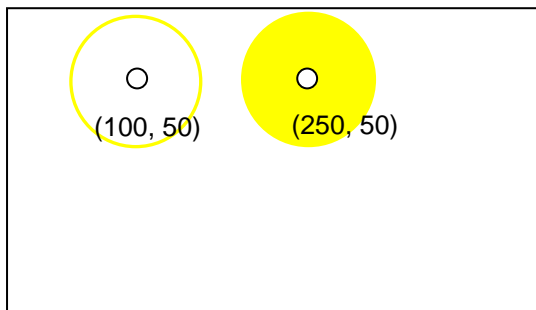


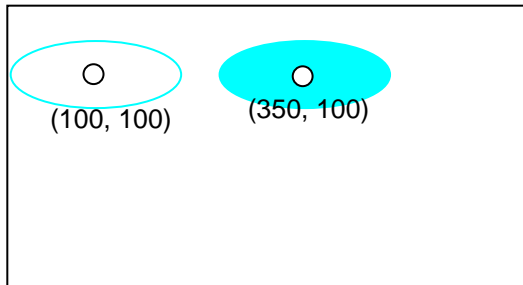
Figure 2.2 : Draw a Yellow Circle ,Center(100,50), R = 50,  
and draw a Yellow Circle Fill, Center(250,50), R = 50.

**Example 2:**

```

Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//When color depth = 8bpp
Draw_Ellipse(0x1f,100,100,100,50);
Draw_Ellipse_Fill(0x1f,350,100,100,50);
Or
//When color depth = 16bpp
Draw_Ellipse(0x07ff,100,100,100,50);
Draw_Ellipse_Fill(0x07ff,350,100,100,50);
Or
//When color depth = 24bpp
Draw_Ellipse(0x00ffff,100,100,100,50);
Draw_Ellipse_Fill(0x00ffff,350,100,100,50);
    
```

LCD display screen(Layer1):



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



**2.2: Curve Input**

RA8871M 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, RA8871M 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:

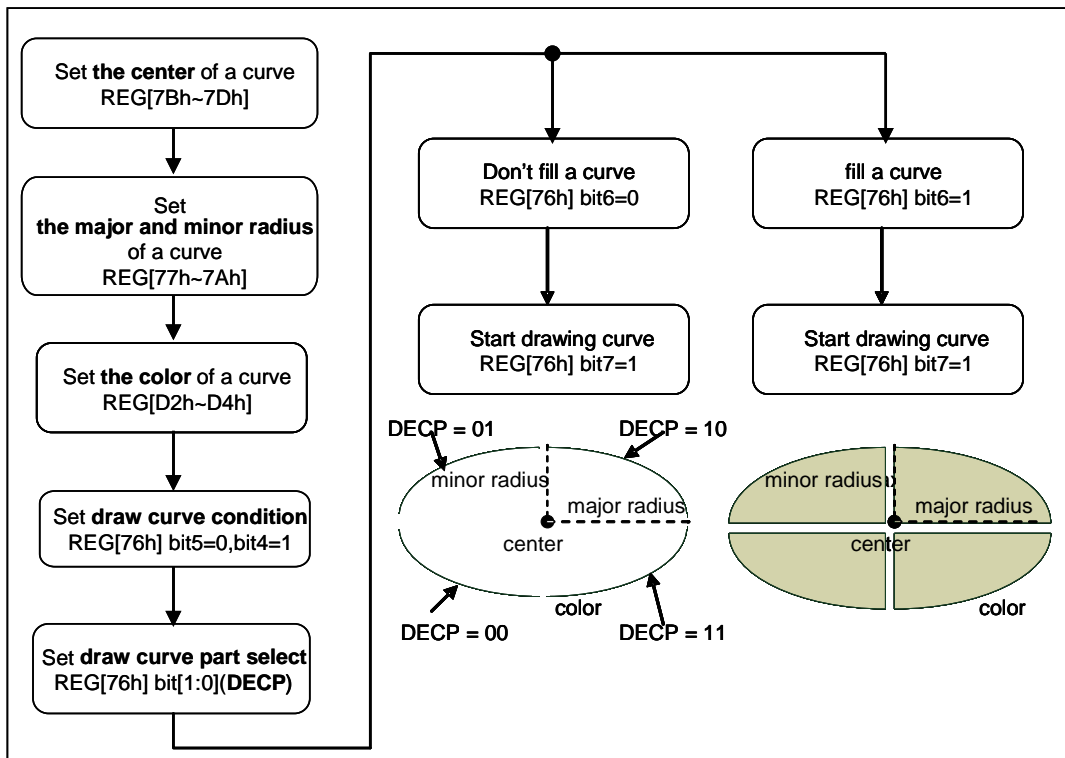


Figure 2-4

**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  
)
```

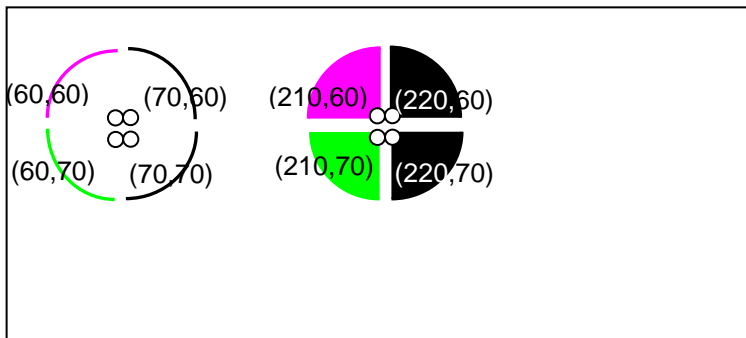
Example :

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1); //set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//When color deepth = 8bpp
Draw_Left_Up_Curve(0xe3,60,60,50,50);
Draw_Right_Down_Curve(0x00,70,70,50,50);
Draw_Right_Up_Curve(0x00,70,60,50,50);
Draw_Left_Down_Curve(0x1c,60,70,50,50);
Draw_Left_Up_Curve_Fill(0xe3,210,60,50,50);
Draw_Right_Down_Curve_Fill(0x00,220,70,50,50);
Draw_Right_Up_Curve_Fill(0x00,220,60,50,50);
Draw_Left_Down_Curve_Fill(0x1c,210,70,50,50);
Or
//When color deepth = 16bpp
Draw_Left_Up_Curve(0xf11f,60,60,50,50);
Draw_Right_Down_Curve(0x0000,70,70,50,50);
Draw_Right_Up_Curve(0x0000,70,60,50,50);
Draw_Left_Down_Curve(0x07e0,60,70,50,50);
Draw_Left_Up_Curve_Fill(0xf11f,210,60,50,50);
Draw_Right_Down_Curve_Fill(0x0000,220,70,50,50);
Draw_Right_Up_Curve_Fill(0x0000,220,60,50,50);
Draw_Left_Down_Curve_Fill(0x07e0,210,70,50,50);
Or
//When color deepth = 24bpp
Draw_Left_Up_Curve(0xff00ff,60,60,50,50);
Draw_Right_Down_Curve(0x000000,70,70,50,50);
Draw_Right_Up_Curve(0x000000,70,60,50,50);
Draw_Left_Down_Curve(0x00ff00,60,70,50,50);
Draw_Left_Up_Curve_Fill(0xff00ff,210,60,50,50);
Draw_Right_Down_Curve_Fill(0x000000,220,70,50,50);
Draw_Right_Up_Curve_Fill(0x000000,220,60,50,50);
Draw_Left_Down_Curve_Fill(0x00ff00,210,70,50,50);
```

**8 step for this program:**

1. Draw a pink upper left curve,Center(60,60), X\_R = 50,Y\_R=50
2. Draw a white lower right curve,Center(70,70), X\_R = 50,Y\_R=50
3. Draw a white upper right curve,Center(70,60), X\_R = 50,Y\_R=50
4. Draw a green lower left curve,Center(60,70), X\_R = 50,Y\_R=50
5. Draw a pink upper left curve fill,Center(210,60), X\_R = 50,Y\_R=50
6. Draw a white lower right curve fill,Center(220,70), X\_R = 50,Y\_R=50
7. Draw a white upper right curve fill,Center(220,60), X\_R = 50,Y\_R=50
8. Draw a green lower left curve fill,Center(210,70), X\_R = 50,Y\_R=50

**LCD display screen(Layer1):**



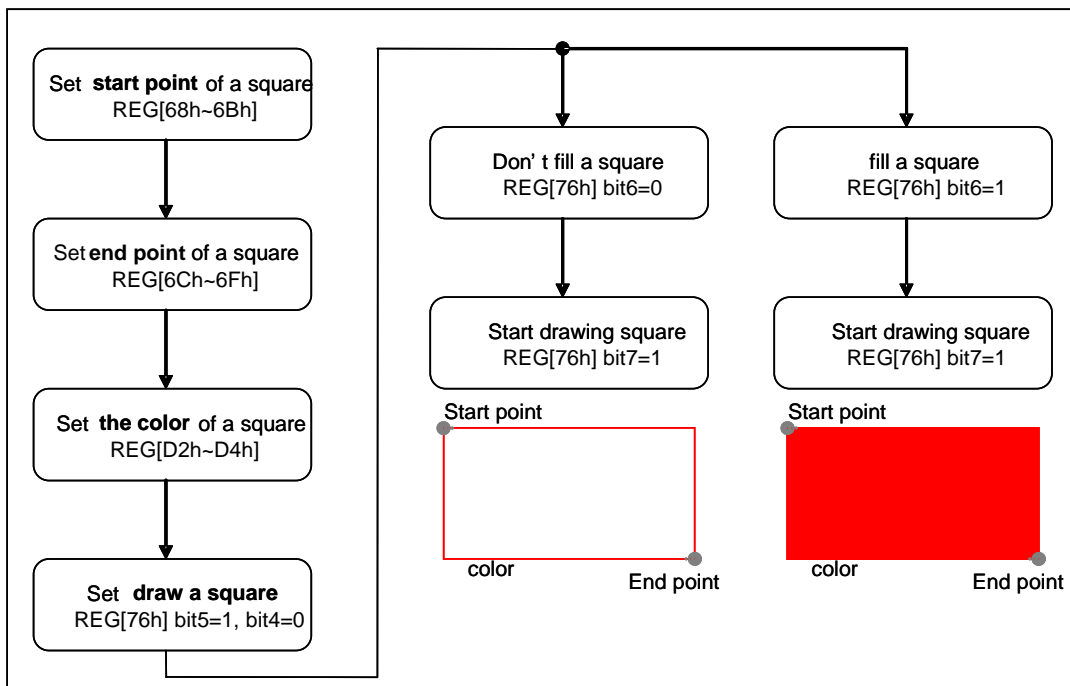
**Figure 2.5 : Draw curve example.**

**2.3: Square Input**

RA8871M 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, RA8871M 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:**

```

Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1); //set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//when color depth = 8bpp
Draw_Square(0xe0,50,50,150,150);
Draw_Square_Fill(0xe0,200,50,300,150);
Or
//When color deepth = 16bpp
Draw_Square(0xf800,50,50,150,150);
Draw_Square_Fill(0xf800,200,50,300,150);
Or
//When color deepth = 24bpp

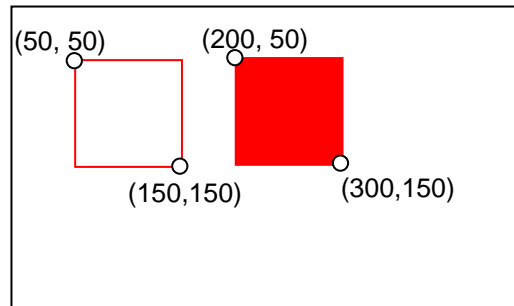
```



```
Draw_Square(0xff0000,50,50,150,150);
```

```
Draw_Square_Fill(0xff0000,200,50,300,150);
```

LCD display screen(Layer1):



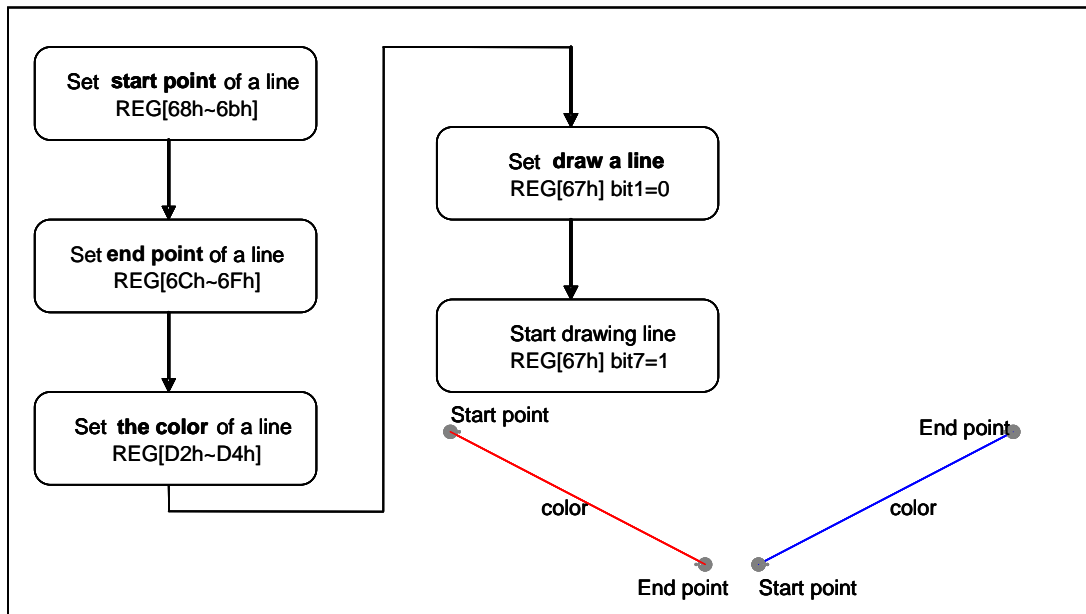
**Figure 2.7 : Draw a red Square from Point1(50,50) to Point2(150,150)  
and Draw a red Square Fill from Point1(200,50) to Point2(300,150)**

**2.4:Line Input**

RA8871M 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, RA8871M 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 :

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//When color depth = 8bpp
Draw_Line(0xe0,10,10,200,200);
Or
//When color depth = 16bpp
Draw_Line(0xf800,10,10,200,200);
Or
//When color depth = 24bpp
Draw_Line(0xff0000,10,10,200,200);
```

### LCD display screen:

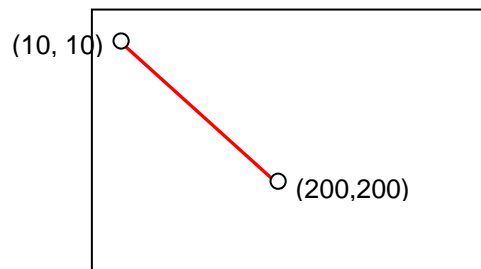


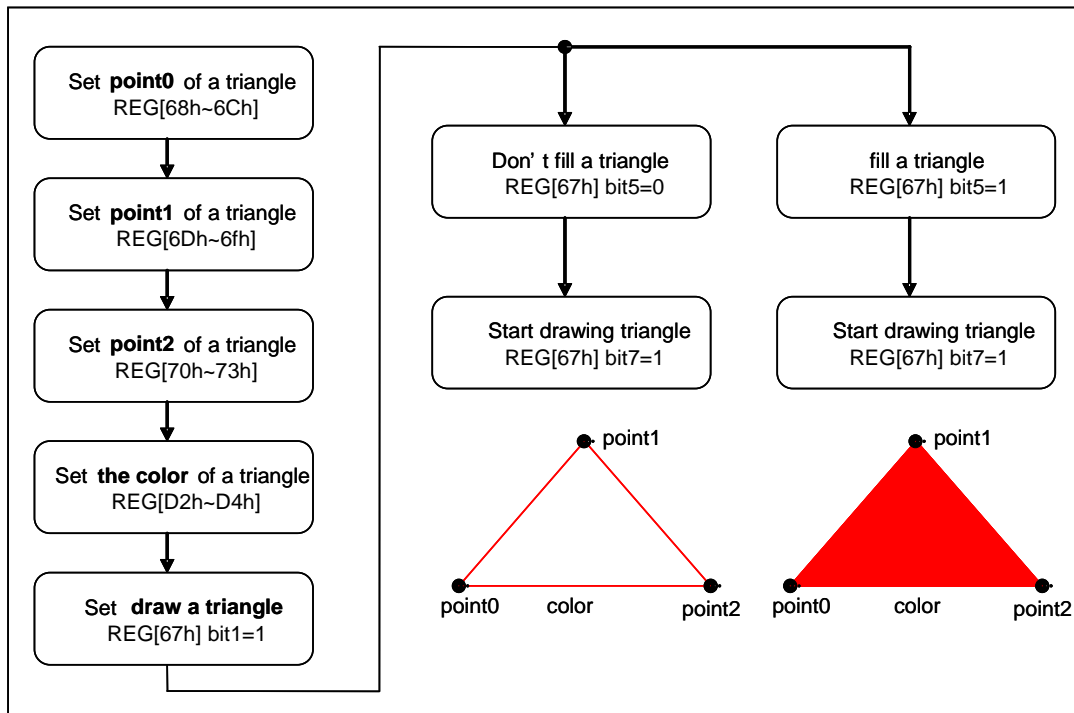
Figure 2.9 : Draw a red Line from Point1(10,10) to Point2(200,200).

**2.5: Triangle Input**

RA8871M 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~6Ch], 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, RA8871M 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:



**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:**

```

Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//When color depth = 8bpp
Draw_Triangle(0x07,150,0,150,100,250,100);
Draw_Triangle_Fill(0x03,300,0,300,100,400,100);
//Or
//When color depth = 16bpp
Draw_Triangle(0x001f,150,0,150,100,250,100);

```

```
Draw_Triangle_Fill(0x001f,300,0,300,100,400,100);
```

```
//Or
```

```
//When color depth = 24bpp
```

```
Draw_Triangle(0x0000ff,150,0,150,100,250,100);
```

```
Draw_Triangle_Fill(0x0000ff,300,0,300,100,400,100);
```

LCD display screen(Layer1):

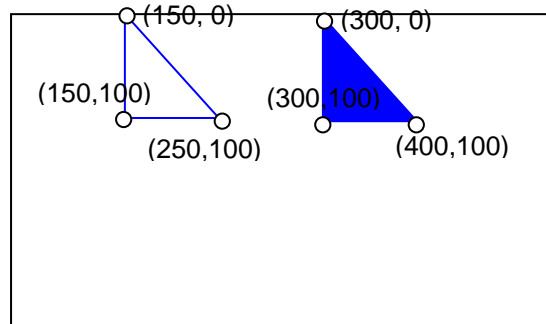


Figure 2.11 : Draw a blue Triangle by Point1(150,0) ,Point2(150,100) and Point3(250,100) and draw a blue Triangle Fill by Point1(300,0) ,Point2(300,100) and Point3(400,100)

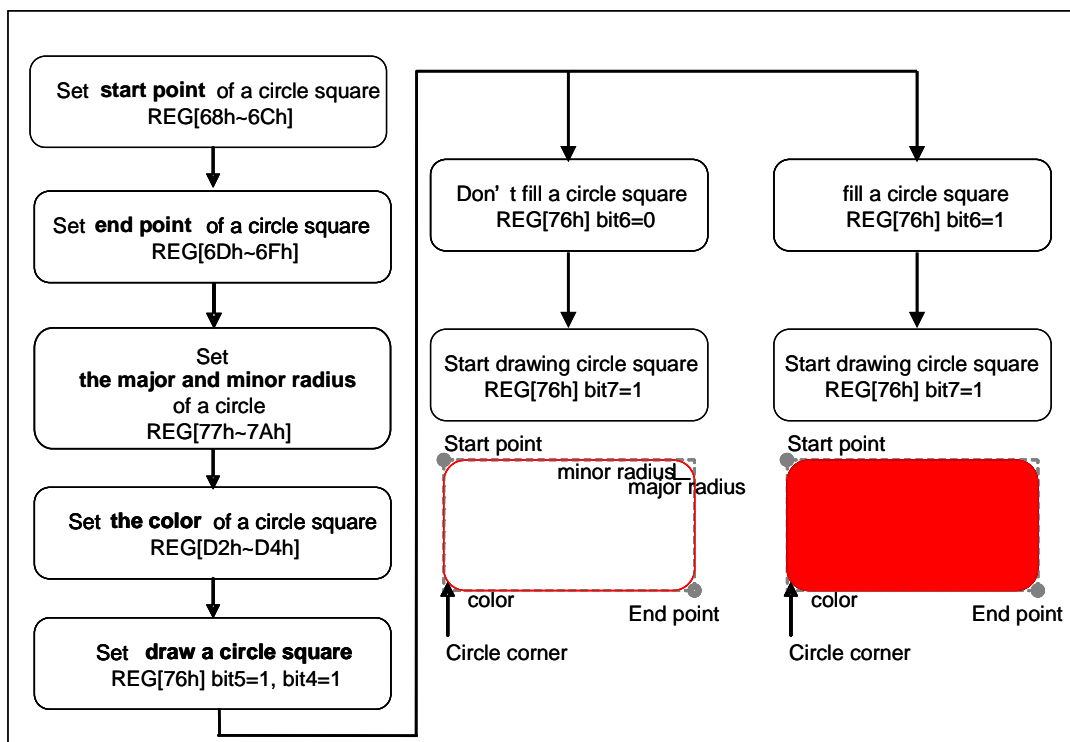
**2.6: Square Of Circle Corner Input**

RA8871M 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, RA8871M 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
)
```



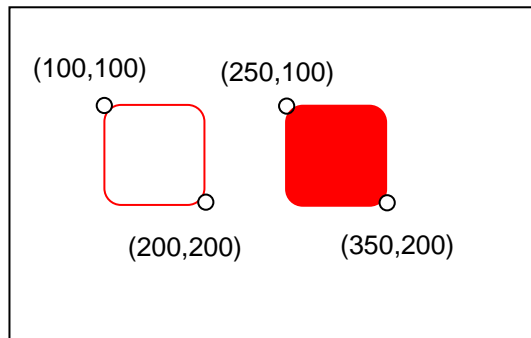
**Example 1:**

```

Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6
Active_Window_XY(0,0);
Active_Window_WH(Panel_width,Panel_length); //set full LCD size can draw graph
+
//When color depth = 8bpp
Draw_Circle_Square(0xe0, 100,100,200,200,20,30);
Draw_Circle_Square_Fill(0xe0, 250,100,350,200,20,30);
Or
//When color depth = 16bpp
Draw_Circle_Square(0xf800,100,100,200,200,20,30);
Draw_Circle_Square_Fill(0xf800,250,100,350,200,20,30);
Or
//When color depth = 24bpp
Draw_Circle_Square(0xff0000, 100,100,200,200,20,30);
Draw_Circle_Square_Fill(0xff0000,250,100,350,200,20,30);

```

**LCD display screen(Layer1):**



**Figure 2.13 :** Draw a red Square Of Circle Corner from Point(100,100) to Point(200,200) with X\_R = 20,Y\_R=30, and draw a red Square Of Circle Corner Fill from Point(250,100) to Point(350,200) with X\_R =20,Y\_R=30.

### Chapter 3 : 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 RA8871M. The process unit of DMA function is pixel. Regarding the flowchart of DMA function, please refer to the description as below.

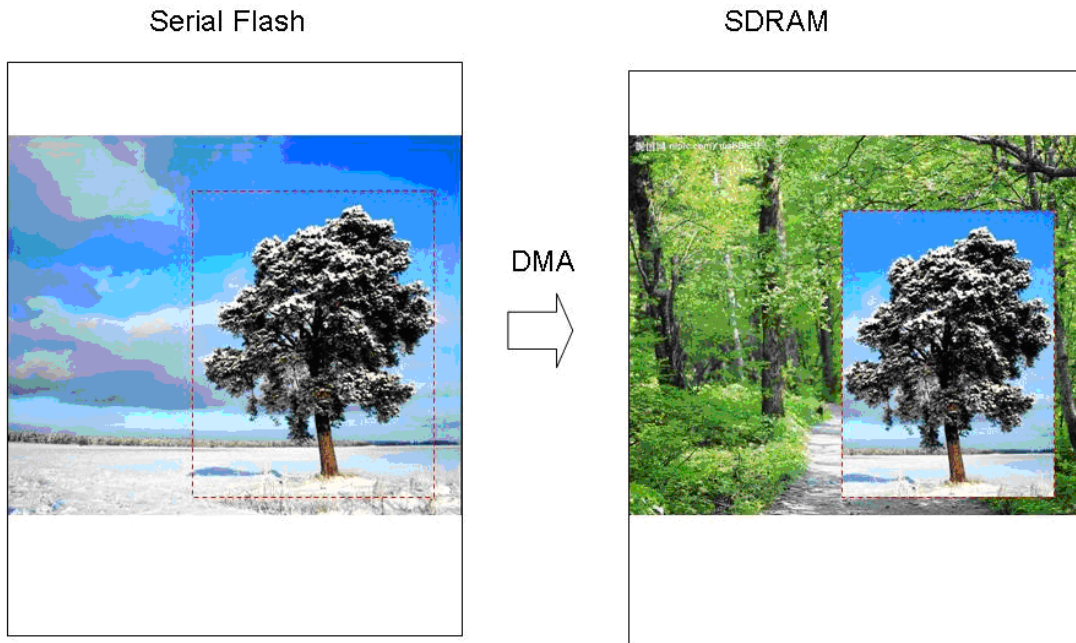


Figure 3.1 : DMA Function

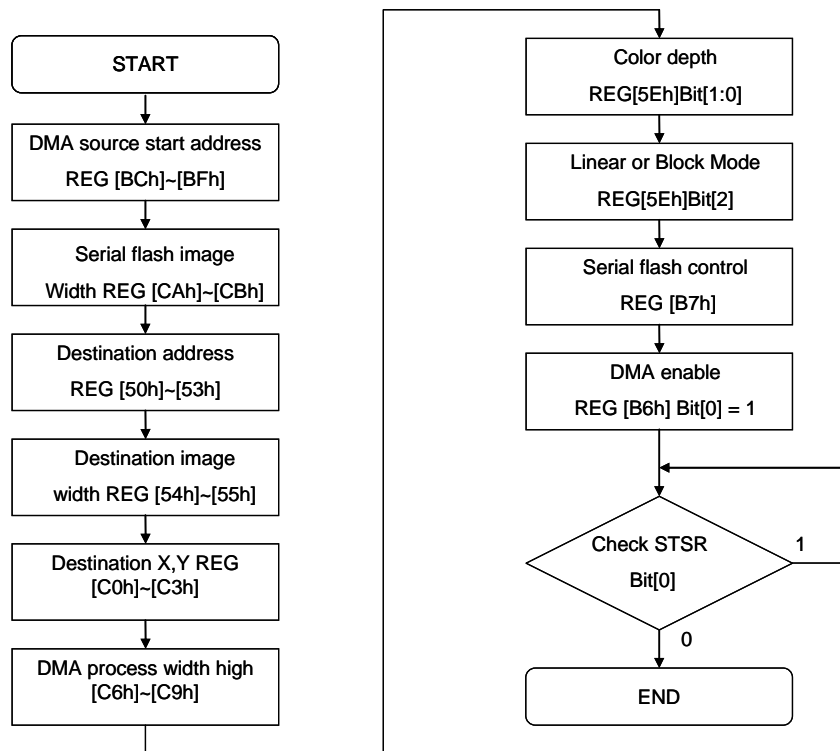
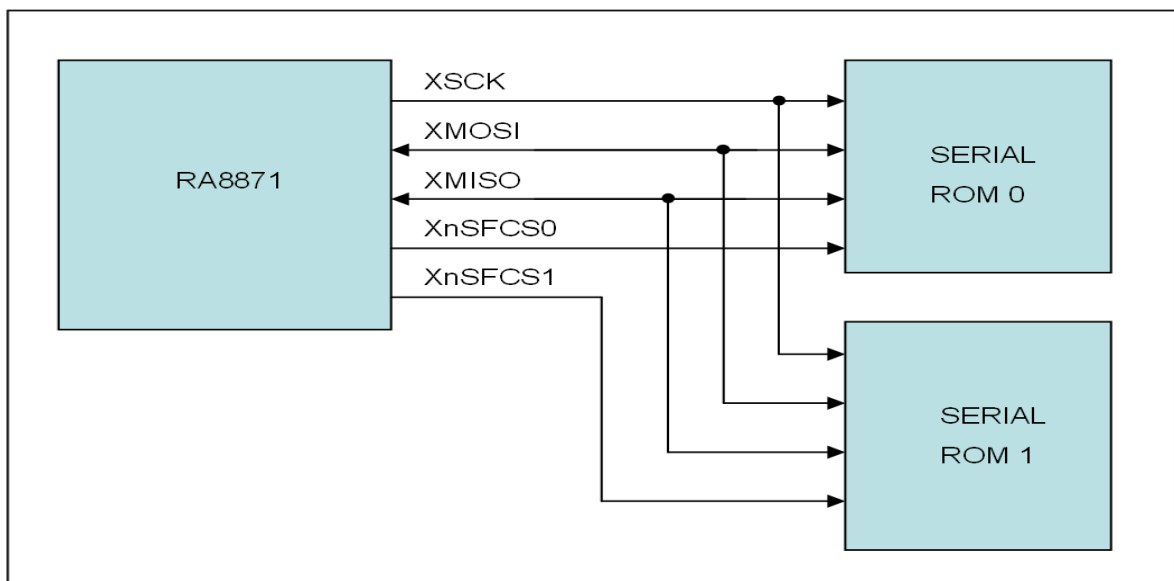


Figure 3.21 : Enable DMA Procedure – Check Flag

**3-2.Serial Flash Control Unit**

RA8871M 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.

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, RA8871M 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 3.3: RA8871M Serial Flash/ROM System**

### 3.3. API Example and Display results on LCD for the DMA Function:

We provider two API for user as below:

```

void DMA_24bit
(
    unsigned char SCS //SCS : 0 = Use SCS0, 1 = Use SCS1
    ,unsigned char Clk //Clk : SPI Clock = System Clock /{(Clk+1)*2}
    ,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 SCS //SCS : 0 = Use SCS0, 1 = Use SCS1
    ,unsigned char Clk //Clk : SPI Clock = System Clock /{(Clk+1)*2}
    ,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
)
    
```

Example :

`Write_Layer(1); //set memory read/write layer.Reference Page.5~6`

`Show_Layer(1);//set LCD display layer. Reference Page.5~6`

+

`(Flash = 128Mbit or under 128Mbit)`

`DMA_24bit(1,0,0,0,480,272,480,0);`

Or

`(Flash over 128Mbit)`

`open switch_24bits_to_32bits(1);`

`DMA_32bit(1,0,0,0,480,272,480,0);`

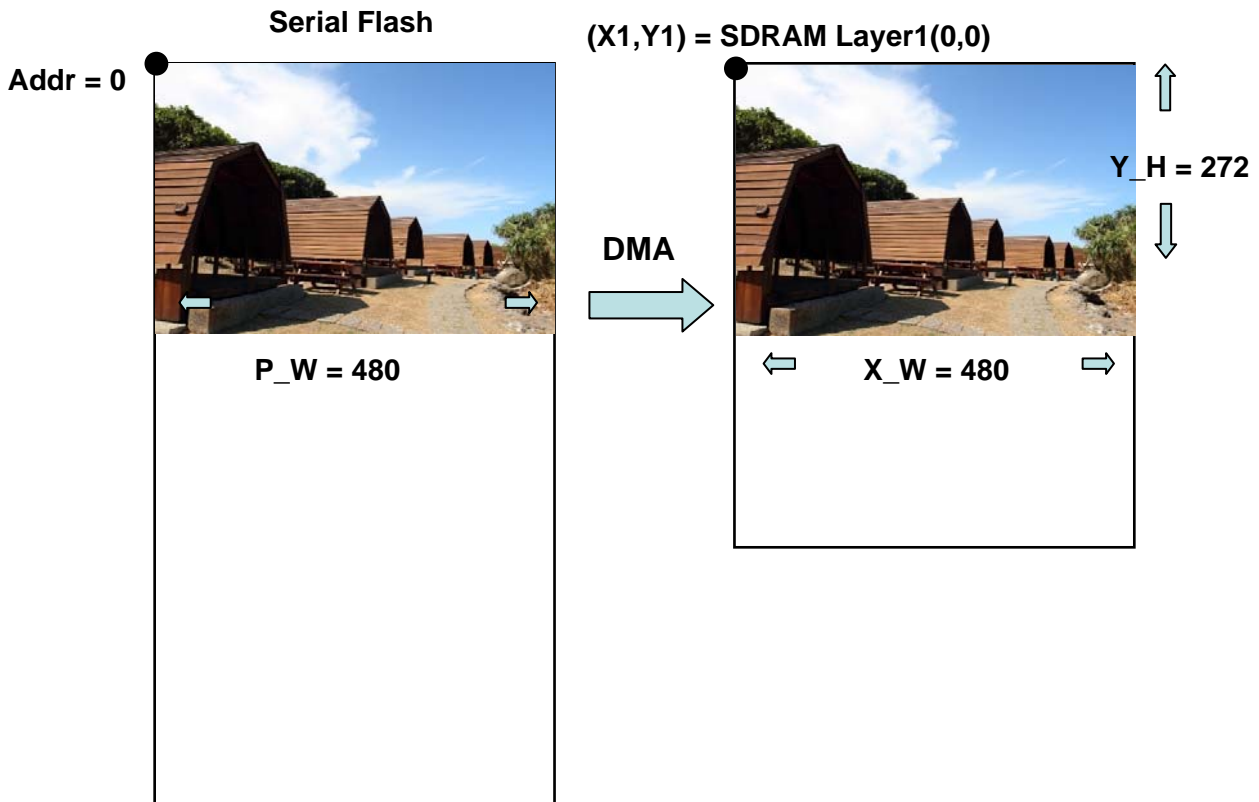
**Condition :**

**SCS = 1 : Use SCS1. Clk = 0 , SPI Clock = System Clock / {(0+1)\*2} = System Clock.**

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

**X\_W : DMA Block width = 480 . Y\_H : DMA Block height = 272.**

**P\_W : DMA Picture width =480 . Addr :DMA Source Start address = 0.**



**Figure 3.4: Using DMA Write data into SDRAM(Layer1) from Serial Flash**

## Chapter 4 : Block Transfer Engine

### Overview:

The RA8871M 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, RA8871M 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. RA8871M 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**

<b>BTE Operation REG[91h] Bits [3:0]</b>	<b>BTE Operation</b>
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	$\sim S0 \cdot \sim S1$ or $\sim ( S0+S1 )$
0010b	$\sim S0 \cdot S1$
0011b	$\sim S0$
0100b	$S0 \cdot \sim S1$
0101b	$\sim S1$
0110b	$S0 \wedge S1$
0111b	$\sim S0 + \sim S1$ or $\sim ( S0 \cdot S1 )$
1000b	$S0 \cdot S1$
1001b	$\sim ( S0 \wedge S1 )$
1010b	$S1$
1011b	$\sim S0 + S1$
1100b	$S0$
1101b	$S0 + \sim S1$
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 =  $\sim S0 \cdot S1$
- 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,

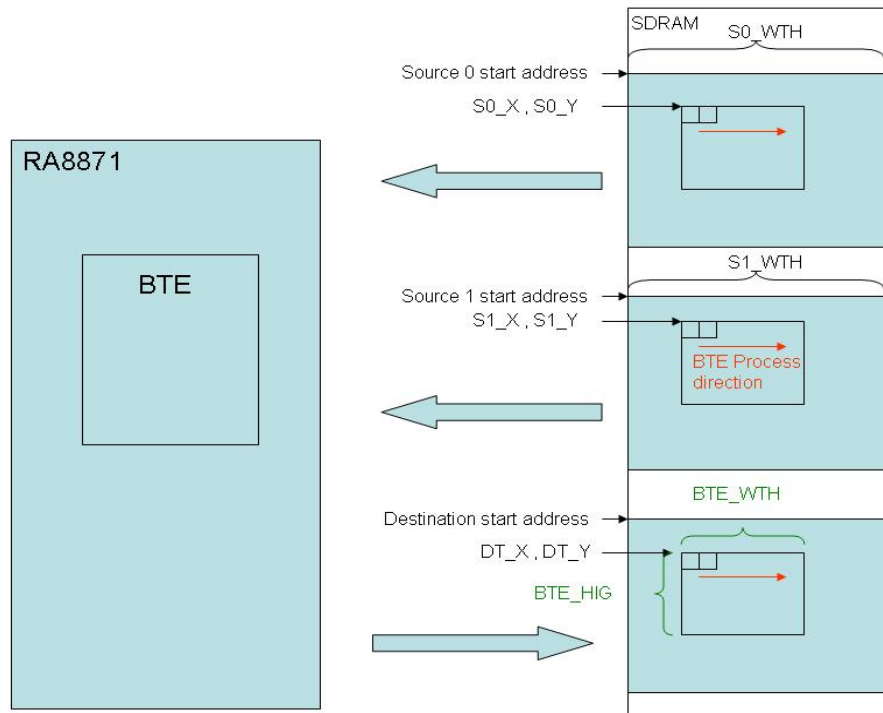
- Source 0 red color just only compare REG [D5h] Bit [7:5],
- Source 0 green color just only compare REG [D6h] Bit [7:5],
- Source 0 blue color just only compare REG [D7h] Bit [7:6]

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]

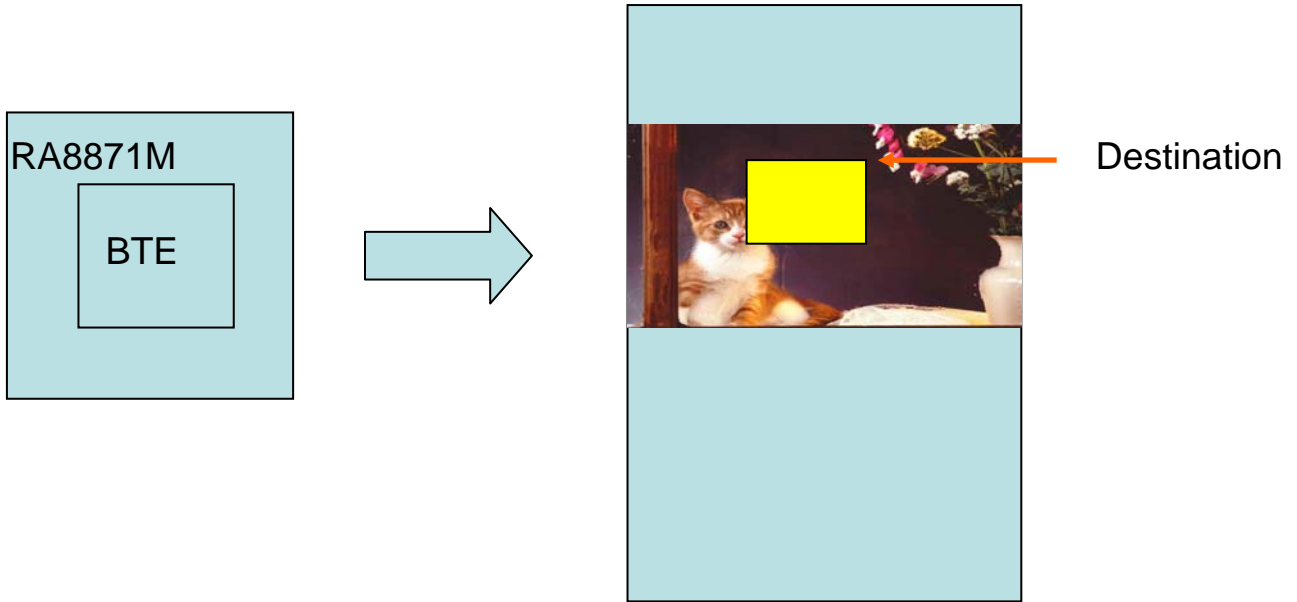


**Figure 4-1 : Memory Access of BTE Function**

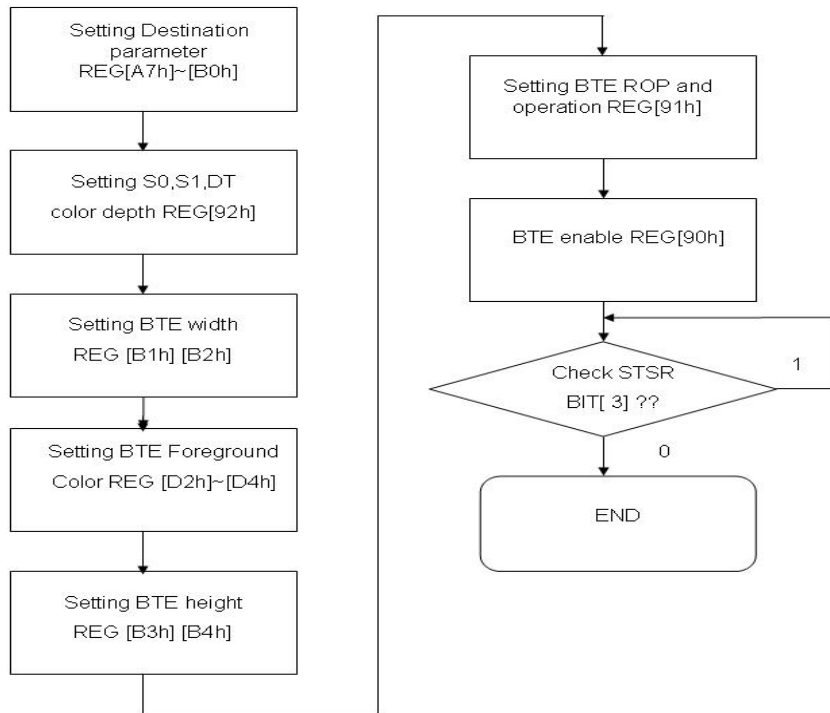


**4.1.1: Solid Fill**

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**



**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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xe0,200,200); //When color depth = 8bpp
```

or

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xf800,200,200); //When color depth = 16bpp
```

or

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xFF0000,200,200); //When color depth = 24bpp
```

Condition:

start address of destination = Layer1, ImageWidth = Panel\_width(define in userdef.h) , coordinate of destination = (0,0)  
 Foreground Color: 0xe0 (8bpp) (R3G3B2) 、 0xf800(16bpp)(R5G6B5) 、 0xFF0000(24bpp)(R8G8B8)  
 BTE Window Size = 200x200

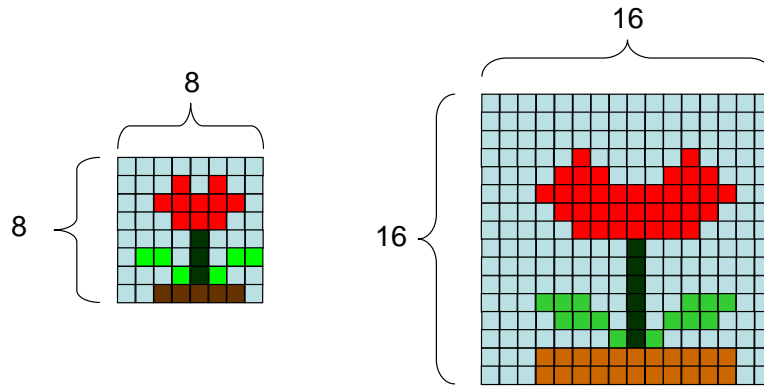
This program result(Layer1):



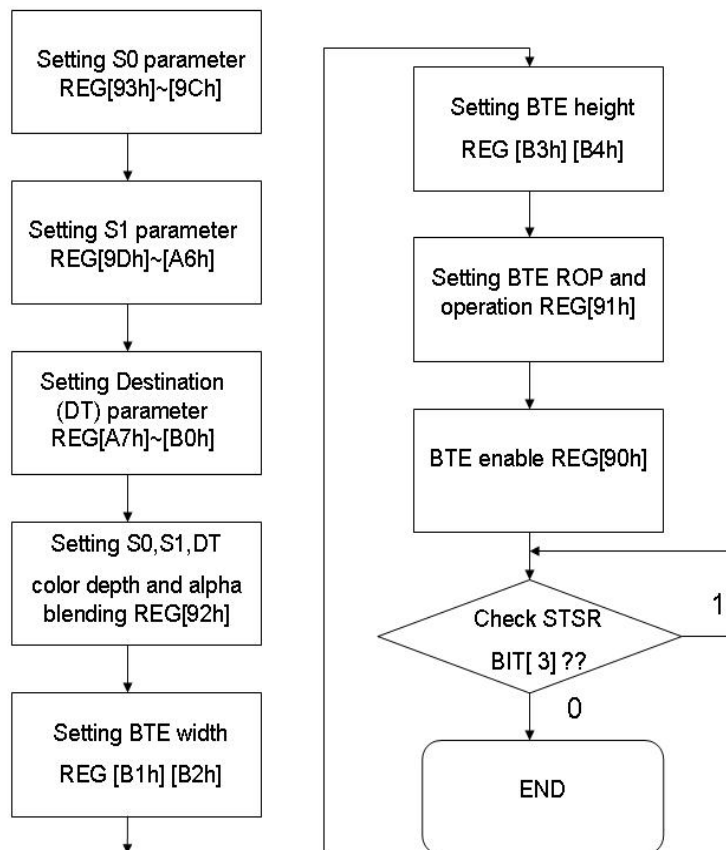
Figure 4-4 : Use Solid Fill to fill Layer1[(0, 0) ~ (199,199)] by red color

**4.2.1: Pattern Fill with ROP**

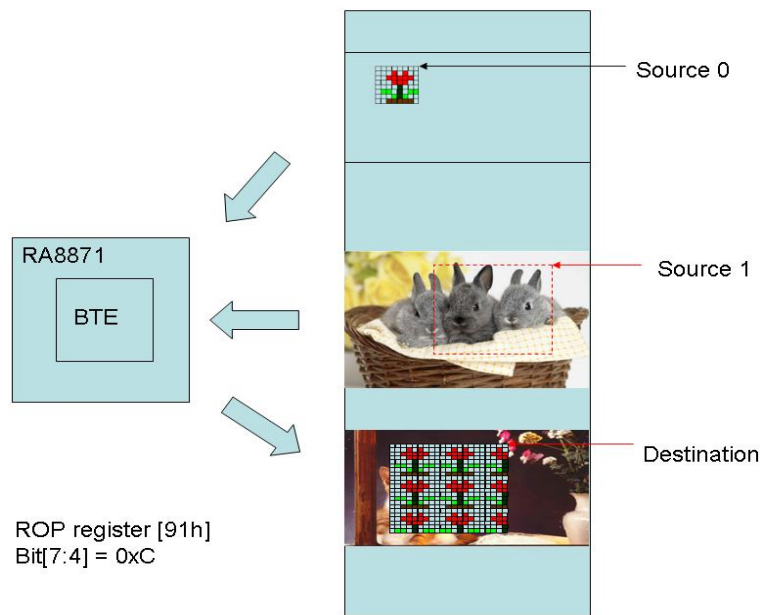
“Pattern Fill with ROP” operation fills a specified rectangular area of the SDRAM with a dedicated pattern repeatedly. The fill pattern is an array of 8x8/16x16 pixels stored in the SDRAM. The pattern can be logically combined with the destination using one of the 16 ROP codes. The operation can be used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



**Figure 4-5 : Pattern Format**

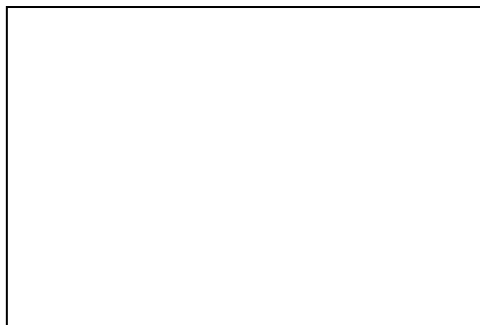


**Figure 4-6 : Flow Chart**



**Figure 4-72 : Hardware Data Flow**

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



**Figure 4-9 (16x16) Icon**

**Figure 4-8: SDRAM current data in this example**

**API for pattern fill with ROP:**

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

```

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

```

Example:

```

Write_Layer(1); //set memory read/write layer.Reference Page.5~6
Show_Layer(1);//set LCD display layer. Reference Page.5~6

+
MCU_8bit_ColorDepth_8bpp
DMA_24bit(1,0,0,0,480,272,480,6144000);
MPU8_8bpp_Memory_Write(0,0,16,16,lcon_8bit_8bpp);
or
MCU_8bit_ColorDepth_16bpp
DMA_24bit(1,0,0,0,480,272,480,4761600);
MPU8_16bpp_Memory_Write(0,0,16,16,lcon_8bit_16bpp);
or
MCU_8bit_ColorDepth_24bpp
DMA_24bit(1,0,0,0,480,272,480,2304000);
MPU8_24bpp_Memory_Write(0,0,16,16,lcon_8bit_24bpp);
or
MCU_16bit_ColorDepth_16bpp
DMA_24bit(1,0,0,0,480,272,480,4761600);
MPU16_16bpp_Memory_Write(0,0,16,16,lcon_16bit_16bpp);
or
MCU_16bit_ColorDepth_24bpp_Mode_1
DMA_24bit(1,0,0,0,480,272,480,2304000);
MPU16_24bpp_Mode1_Memory_Write(0,0,16,16,lcon_16bit_24bpp_mode1);
or
MCU_16bit_ColorDepth_24bpp_Mode_2
DMA_24bit(1,0,0,0,480,272,480,2304000);
MPU16_24bpp_Mode2_Memory_Write(0,0,16,16,lcon_16bit_24bpp_mode2);
+

```

```
BTE_Pattern_Fill(1,Layer1,Panel_width,0,0,Layer1,Panel_width,0,0,Layer1,Panel_width,200,100,12,100,100);
```

Condition:

**P\_8x8\_or\_16x16 = 1 , Pattern size = 16x16**

**Source 0 : Start Address = Layer1, Image Width = Panel\_width , coordinate (0,0)**

**Source 1 : Start Address = Layer1, Image Width = Panel\_width , coordinate (0,0)**

**Destination : Start Address = Layer1, Image Width = Panel\_width , coordinate (200,100)**

**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**

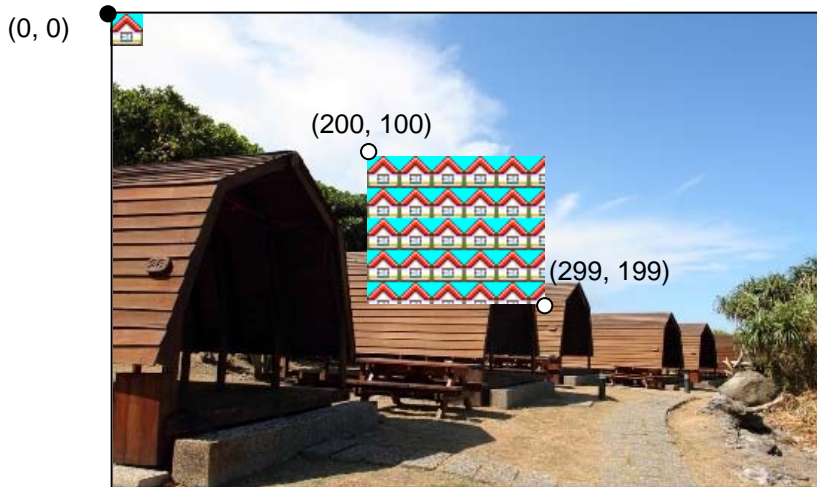


Figure 4-10: Using DMA function get a picture data from ext. Flash and write into SDRAM

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



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

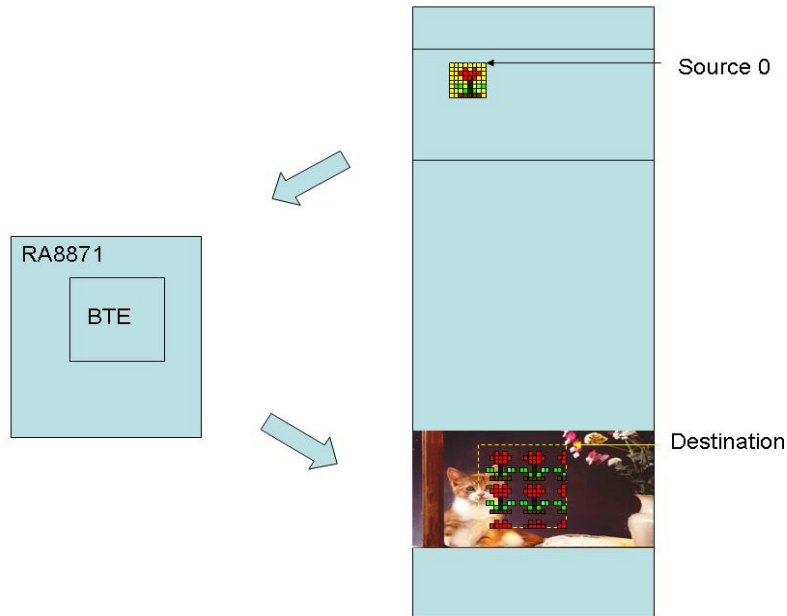
**Step 3 : Execute Pattern fill function**

**Figure 4-12: Accomplished pattern fill function, Source range is from Layer1(0, 0) to Layer1(15, 15), and Destination range is from Layer1(200,100) to Layer1(299,199)**

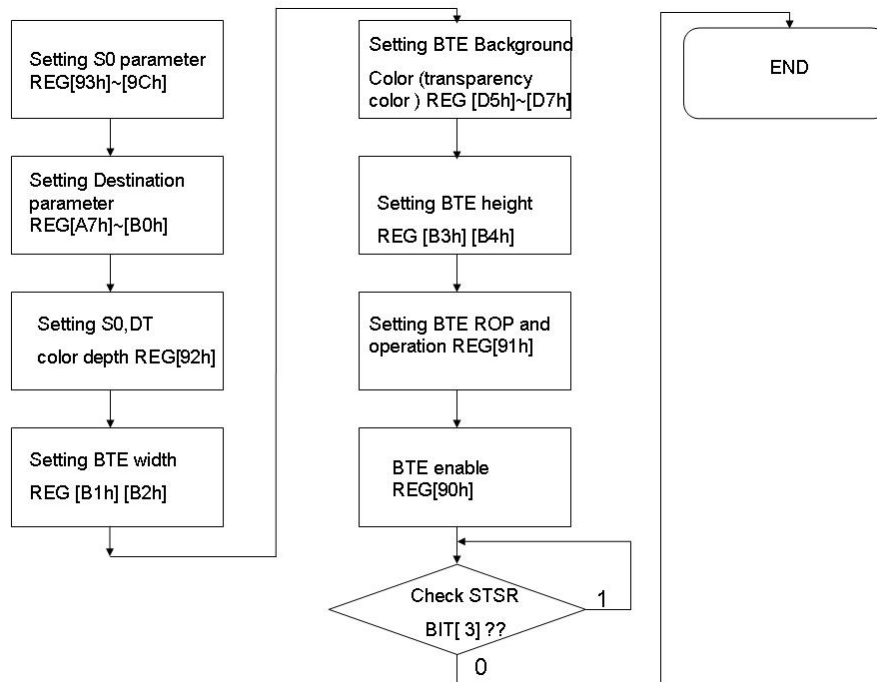


**4.2.3:Pattern Fill With Chroma Key**

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**



**Figure 4-11 : Flow Chart**

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



**Figure 4-13 : (16x16) Icon**

**Figure 4-12: SDRAM current data in this example**

**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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
    Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
//MCU_8bit_ColorDepth_8bpp
```

```
DMA_24bit(1,0,0,0,480,272,480,6144000);
```

```
MPU8_8bpp_Memory_Write(0,0,16,16,Icon_8bit_8bpp);
```

```
BTE_Pattern_Fill_With_Chroma_key(1,Layer1,Panel_width,0,0,Layer1,Panel_width,0,0,Layer1,
Panel_width,200,100,12,0x1f,100,100);
```

or

```
//MCU_8bit_ColorDepth_16bpp
```

```
DMA_24bit(1,0,0,0,480,272,480,4761600);
```

```
MPU8_16bpp_Memory_Write(0,0,16,16,Icon_8bit_16bpp);
```

```
BTE_Pattern_Fill_With_Chroma_key(1,Layer1,Panel_width,0,0,Layer1,Panel_width,0,0,Layer1,
Panel_width,200,100,12,0x07ff,100,100);
```

or

```
//MCU_8bit_ColorDepth_24bpp
```

```
DMA_24bit(1,0,0,0,480,272,480,2304000);
```

```
MPU8_24bpp_Memory_Write(0,0,16,16,Icon_8bit_24bpp);
```

```
BTE_Pattern_Fill_With_Chroma_key(1,Layer1,Panel_width,0,0,Layer1,Panel_width,0,0,Layer1,
Panel_width,200,100,12,0x00ffff,100,100);
```

or

```
//MCU_16bit_ColorDepth_16bpp
```

```
DMA_24bit(1,0,0,0,480,272,480,4761600);
```

```
MPU16_16bpp_Memory_Write(0,0,16,16,Icon_16bit_16bpp);
```

```
BTE_Pattern_Fill_With_Chroma_key(1,Layer1,Panel_width,0,0,Layer1,Panel_width,0,0,Layer1,
Panel_width,200,100,12,0x07ff,100,100);
```

or

**//MCU\_16bit\_ColorDepth\_24bpp\_Mode\_1**

**DMA\_24bit(1,0,0,0,480,272,480,2304000);**

**MPU16\_24bpp\_Mode1\_Memory\_Write(0,0,16,16,Icon\_16bit\_24bpp\_mode1);**

**BTE\_Pattern\_Fill\_With\_Chroma\_key(1,Layer1,Panel\_width,0,0,Layer1,Panel\_width,0,0,Layer1,Panel\_width,200,100,12,0x00ffff,100,100);**

or

**//MCU\_16bit\_ColorDepth\_24bpp\_Mode\_2**

**DMA\_24bit(1,0,0,0,480,272,480,2304000);**

**MPU16\_24bpp\_Mode2\_Memory\_Write(0,0,16,16,Icon\_16bit\_24bpp\_mode2);**

**BTE\_Pattern\_Fill\_With\_Chroma\_key(1,Layer1,Panel\_width,0,0,Layer1,Panel\_width,0,0,Layer1,Panel\_width,200,100,12,0x00ffff,100,100);**

Condition:

**P\_8x8\_or\_16x16 = 1 , Pattern size = 16x16**

**Source 0 : Start Address = Layer1, Image Width = Panel\_width,Coordinate = (0,0)**

**Source 1 : Start Address = Layer1, Image Width = Panel\_width, Coordinate = (0,0)**

**Destination : Start Address = Layer1, Image Width = Panel\_width, Coordinate = (200,100)**

**ROP\_Code = 12 : Destination data = Source 0 data. BTE Window Size = 100x100**

**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)**

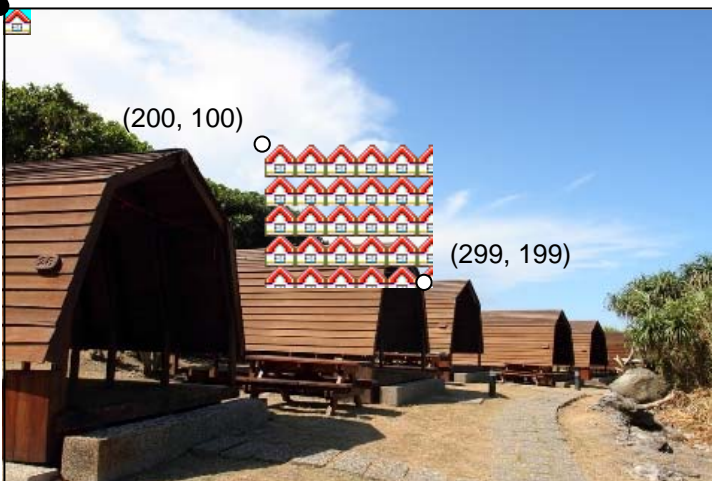
(0, 0)



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

**Step 3 : Execute Pattern fill with chroma key function**

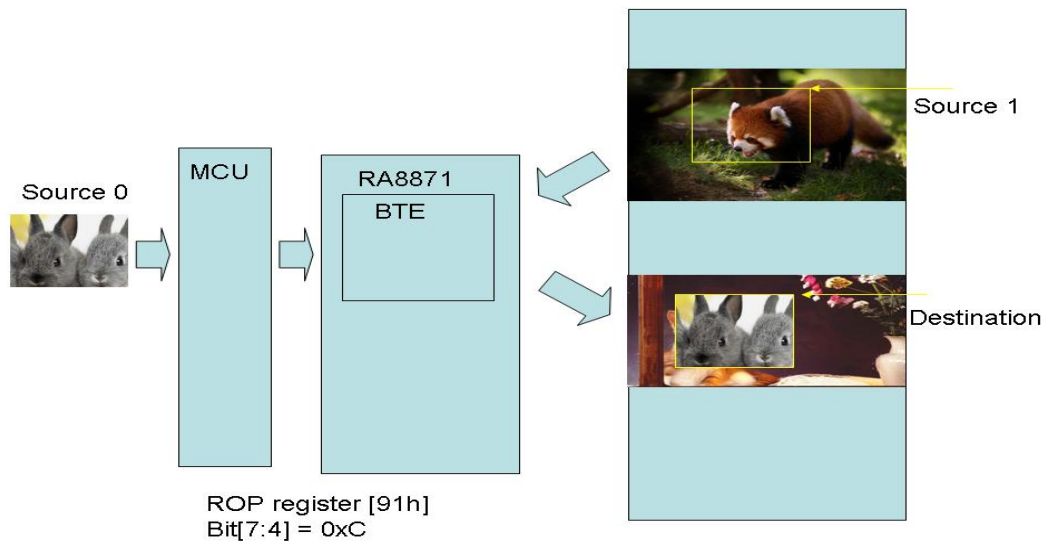
(0, b)



**Figure 4-16: Accomplished pattern fill with chroma key function, Source range is from Layer1(0, 0) to Layer1(15, 15), and Destination range is from Layer1(200,100) to Layer1(299,199)**

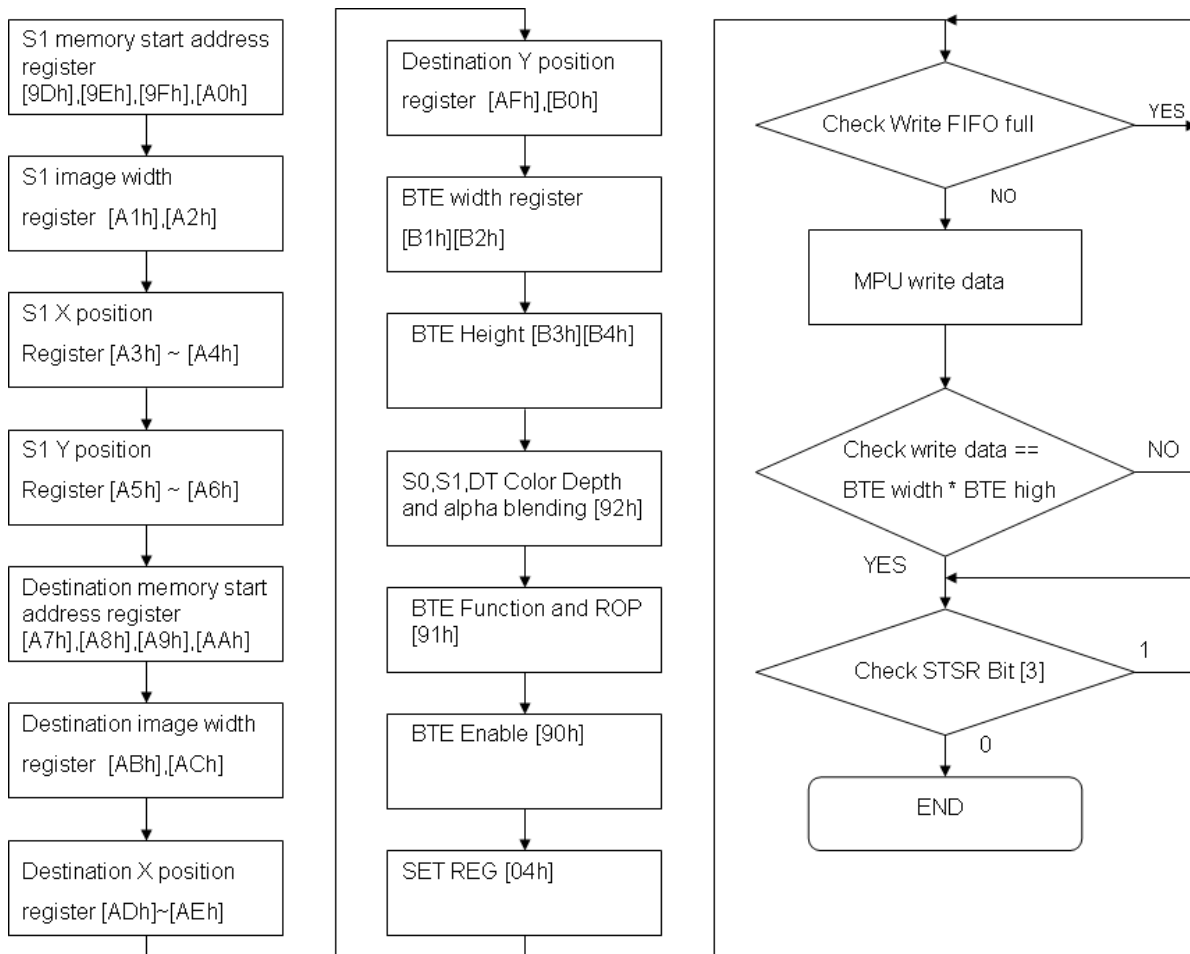
**4.3.1: MCU Write with ROP**

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.



**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:

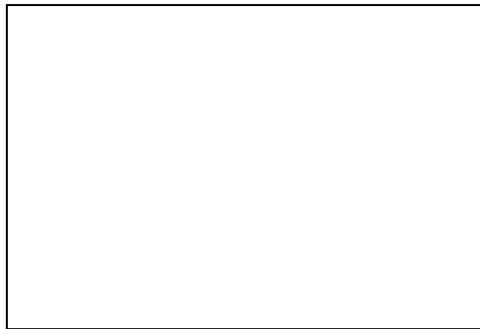


Figure 4-19: SDRAM current data in this example



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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
//Use 8bit MCU, 8bpp color depth
```

```
BTE_MCU_Write_MCU_8bit(Layer1,Panel_width,0,0,Layer1,Panel_width,100,100,12,128,128
,gImage_8);
```

or

```
//Use 8bit MCU, 16bpp color depth
```

```
BTE_MCU_Write_MCU_8bit(Layer1,Panel_width,0,0,Layer1,Panel_width,100,100,12,128,128
,gImage_16);
```

or

```
//Use 8bit MCU, 24bpp color depth
```

```
BTE_MCU_Write_MCU_8bit(Layer1,Panel_width,0,0,Layer1,Panel_width,100,100,12,128,128
,gImage_24);
```

or

```
//Use 16bit MCU, 16bpp color depth
```

```
BTE_MCU_Write_MCU_16bit(Layer1,Panel_width,0,0,Layer1,Panel_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(Layer1,Panel_width,0,0,Layer1,Panel_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(Layer1,Panel_width,0,0,Layer1,Panel_width,100,100,12,128,128
,pic1624);
```

**Condition:****Source 0 from MCU.****Source 1 : Start Address = Layer1, Image Width = Panel\_width, Coordinate = (0,0) .****Destination: Start Address = Layer1, Image Width = Panel\_width, Coordinate = (100,100) .****ROP Code = 12 → Destination data= Source 0 data, Don't care Source 1.****BTE Window Size = 128x128 .**

LCD display screen(Layer1):

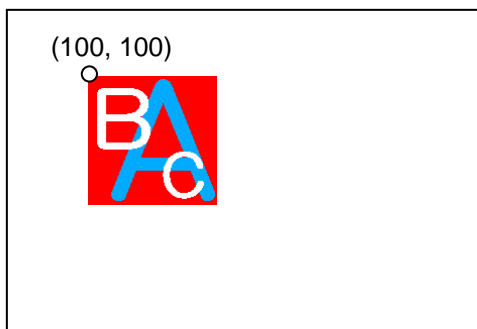
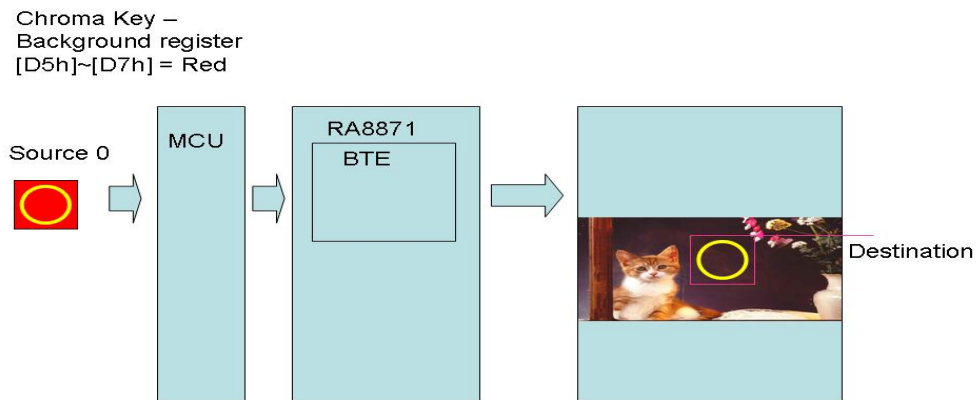


Figure 4-21: BTE MCU Write picture into SDRAM. The destination address is from Layer1(100,100) to Layer1(227,227)]

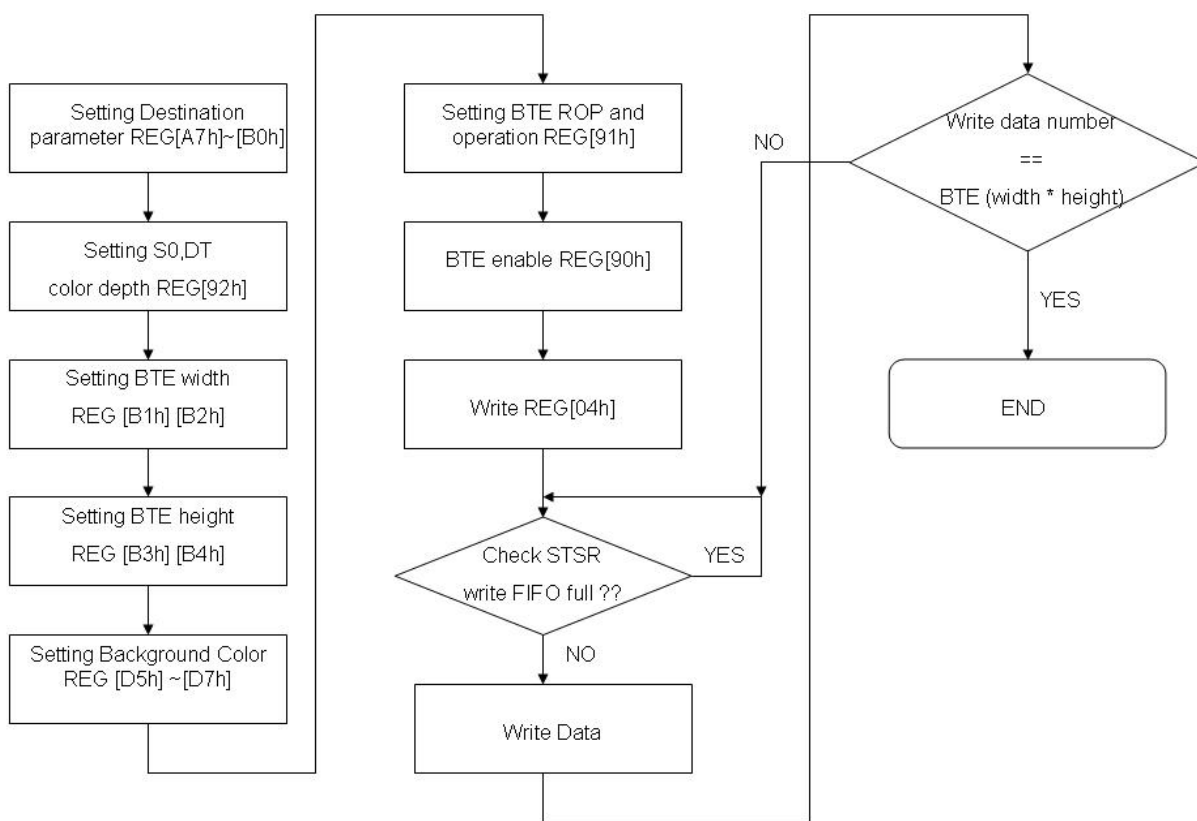
**4.3.3:MCU Write With Chroma Key (w/o ROP)**

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 RA8871M, 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**



**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 RA8871M, 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:

**Write\_Layer(1);**//memory address switch to layer1.Reference Page.5~6

**Show\_Layer(1);**//LCD show layer1 data. Reference Page.5~6

+

**//Use 8bit MCU , 8bpp color depth**

**BTE\_MCU\_Write\_Chroma\_key\_MCU\_8bit(Layer1,Panel\_width,100,100,0xe0,128,128,gImage\_8);**

or

**//Use 8bit MCU , 16bpp color depth**

**BTE\_MCU\_Write\_Chroma\_key\_MCU\_8bit(Layer1,Panel\_width,100,100,0xf800,128,128,gImage\_16);**

or

**//Use 8bit MCU , 24bpp color depth**

**BTE\_MCU\_Write\_Chroma\_key\_MCU\_8bit(Layer1,Panel\_width,100,100,0xff0000,128,128,gImage\_24);**

or

**//Use 16bit MCU , 16bpp color depth**

**BTE\_MCU\_Write\_Chroma\_key\_MCU\_16bit(Layer1,Panel\_width,100,100,0xf800,128,128,pic1616);**

or

**//Use 16bit MCU , 24bpp color depth and data format use mode 1**

**BTE\_MCU\_Write\_Chroma\_key\_MCU\_16bit(Layer1,Panel\_width,100,100,0xff0000,128,128,pic16241);**

or

**//Use 16bit MCU , 24bpp color depth and data format use mode 2**

**BTE\_MCU\_Write\_Chroma\_key\_MCU\_16bit(Layer1,Panel\_width,100,100,0xff0000,128,128,pic1624);**

**Condition:**

**Source 0 from MCU,**

**Destination: Start Address = Layer1, Image Width = Panel\_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.**

LCD data screen(Layer1):

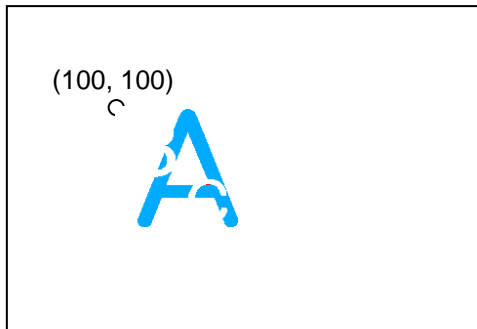
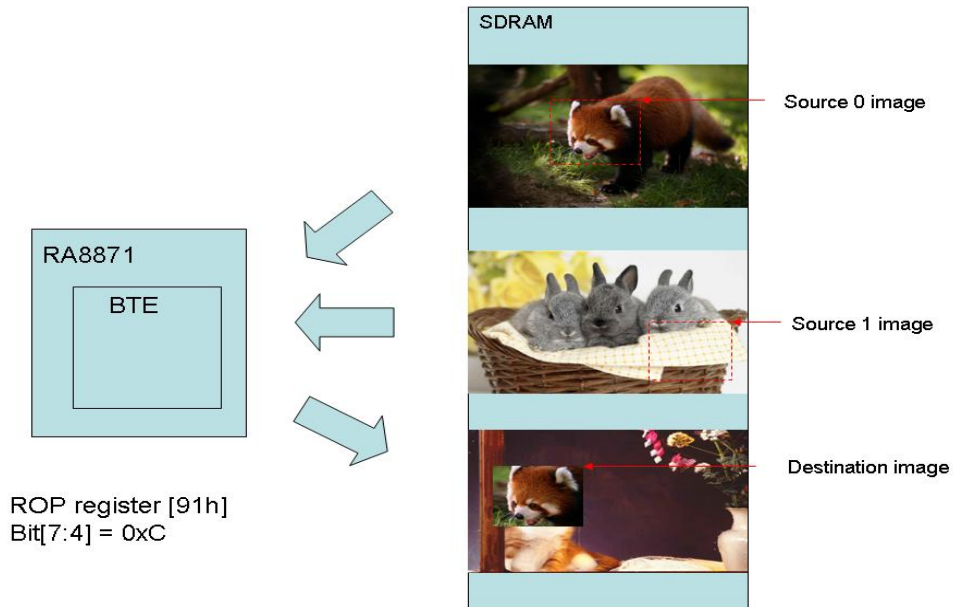


Figure 4-26: BTE MCU Write With Choma Key , Transparency color = red color.  
The destination address is from Layer1(100,100) to Layer1(227,227).

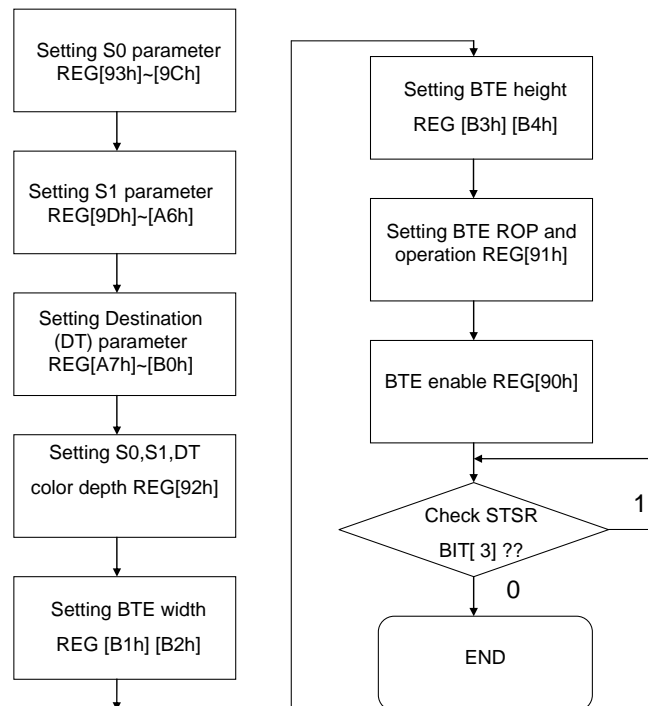


**4.4.1: Memory Copy with ROP**

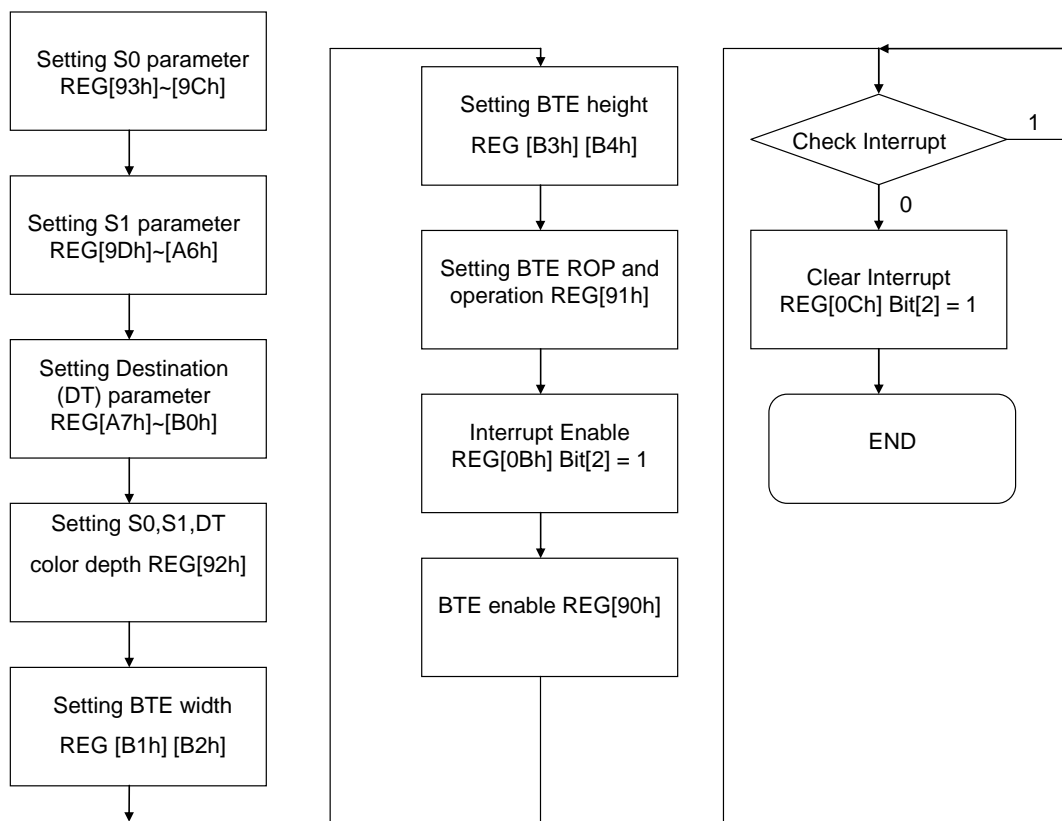
The Memory Copy moves a specific area of the SDRAM to a different area of the SDRAM. This operation can speed up the data copy operation from one block to another and save a lot of MCU processing time and loading.



**Figure 4-27 : Hardware Data Flow**



**Figure 4-28 : Flow Chart**



**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 = Layer1, Image Width = Panel_width, Coordinate = (0,0) .
Source 1 : Start Address = Layer1, Image Width = Panel_width, Coordinate = (0,0)
Destination: Start Address = Layer1, Image Width = Panel_width, Coordinate = (250,0) .
ROP Code = 12 →Destination = Source 0 , Don't care Source 1.
BTE Window Size = 128x128 .*/
```

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xe0,200,200); //8bpp color depth
```

```
Draw_Circle_Fill(0xfc,100,100,50); //8bpp color depth
```

or

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xf800,200,200); //16bpp color depth
```

```
Draw_Circle_Fill(0xffe0,100,100,50); //16bpp color depth
```

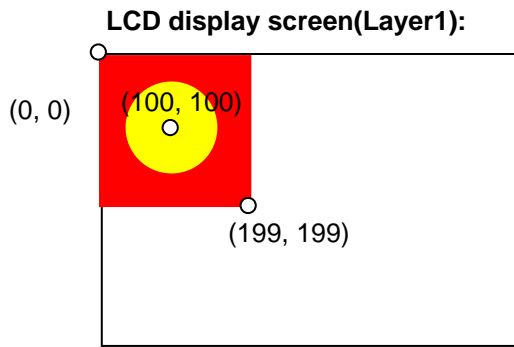
or

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xFF0000,200,200); //24bpp color depth
```

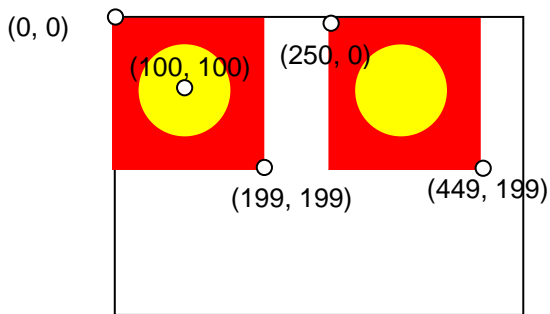
```
Draw_Circle_Fill(0xffff00,100,100,50); //24bpp color depth
```

+

```
BTE_Memory_Copy(Layer1,Panel_width,0,0,Layer1,Panel_width,0,0,Layer1,Panel_width,250,0,12,200,200);
```



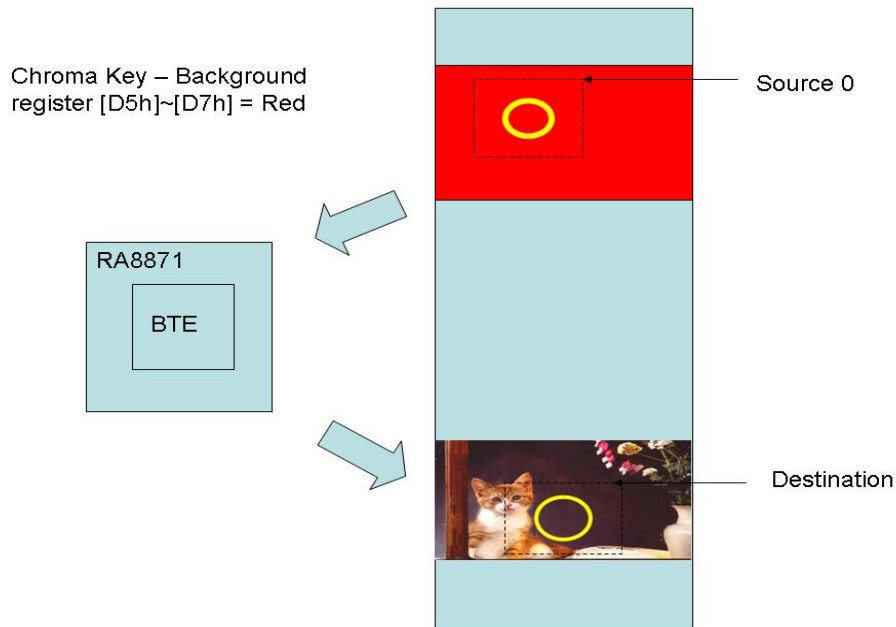
**Figure 4-30: draw a square filled with red and circle filled with yellow**



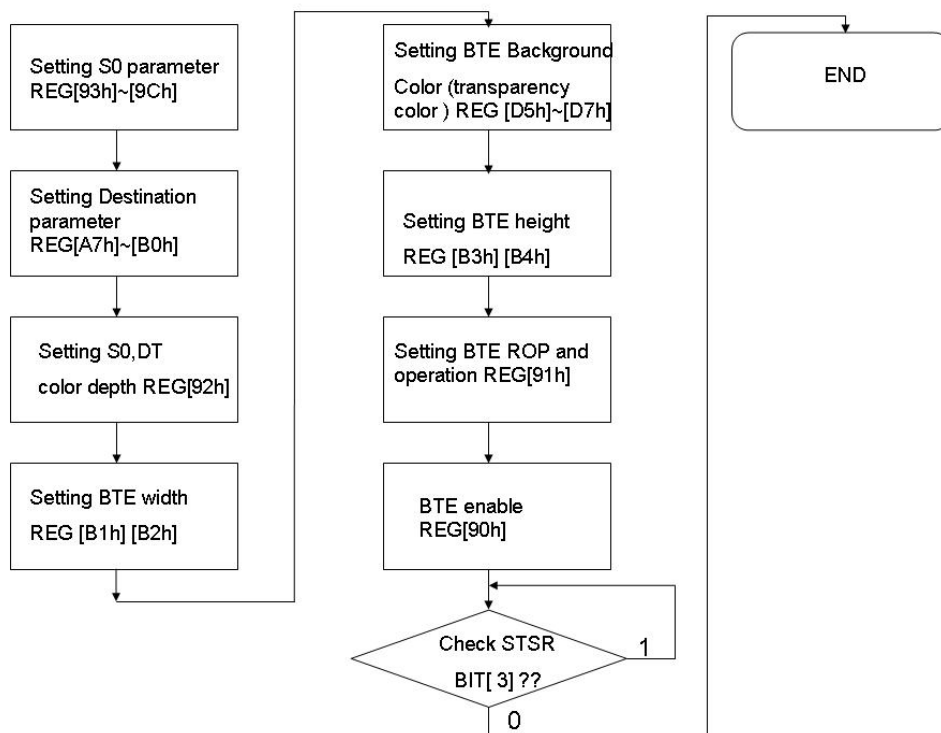
**Figure 4-31 : copy a same pattern by BTE Memory Copy with ROP**

**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**



**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 = Layer1, Image Width = Panel\_width, coordinate = (0,0) .*

*Source 1 : Start Address = Layer1, Image Width = Panel\_width, coordinate = (0,0)*

*Destination: Start Address = Layer1, Image Width = Panel\_width, coordinate = (250,0) .*

*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.\*/*

**Write\_Layer(1); //set memory read/write layer.Reference Page.5~6**

**Show\_Layer(1);//set LCD display layer. Reference Page.5~6**

**+**

**//8bpp color depth**

**BTE\_Solid\_Fill(Layer1,Panel\_width,0,0,0xe0,200,200);**

**Draw\_Circle\_Fill(0xfc,100,100,50);**

**BTE\_Memory\_Copy\_Chroma\_key(Layer1,Panel\_width,0,0,Layer1,Panel\_width,250,0,0xe0,200,200);**

**Or**

**//16bpp color depth**

```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xf800,200,200);
```

```
Draw_Circle_Fill(0xffe0,100,100,50);
```

```
BTE_Memory_Copy_Chroma_key(Layer1,Panel_width,0,0,Layer1,Panel_width,250,0,0xf800,200,200);
```

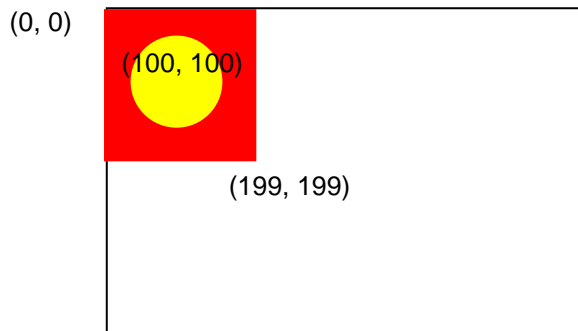
or

**//24bpp color depth**

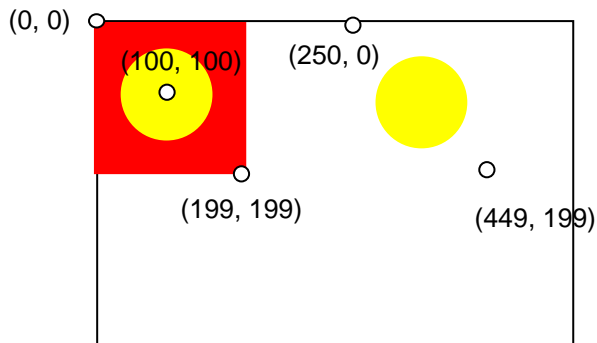
```
BTE_Solid_Fill(Layer1,Panel_width,0,0,0xFF0000,200,200);
```

```
Draw_Circle_Fill(0xffff00,100,100,50);
```

```
BTE_Memory_Copy_Chroma_key(Layer1,Panel_width,0,0,Layer1,Panel_width,250,0,0xff0000,200,200);
```



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

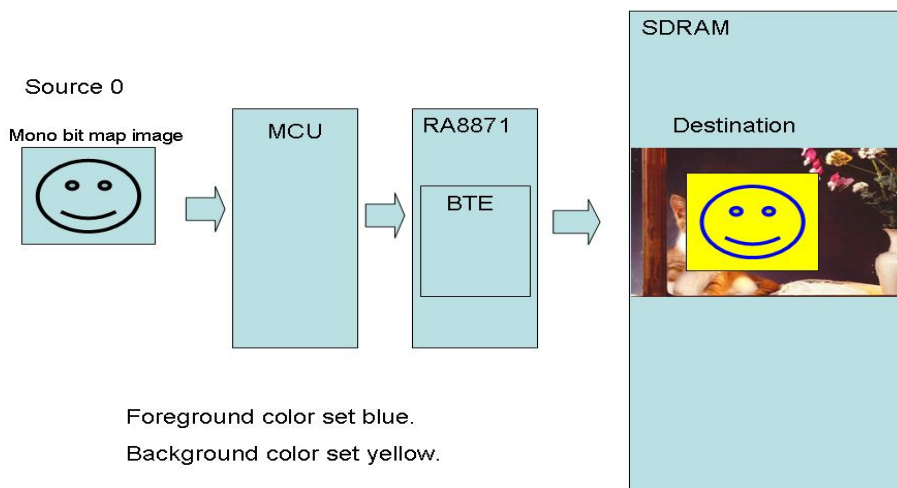


**Figure 4-35 : example for BTE Memory Copy with Chroma key**

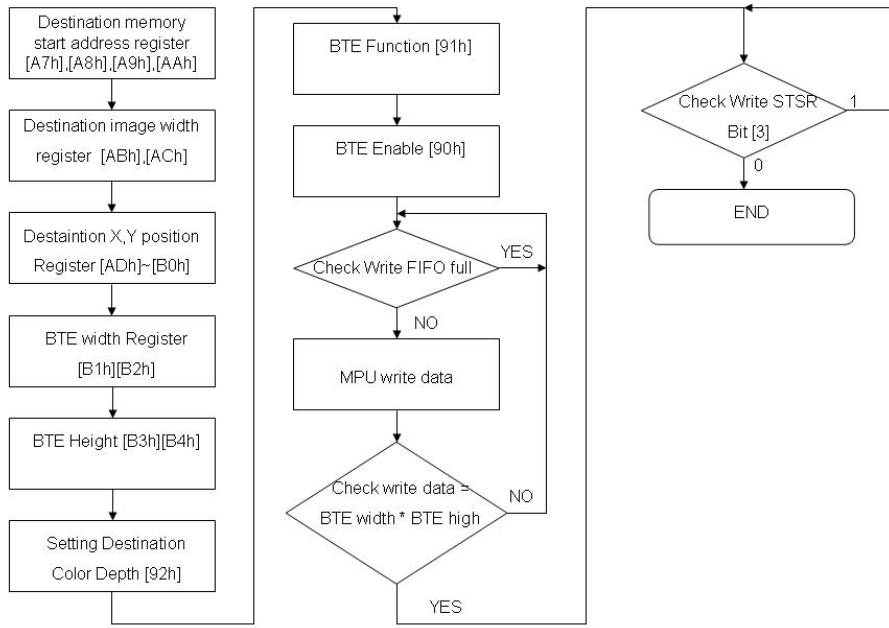


**4.5.1:MCU Write With Color Expansion**

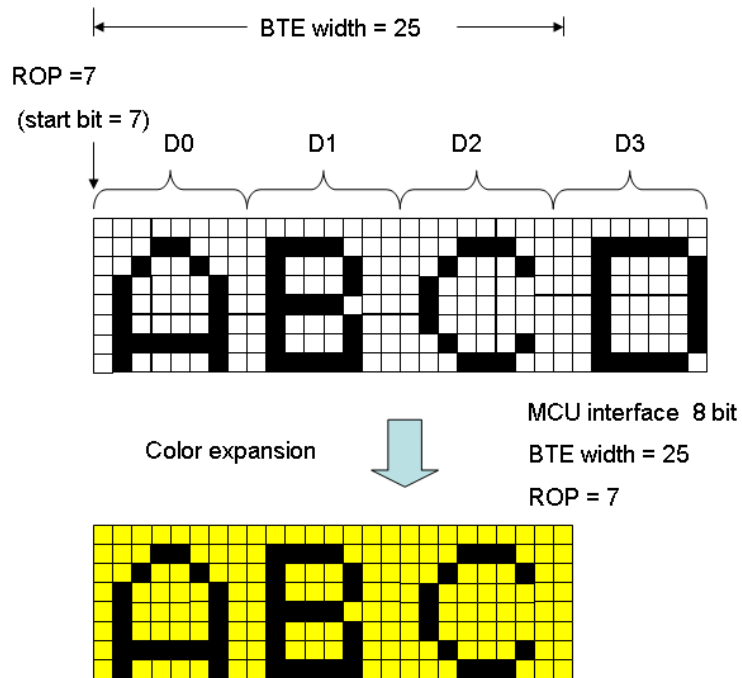
“MCU Write With Color Expansion” is a useful operation to translate monochrome data of MCU interface to be colorful one. In the operation, the source data will be treated as a monochrome bit-map. The bit-wise data is translated to multi-bits per pixel color data by the setting of “BTE Foreground Color” 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.



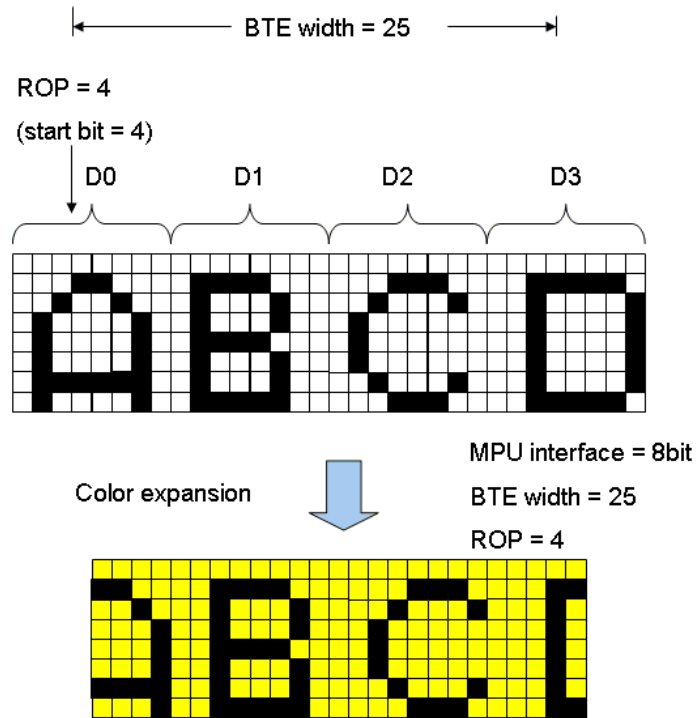
**Figure 4-36 : Hardware Data Flow**



**Figure 4-37 : Flow Chart**



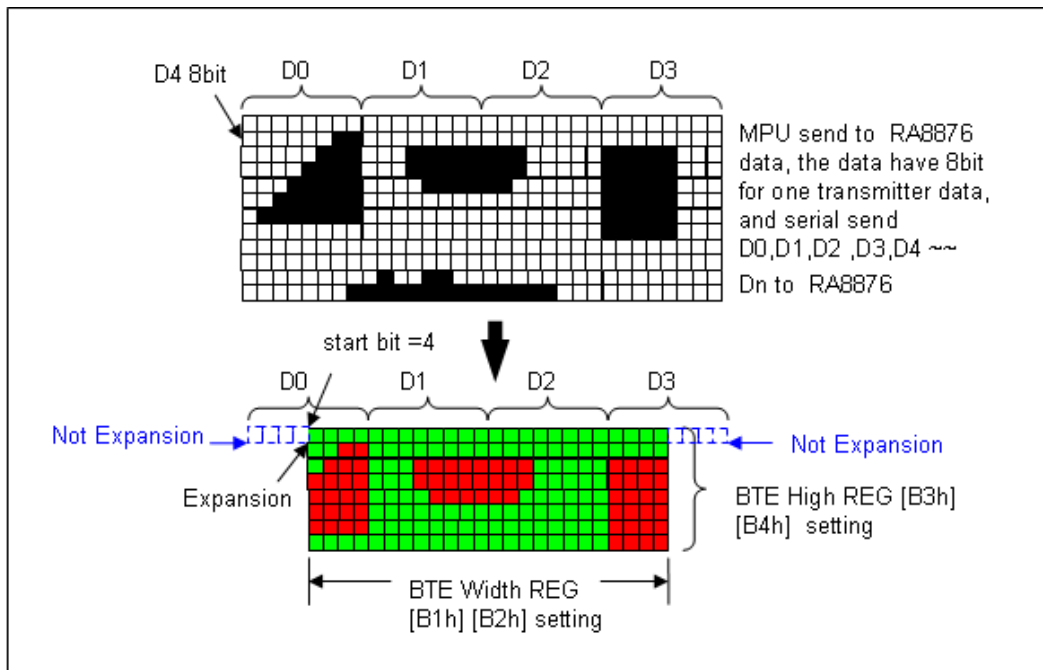
**Figure 4-38 :Start Bit Example 1**



**Figure 4-39 : Start bit Exapmle 2**

**Note:**

1. Calculate sent data numbers per row =  $((\text{BTE Width size REG} - (\text{MCU interface bits} - (\text{start bit} + 1))) / \text{MCU interface bits}) + ((\text{start bit} + 1) \% (\text{MCU interface}))$
2. Total data number =  $(\text{sent data numbers per row}) \times \text{BTE Vertical REG setting}$



**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)

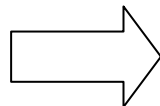


Figure 4-42 :

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 = Layer1

**Des\_W** : image width of Destination (recommend = canvas image width) = Panel\_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 \*/

**Write\_Layer(1);** //set memory read/write layer.Reference Page.5~6

**Show\_Layer(1);**//set LCD display layer. Reference Page.5~6

+

**//MCU\_8bit\_ColorDepth\_8bpp** //setting in UserDef.h

**BTE\_MCU\_Write\_ColorExpansion\_MCU\_8bit(Layer1,Panel\_width,0,0,128,128,0x03,0x1c,gImage\_1);**

or

**//MCU\_8bit\_ColorDepth\_16bpp** //setting in UserDef.h

**BTE\_MCU\_Write\_ColorExpansion\_MCU\_8bit(Layer1,Panel\_width,0,0,128,128,0x001f,0x07e0,gImage\_1);**

or

`//MCU_8bit_ColorDepth_24bpp` `//setting in UserDef.h`

```
BTE_MCU_Write_ColorExpansion_MCU_8bit(Layer1,Panel_width,0,0,128,128,0x0000ff,0x00ff00
,glImage_1);
```

or

`//MCU_16bit_ColorDepth_16bpp` `//setting in UserDef.h`

```
BTE_MCU_Write_ColorExpansion_MCU_16bit(Layer1,Panel_width,0,0,128,128,0x001f,0x07e0,Test);
```

or

`//MCU_16bit_ColorDepth_24bpp_Mode_1` `//setting in UserDef.h`

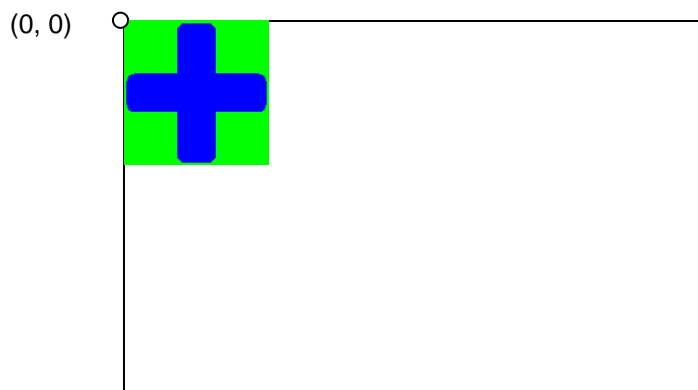
```
BTE_MCU_Write_ColorExpansion_MCU_16bit(Layer1,Panel_width,0,0,128,128,0x0000ff,0x00ff00,Test);
```

or

`//MCU_16bit_ColorDepth_24bpp_Mode_2` `//setting in UserDef.h`

```
BTE_MCU_Write_ColorExpansion_MCU_16bit(Layer1,Panel_width,0,0,128,128,0x0000ff,0x00ff00,Test);
```

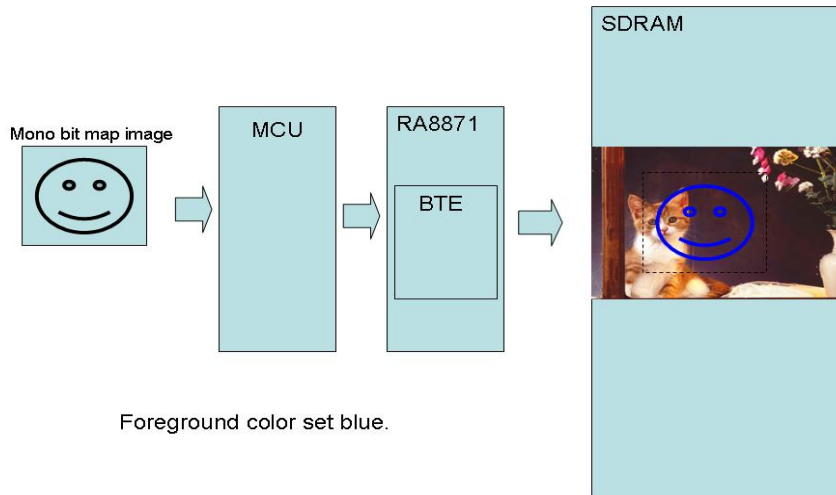
**Virtual display on LCD(Layer1):**



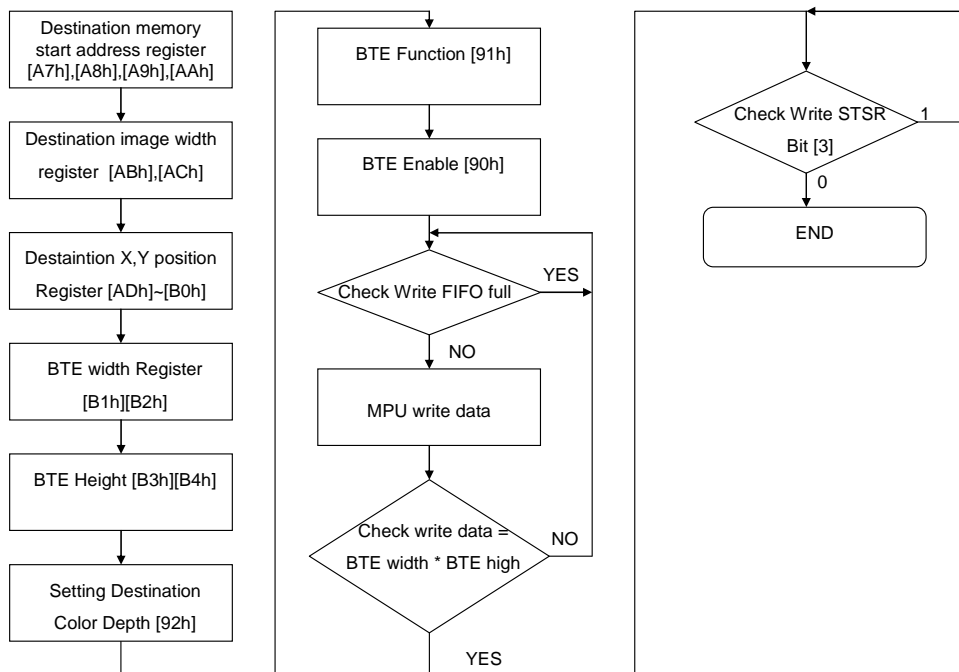
**Figure 4-43 : BTE MCU Write with Color Expansion**

**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**



**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)

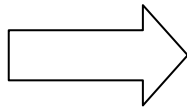


Figure 4-47 :

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 = Layer1*

*Des\_W : image width of Destination (recommend = canvas image width) = Panel\_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)\*/*

**Write\_Layer(1); //set memory read/write layer.Reference Page.5~6**

**Show\_Layer(1);//set LCD display layer. Reference Page.5~6**

**+**

**//MCU\_8bit\_ColorDepth\_8bpp //setting in UserDef.h**

**BTE\_MCU\_Write\_ColorExpansion\_Chroma\_key\_MCU\_8bit(Layer1,Panel\_width,0,0,128,128,0xe0**

**,gImage\_1);**

**or**

**//MCU\_8bit\_ColorDepth\_16bpp //setting in UserDef.h**

**BTE\_MCU\_Write\_ColorExpansion\_Chroma\_key\_MCU\_8bit(Layer1,Panel\_width,0,0,128,128,0xf800**

**,gImage\_1);**

**or**

**//MCU\_8bit\_ColorDepth\_24bpp //setting in UserDef.h**

**BTE\_MCU\_Write\_ColorExpansion\_Chroma\_key\_MCU\_8bit(Layer1,Panel\_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(Layer1,Panel\_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(Layer1,Panel\_width,0,0,128,128,0xff0000**

```
,Test);
```

```
or
```

```
//MCU_16bit_ColorDepth_24bpp_Mode_2
```

```
//setting in UserDef.h
```

```
BTE_MCU_Write_ColorExpansion_Chroma_key_MCU_16bit(Layer1,Panel_width,0,0,128,128,0xff0000
```

```
,Test);
```

LCD display screen(Layer1):

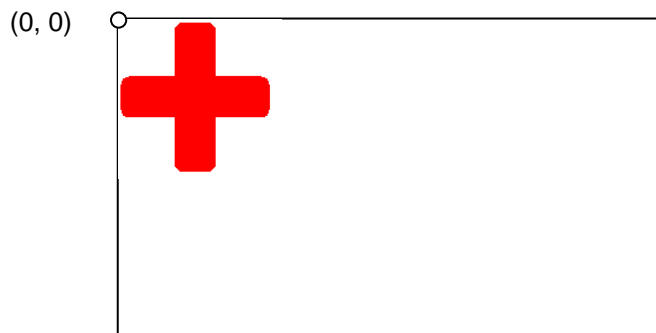


Figure 4-48 : BTE MCU Write With Color Expansion and Chroma key

**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. RA8871M 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 RA8871M.

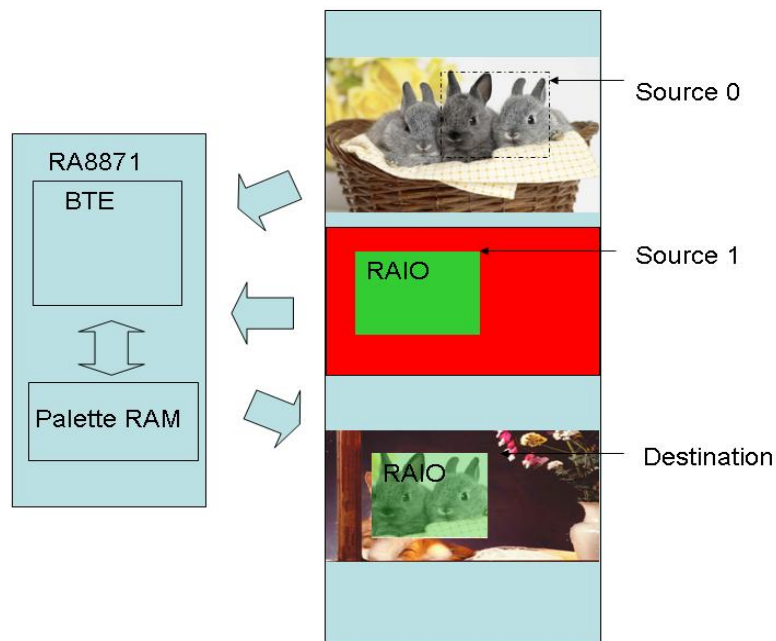
**4.6.1: Memory Copy with Alpha Blending**

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**. , Picture can be operated in 8 bpp/16bpp/24bpp mode. Pixel mode is only operated in 8bpp/16bpp mode. In the pixel mode – 16bpp the source 1 data bit [15:12] is alpha level, the other bits are color data, in pixel mode 8bpp the source 1 data [7:6] is alpha level. Bit [5:0] is use to destine palette color data.

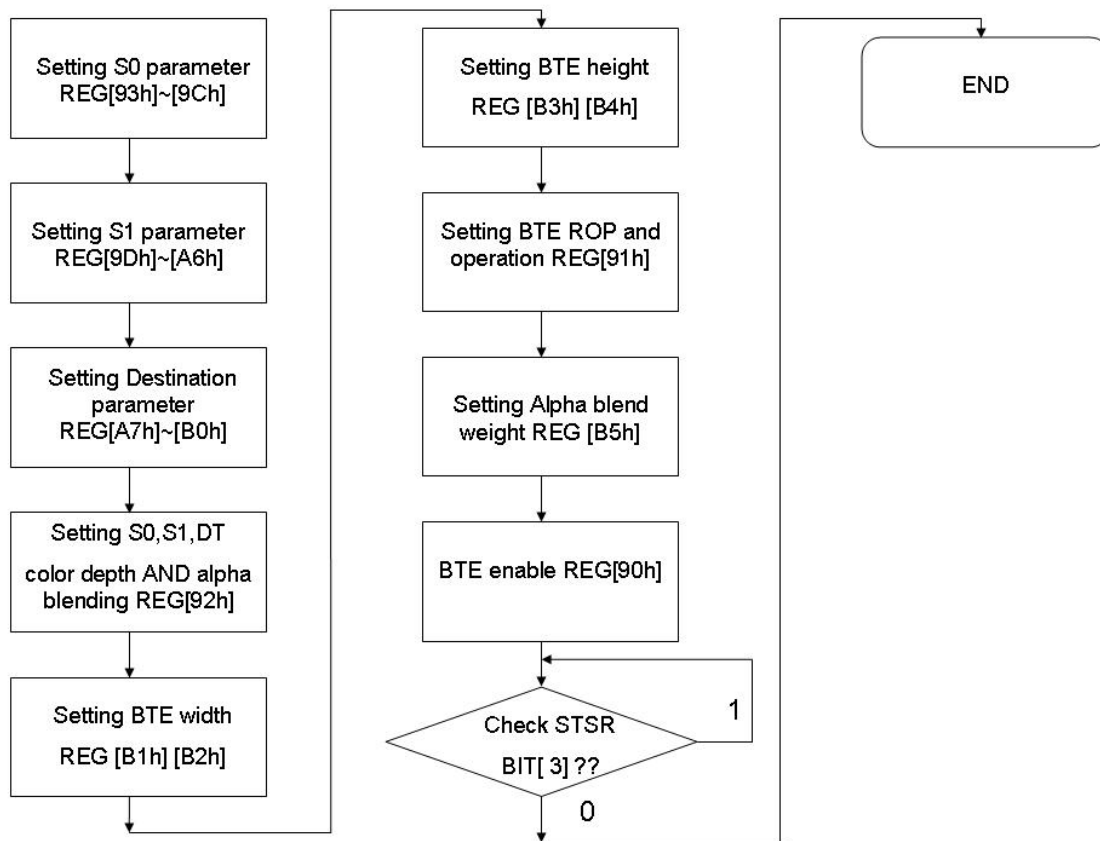
Picture mode - Destination data = (Source 0 \* (1- alpha Level)) + (Source 1 \* alpha Level);

Pixel mode 16bpp - Destination data = (Source 0 \* (1 - alpha Level)) + (Source 1 [11:0] \* alpha Level)

Pixel mode 8bpp – Destination data = (Source 0 \* (1 - alpha Level)) + (Index palette (Source 1[5:0]) \* alpha Level)



**Figure 4-49 : Picture Mode Hardware Data Flow**



**Figure 4-50 : Picture Mode Flow Chart**

#### 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 = Layer1, Image Width = Panel\_width, coordinate = (0,0) .*

*Source 1 : Start Address = Layer2, Image Width = Panel\_width, coordinate = (0,0) .*

*Destination: Start Address = Layer1, Image Width = Panel\_width, coordinate = (0,0) .*

*BTE Window Size = 200x200 , alpha = 16 .\*/*

**Write\_Layer(1); //set memory read/write layer.Reference Page.5~6**

**Show\_Layer(1);//set LCD display layer. Reference Page.5~6**

**+**

**//When Color Depth = 8bpp**

**DMA\_24bit(1,0,0,0,480,272,480,6144000);**

**BTE\_Solid\_Fill(Layer2,Panel\_width,0,0,0x1c,200,200);**

**or**

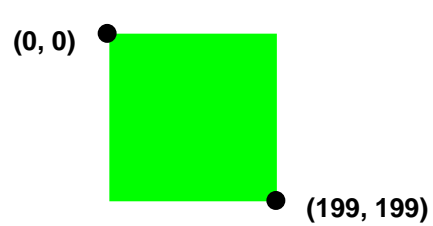
**//When Color Depth = 16bpp**

```
DMA_24bit(1,0,0,0,480,272,480,4761600);
BTE_Solid_Fill(Layer2,Panel_width,0,0,0x07e0,200,200);
or
//When Color Depth = 24bpp
DMA_24bit(1,0,0,0,480,272,480,2304000);
BTE_Solid_Fill(Layer2,Panel_width,0,0,0x00FF00,200,200);
+
BTE_Alpha_Blending(Layer1,Panel_width,0,0,Layer2,Panel_width,0,0,Layer1,Panel_width,0,0,200,200,16);
```

Layer 1 data:



Layer 2 data:



LCD screen:



Figure 4-51 : Source0 is Layer1[(0,0) ~ (199,199)] ,Source1 is Layer2[(0,0) ~ (199,199)], executes alpha blending and showing on Destination Layer1[(0,0) ~ (199,199)], alpha = 16.

Destination:

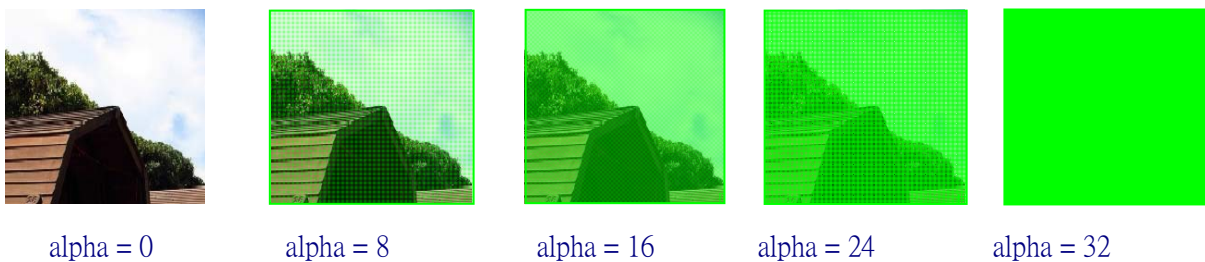


Figure 4-52: Destination in different alpha/transparent level.

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

## Chapter 5: Picture In Picture Function

### 5.1: PIP Window

RA8871M 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.

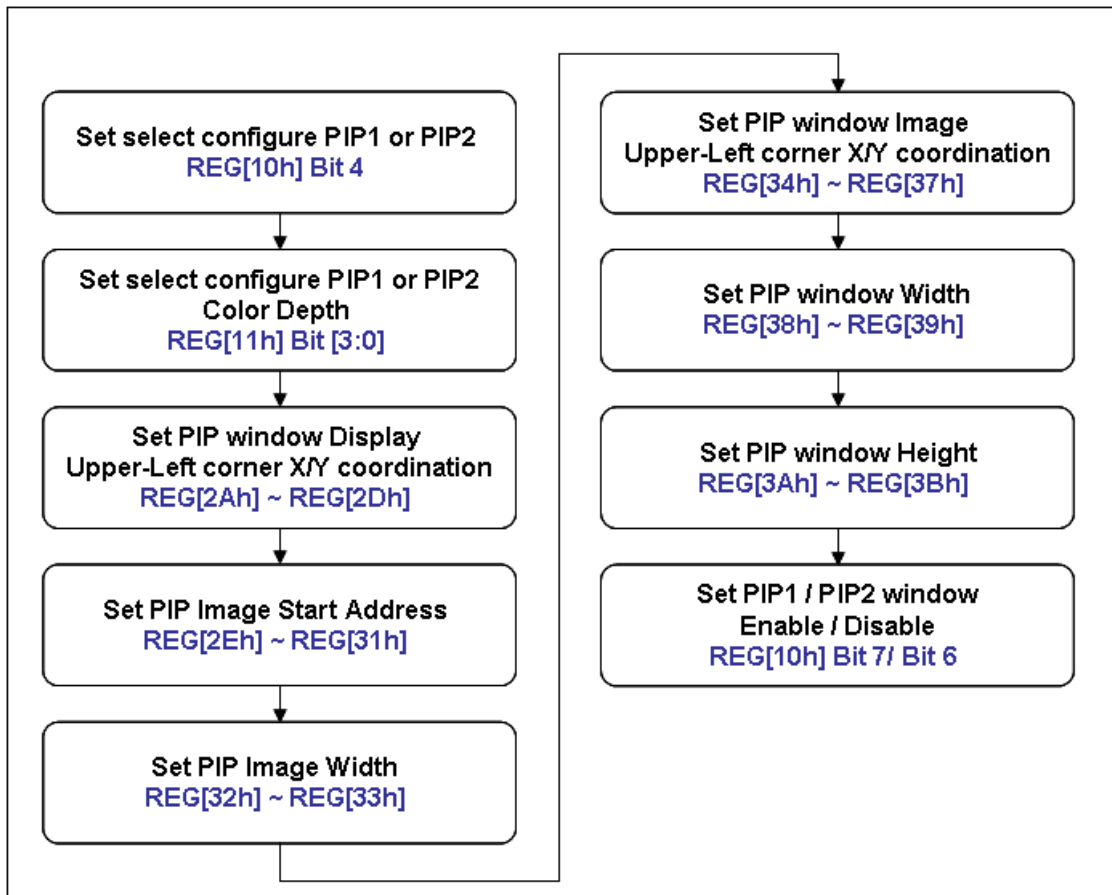
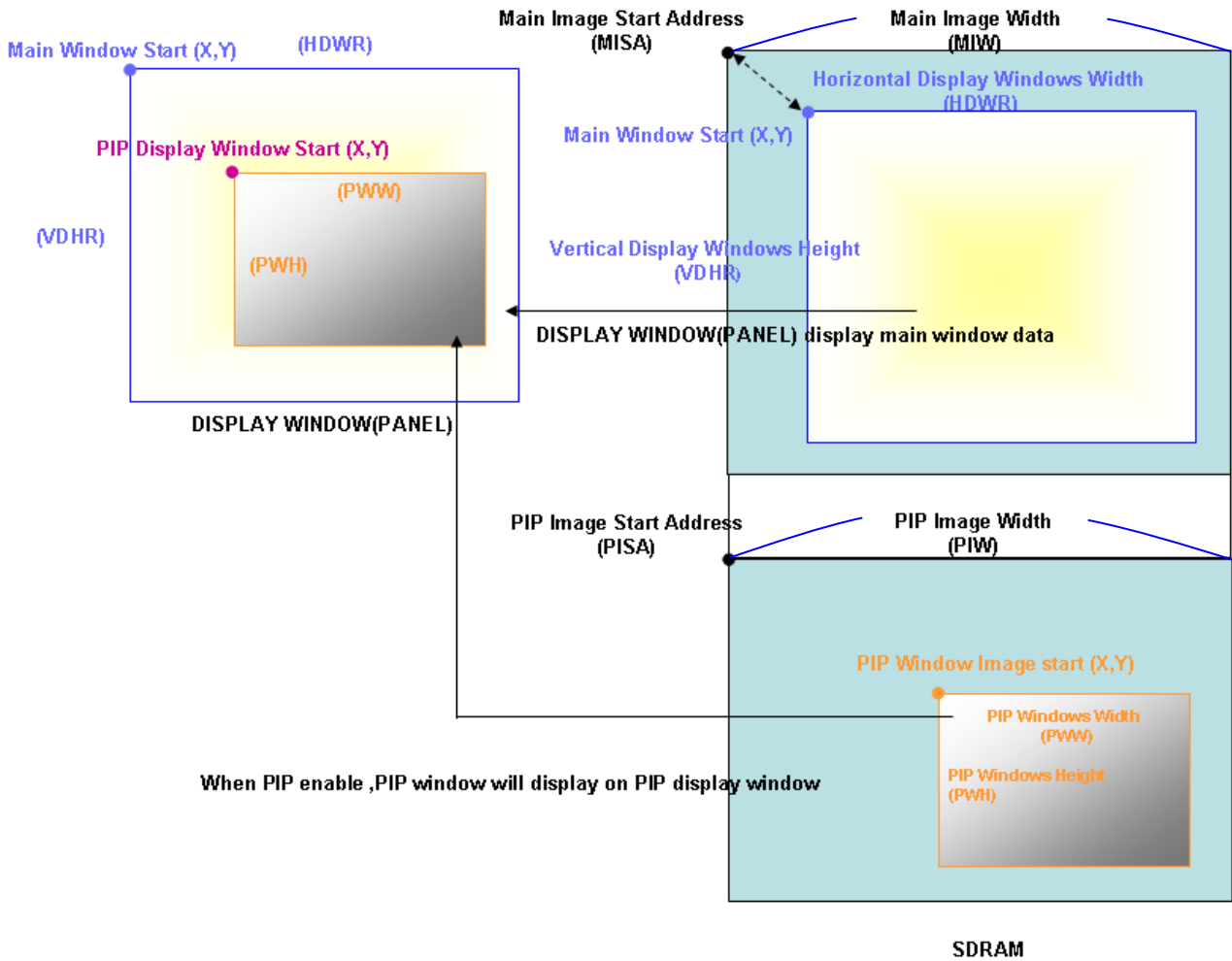


Figure 5-1 Flow Char

**5.3: Diagrammatic explanation for PIP function**



**Figure 5-2**



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

RA8871M 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)

`Write_Layer(2); //set memory read/write layer.Reference Page.5~6`

`Show_Layer(1); //set LCD display layer. Reference Page.5~6`

+

`//When color depth =8`

`DMA_24bit(1,0,0,0,480,320,480,6144000); //write 480x320 picture to layer 2`

`Write_Layer(3); //set memory read/write layer.Reference Page.5~6`

`DMA_24bit(1,0,0,0,480,320,480, 6297600); //write 480x320picture to layer 3`

Or

`//When color depth = 16bpp`

`DMA_24bit(1,0,0,0,480,320,480, 4761600); //write 480x320picture to layer 2`

`Write_Layer(3); //set memory read/write layer.Reference Page.5~6`

`DMA_24bit(1,0,0,0,480,320,480, 5068800); //write 480x320picture to layer 3`

Or

`//When color depth = 24bpp`

`DMA_24bit(1,0,0,0,480,320,480, 2304000); //write 480x320picture to layer 2`

`Write_Layer(3); //set memory read/write layer.Reference Page.5~6`

`DMA_24bit(1,0,0,0,480,320,480, 2764800); //write 480x320picture to layer 3`

+

**PIP(1,1,Layer2,0,0,Panel\_width,0,0,400,480);**

**PIP(1,2,Layer3,0,0,Panel\_width,400,0,400,480);**

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

**Layer 1 data:**



480x320 fill white color

**Layer 2 data:**



480x320 picture

**Layer 3 data:**



480x320 picture

**Figure 5-3 :Write wanted display data into SDRAM**

**Step 2 :**

Turn on the PIP function, PIP 1 window is Layer 2[(0,0) to (199,319)], the display position is Layer 1 [(0,0) to (199,319)].

PIP 2 window is Layer 3[(0 , 0) to(199,319)], the display position is Layer 1[(200,0) to (479,319)], LCD screen display Layer 1 information.

**PIP 1 and PIP2 off, LCD screen as follows:**



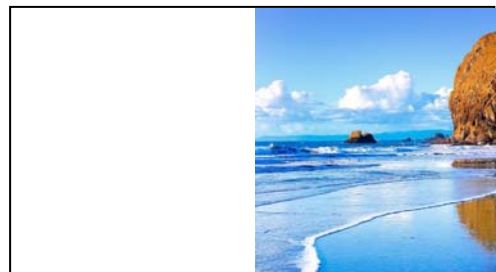
**PIP 1 and PIP2 on, LCD screen as follows:**



**PIP 1 on, PIP2 off, LCD screen as follows:**



**PIP 1 off, PIP2 on, LCD screen as follows:**



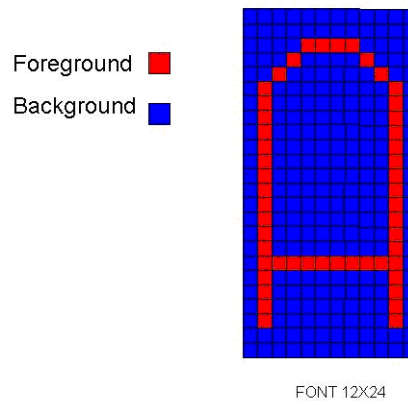
## Chapter 6 : Font

RA8871M 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 RA8871M can be classified with two sources:

### TEXT INPUT

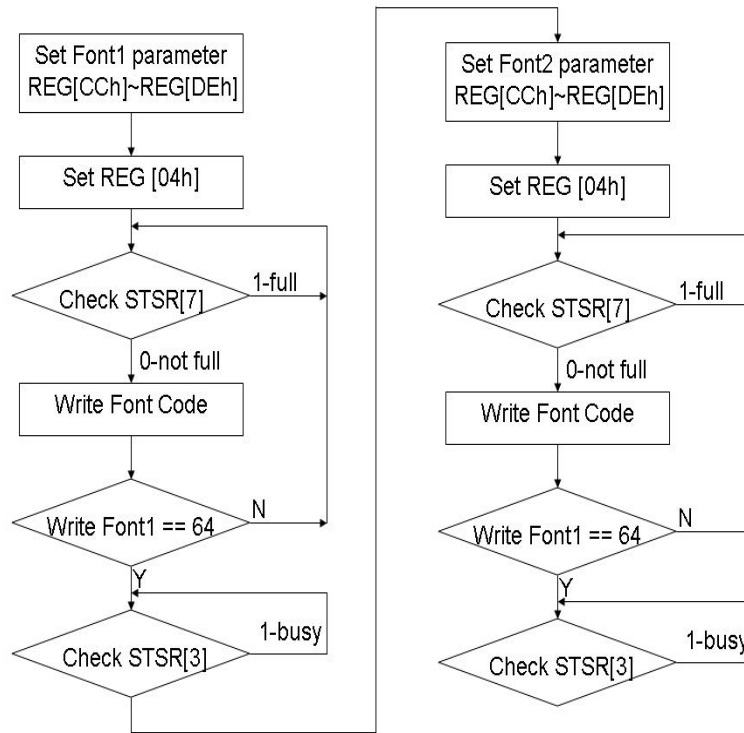
1. Embedded Characters .
2. External Character ROM .



**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].

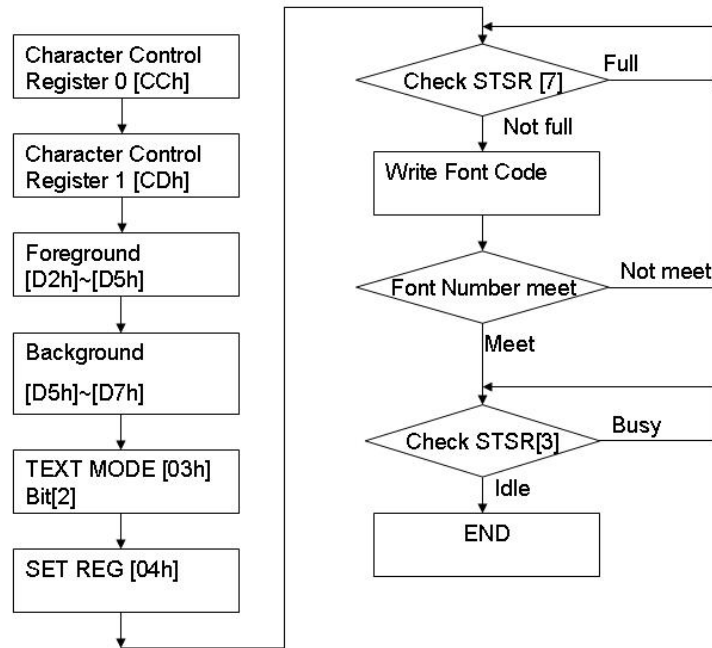
Ex : We write 64 characters as font1 and 64 characters as font2 to RA8871M



**Figure 6-2 : Font display flow chart**

**Embedded Characters**

The RA8871M embedded 8x16,12x24,16x32 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	€	,	f	„	…	†	‡	^	%	Š	<	Œ		Ž		
9		‘	’	“	”	•	-	-	~	™	Š	>	œ	ž	ÿ	
A		ı	ø	£	¤	¥	!	§	”	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ë	Ì	Í	Î
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ë	ì	í	î
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	{		}	~	
	À	Á	Â	Ã	Ä	Å	Ā	Ă	Ą	Ć	Č	Ĉ	Ď	Ě	Ě	Ě
	à	á	â	ã	ä	å	ā	ă	ą	ć	č	ĉ	ď	ě	ě	ě
	Ř	Á	Â	Ã	Ä	Å	Ā	Ă	Ą	Ć	Č	Ĉ	Ď	Ě	Ě	Ě
	ř	á	â	ã	ä	å	ā	ă	ą	ć	č	ĉ	ď	ě	ě	ě
	Đ	Ń	Ń	Ó	Ô	Ö	×	Ř	Ů	Ú	Ů	Ů	Ý	Ť	ß	
	đ	ń	ń	ó	ô	ö	÷	ř	ů	ú	ů	ů	ý	ť	ß	

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 fourth 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		À	Ā	Ą	Ȧ	Ā	Ī	Ĺ	Š	”	Š	Ē	Ĝ	Ʀ	-	Ž	̄
B	°	à	á	ą	ȧ	ā	ī	ĺ	š	”	š	ē	ĝ	ƞ	đ	ž	̇
C	̄	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā	Ā
D	Đ	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń	Ń
E	ā	á	ą	ȧ	ā	ā	ī	ĺ	š	”	š	ē	ĝ	ƞ	đ	ž	̇
F	đ	ñ	õ	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	Ц	
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		Њ	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	ѕ	ў	џ

## External Character ROM

RA8871M 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	√	√	√	√	√
Arial					√
Roman					
Bold					

\*Arial & Roman is variable width.

- Character height: x24

Allowed character sets & width:

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

\*Arial & Roman is variable width.

- Character height: x24

Allowed character sets & width:

	GB2312	GB2312 Extension	ASCII
Normal	√	√	√
Arial			√
Roman			√
Bold			

\*Arial & Roman is variable width.

- Character height: x32

Allowed character sets & width:

	GB2312	GB2312 Extension	ASCII
Normal	√	√	√
Arial			√
Roman			√
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	√	√	√	√
Arial	√	√	√	√
Roman	√			
Bold	√			

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

\*Arial & Roman is variable width.

- Character height: x24

Allowed character sets & width:

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

\*Arial & Roman is variable width.

## 6.2: Display results on LCD for showing Fonts

### 6.2.1: Internal Font :

The RA8871M embedded 8x16,12x24,16x32 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 = Panel_width
Y_H : active window height = Panel_length
FontColor : Set Font Color = 0xe0(8bpp) 、 0xf800(16bpp) 、 0xff0000(24bpp)
BackGroundColor : Set Font BackGround Color = 0x1c(8bpp) 、 0x07e0(16bpp) 、 0x00ff00(24bpp)*/

```

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1); //set LCD display layer. Reference Page.5~6
```

+

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,2,2,0,0);
```

+

```
//When Color Depth = 8bpp
```

```
Print_Internal_Font_String(0,0, Panel_width,Panel_length,0xe0,0x1c, "adfsdfdgfhhgfh");
```

Or

```
//When Color Depth = 16bpp
```

```
Print_Internal_Font_String(0,0, Panel_width,Panel_length,0xf800,0x07e0, "adfsdfdgfhhgfh");
```

or

```
//When Color Depth = 24bpp
```

```
Print_Internal_Font_String(0,0, Panel_width,Panel_length,0xff0000,0x00ff00, "adfsdfdgfhhgfh");
```

Virtual display on LCD :

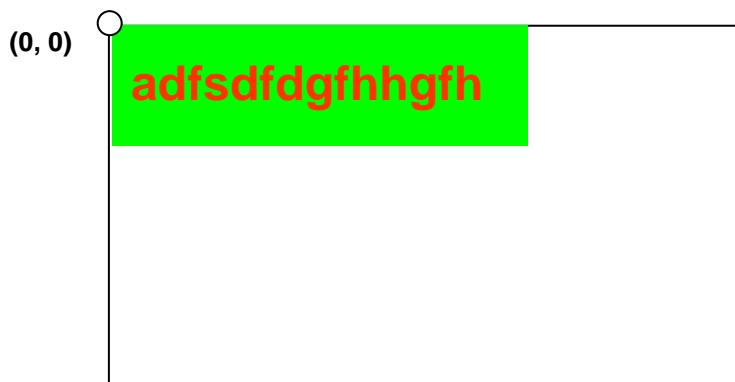


Figure 6-4 Internal Font Example

### 6.2.2:Print Big5 String by external font rom :

RA8871M 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 RA8871M 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 SCS //0 : use CS0 , 1 : use CS1
  ,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 = Panel_width
Y_H : active window height = Panel_length
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*/
```

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1); //set LCD display layer. Reference Page.5~6
```

+

```
//When Color Depth = 8bpp
```

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,2,2,0,0);
```

+

```
Print_BIG5String(3,0,0,0, Panel_width,Panel_length,0xe0,0x1c,"瑞佑科技 123456");
```

Or

```
//When Color Depth = 16bpp
```

```
Print_BIG5String(3,0,0,0, Panel_width,Panel_length,0xf800,0x07e0,"瑞佑科技 123456");
```

Or

```
//When Color Depth = 24bpp
```

```
Print_BIG5String(3,0,0,0, Panel_width,Panel_length,0xff0000,0x00ff00,"瑞佑科技 123456");
```

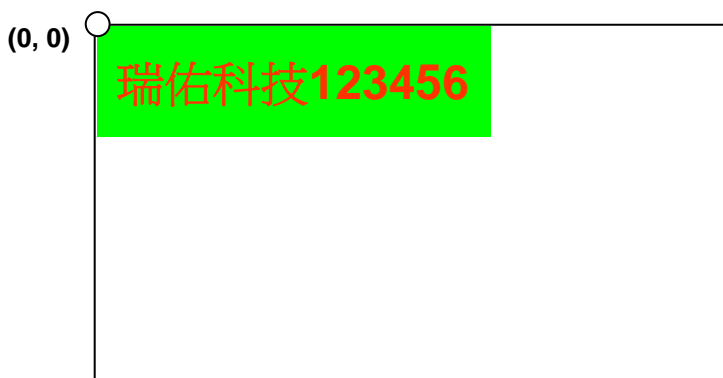


Figure 6-5 Print Big5 Font Example by External Font ROM



### 6.2.3:Print GB2312 String by external font rom :

RA8871M 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 RA8871M 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 SCS //SCS : 0 = CS0 , 1 = CS1
  ,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 = Panel_width
Y_H : active window height = Panel_length
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*/
```

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,2,2,0,0);
```

+

```
//When Color Depth = 8bpp
```

```
Print_GB2312String(3,0,0,0,Panel_width,Panel_length,0xe0,0x03," 瑞佑科技 123456");
```

Or

```
//When Color Depth = 16bpp
```

```
Print_GB2312String(3,0,0,0,Panel_width,Panel_length, 0xf800,0x001f," 瑞佑科技 123456");
```

or

```
//When Color Depth = 24bpp
```

```
Print_GB2312String(3,0,0,0,Panel_width,Panel_length, 0xff0000,0x0000ff," 瑞佑科技 123456");
```

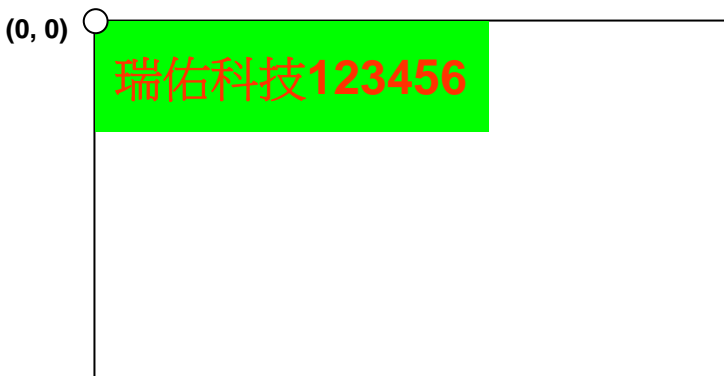


Figure 6-6 Print GB2312 Font Example by External Font ROM

### 6.3.4:Print GB12345 String by external font rom:

RA8871M 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 RA8871M 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 SCS //SCS : 0 = CS0 , 1 = CS1
  ,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 :

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,2,2,0,0);
```

+

```
//When Color Depth = 8bpp
```

```
Print_GB2312String(3,0,0,0,Panel_width,Panel_length,0xe0,0x03," 瑞佑科技 123456");
```

Or

```
//When Color Depth = 16bpp
```

```
Print_GB2312String(3,0,0,0,Panel_width,Panel_length, 0xf800,0x001f," 瑞佑科技 123456");
```

or

```
//When Color Depth = 24bpp
```

```
Print_GB2312String(3,0,0,0,Panel_width,Panel_length, 0xff0000,0x0000ff," 瑞佑科技 123456");
```

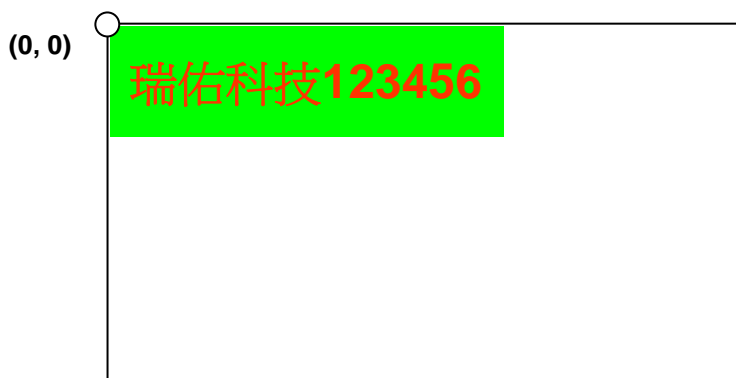


Figure 6-7 Print GB12345 Font Example by External Font ROM

### 6.3.5:Print Unicode String by external font rom:

RA8871M 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 RA8871M 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 SCS //SCS : 0 = CS0 , 1 = CS1
  ,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 :

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
Select_Font_Height_WxN_HxN_ChromaKey_Alignment(24,2,2,0,0);
```

+

```
//When Color Depth = 8bpp
```

```
Print_UnicodeString(3,0,0,0,Panel_width,Panel_length,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(3,0,0,0,Panel_width,Panel_length,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(3,0,0,0,Panel_width,Panel_length,0xff0000,0x00ff00,L"新竹縣竹北市台元一街 8 號 6 樓之 5");
```

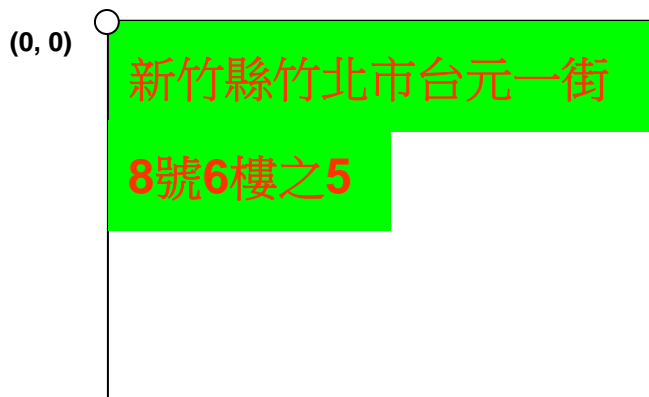


Figure 6-8 : Print Unicode Font Example by External Font ROM

## Chapter 7 : PWM (Pulse Width Modulation)

The RA8871M 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:

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

$$\text{PWM output period} = (\text{Count Buffer} + 1) \times \text{PWM CLK time}$$

$$\text{PWM output high level time} = (\text{Compare Buffer} + 1) \times \text{PWM CLK time}$$

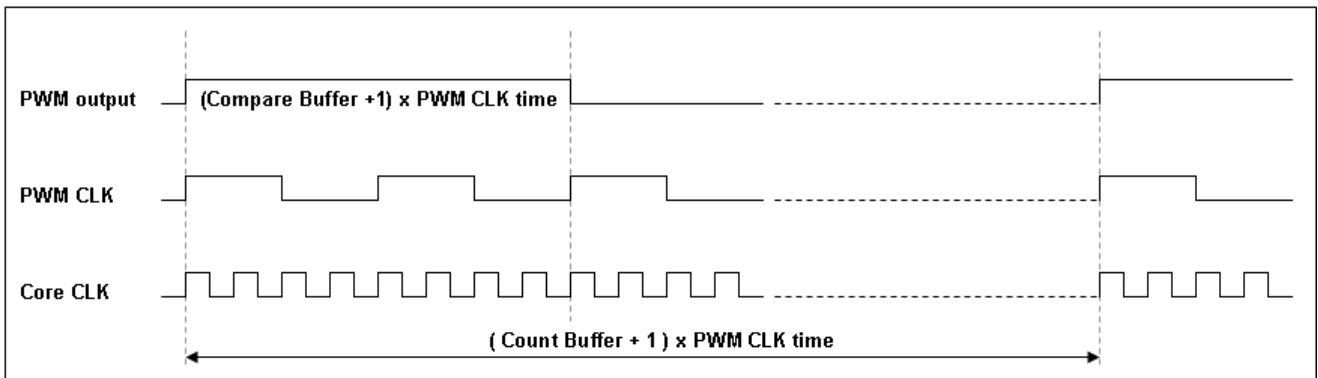


Figure 7-1 : PWM Timing Diagram

**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 Prescaler //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 / Prescaler ) /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 Prescaler //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 / Prescaler ) /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.*

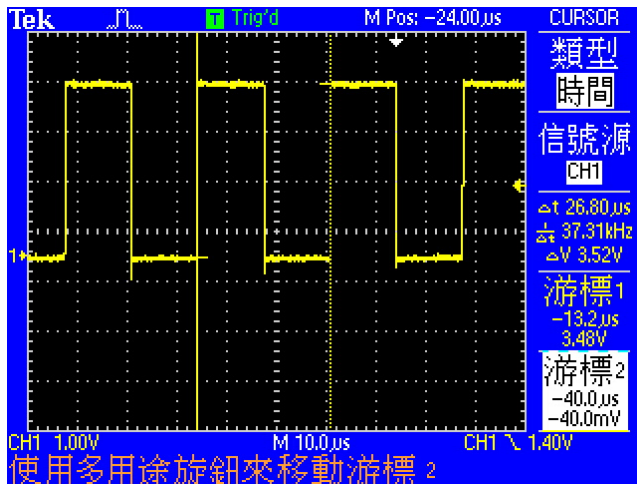
**Clock Divided = 3, Prescaler = 100, Count\_Buffer = 1, Compare\_Buffer = 0.**

**PWM CLK = (Core CLK / Prescaler) / 2<sup>clock divided</sup> = (60M / 100) / 2<sup>3</sup> = 75KHz**

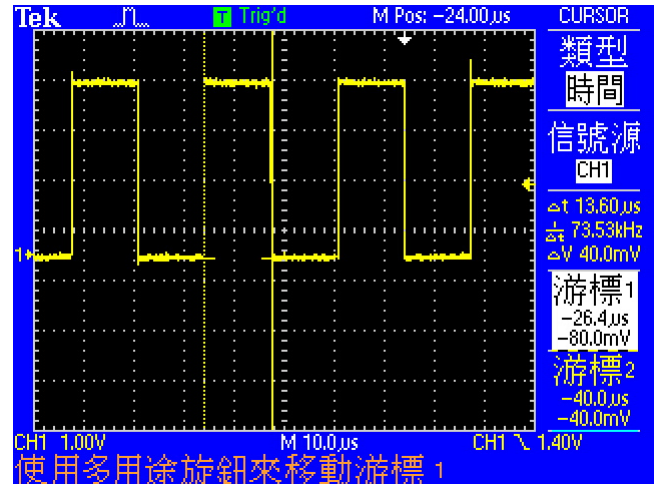
**PWM output period = (Count Buffer + 1) x PWM CLK time = (1+1) x (1 / 75K) ≈ 26.67us**

**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.*

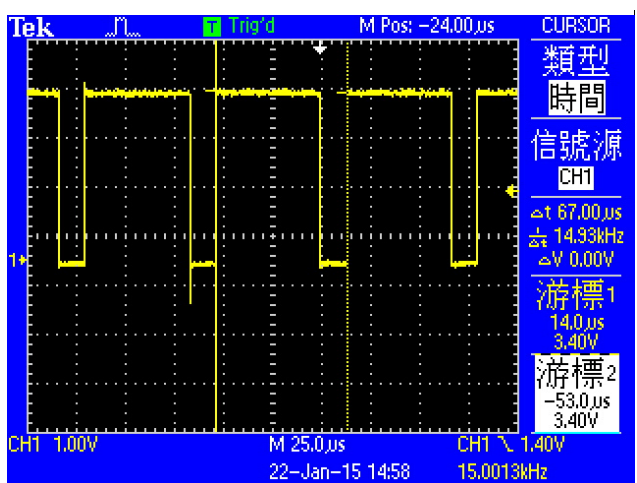
**Clock Divided = 3, Prescaler = 100, Count\_Buffer = 4, Compare\_Buffer = 3.**

**PWM CLK = (Core CLK / Prescaler) / 2<sup>clock divided</sup> = (60M / 100) / 2<sup>3</sup> = 75KHz**

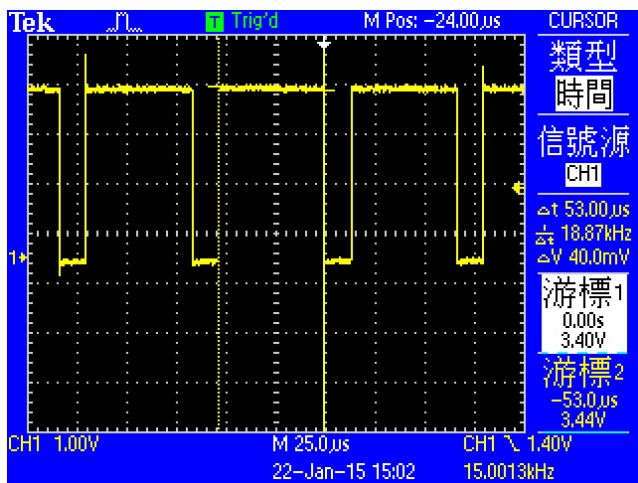
**PWM output period = (Count Buffer + 1) x PWM CLK time = (4+1) x (1 / 75K) ≈ 66.67us**

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

**Oscilloscope waveform:**



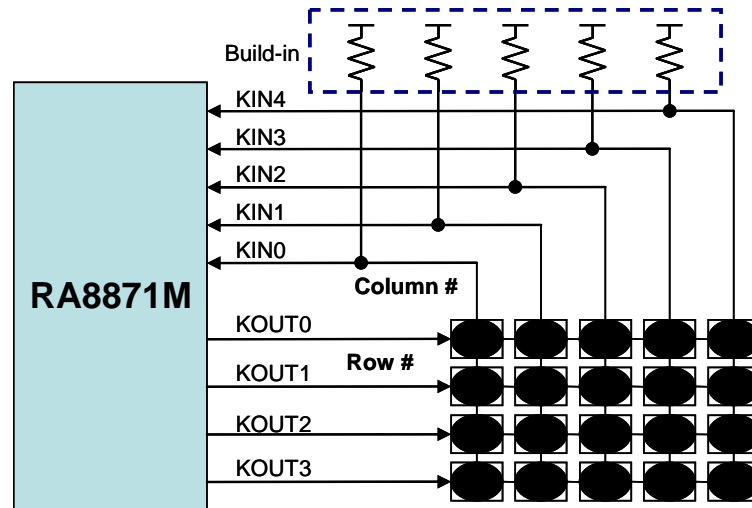
**Figure 7-4 : PWM1 output period waveform**



**Figure 7-5 : PWM1 output high level time**

**Chaper 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 8-3 shows the basic application circuit of Key-Pad on digital panel package type. RA8871M already built-in pull-up resistors in the pins “XKIN[4:0]” so no external circuit is needed.



**Figure 8-3 : Key-Pad Application**

**8.1 Operation**

The RA8871M 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 KEYSKAN control and status register, it is used to configure the options for KEYSKAN, 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 KEYSKAN. 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 RA8871M. “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 RA8871M. “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.

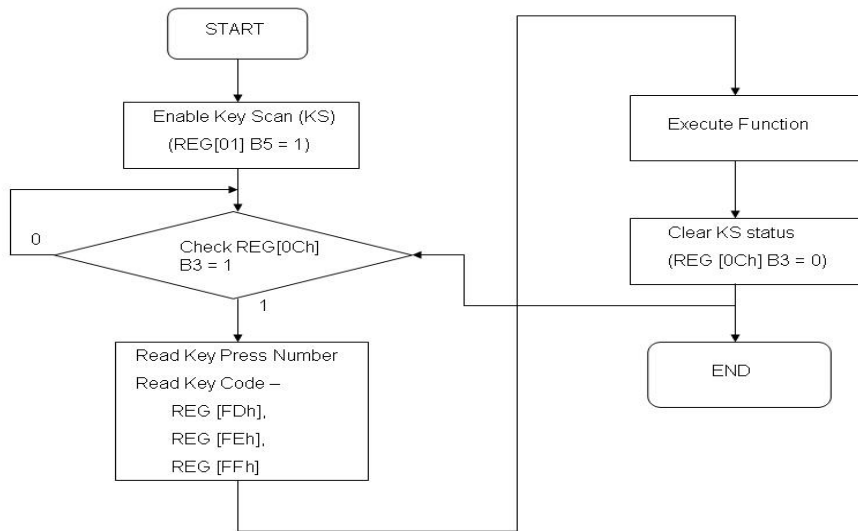
- 1) **Software check method:** to know the key be pressed from keeping check the status of Key-Scan.
- 2) **Hardware check method:** to know the key be pressed from external interrupt signal.

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, RA8871M allows the “Key-stroke wakeup function” for power saving mode. By setting the function on, any legal key-stroke event can wakeup RA8871M from sleep mode. To sense the wakeup event, RA8871M can assert hardware interrupt for MPU which can do software polling from RA8871M.

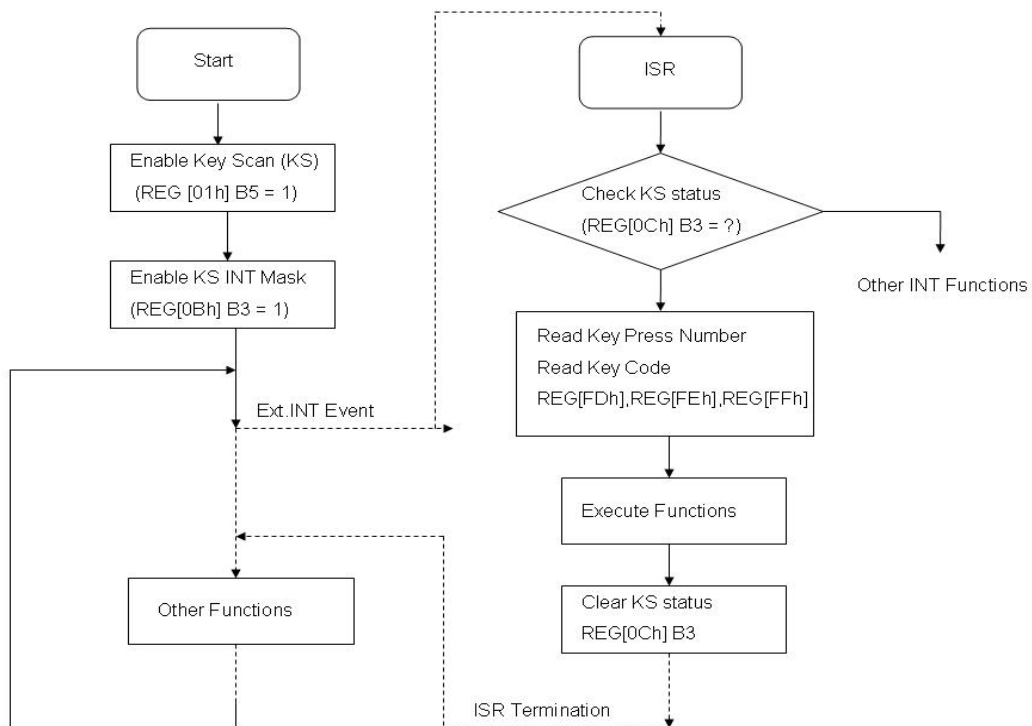
The flowchart of register settings for above applications are shown as following:

1. Software Method



**Figure 8-4 : Key-Scan Flowchart for Software Polling**

2. Hardware Method



**Figure 8-5 : 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:

```
Write_Layer(1); //set memory read/write layer.Reference Page.5~6
```

```
Show_Layer(1);//set LCD display layer. Reference Page.5~6
```

+

```
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(0,0,0x000000,0xffffffff); //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

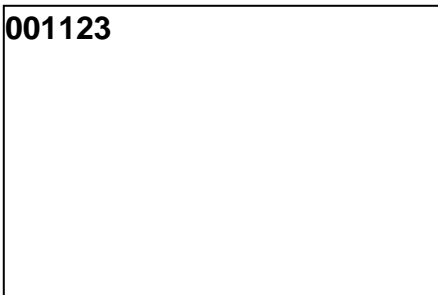
BackGroundColor: Keycode background color = white

**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:**



## Appendix 1 : PLL initialization

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

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

$$10MHz \leq Fin \leq 15MHz \quad \& \quad 10MHz \leq \frac{Fin}{PLLDIVM} \leq 40MHz$$

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

**If  $F_{IN} = 10MHz$**

```

void RA8871_PLL(void)
{
//SCLK Setting :
LCD_RegisterWrite(0x05,0x26);    //PLL Divided by 32
LCD_RegisterWrite(0x06,0x2f);
/* SCLK = {(10MHz/(0+1)) * (0x1f+1)} / 25 = 10 Mhz
REG[05h] SCLK PLL Control Register 1
Bit[5:3] SCLK extra divider
xx1b: divided by 1.
000b: divided by 1.
010b: divided by 2.
100b: divided by 4.
110b: divided by 8.
Bit[2:1] SCLK PLLDIVK[1:0]
SCLK PLL Output divider
00b: divided by 1.
01b: divided by 2.
10b: divided by 4.
11b: divided by 8.
Bit[0] SCLK PLLDIVM
SCLK PLL Pre-driver parameter.
0b: divided by 1.
1b: divided by 2.
REG[06h] SCLK PLL Control Register 2
SCLK PLLDIVN[5:0]
SCLK PLL input parameter, the value should be 1~63. (i.e. value 0 is forbidden).
*/

```



**//MCLK Setting :**

```
LCD_RegisterWrite(0x07,0x06); //PLL Divided by 8
```

```
LCD_RegisterWrite(0x08,0x1f)
```

```
/* MCLK = {(10MHz/(0+1)) * (0x1f+1) } / 23 = 40MHz
```

**REG[07h] MCLK PLL Control Register 1****Bit[2:1]MCLK PLLDIVK**

MCLK PLL Output divider

00b: divided by 1.

01b: divided by 2.

10b: divided by 4.

11b: divided by 8.

**Bit[0]MCLK PLLDIVM**

MCLK PLL Pre-driver parameter.

0b: divided by 1.

1b: divided by 2.

**REG[08h] MCLK PLL Control Register 2****MCLK PLLDIVN[5:0]**

MCLK PLL input parameter, the value should be 1~63. (i.e. value 0 is forbidden).

\*/

**//CCLK Setting :**

```
LCD_RegisterWrite(0x09,0x06); //PLL Divided by 8
```

```
LCD_RegisterWrite(0x0A,0x1f);
```

```
/* CCLK = {(10MHz/(0+1)) * (0x1f+1) } / 22 = 40MHz
```

**REG[09h] CCLK PLL Control Register 1****Bit[2:1]CCLK PLLDIVK**

CCLK PLL Output divider

00b: divided by 1.

01b: divided by 2.

10b: divided by 4.

11b: divided by 8.

**Bit[0]CCLK PLLDIVM**

CCLK PLL Pre-driver parameter.

0b: divided by 1.

1b: divided by 2.

**REG[0Ah] CCLK PLL Control Register 2****CCLK PLLDIVN[5:0]**

CCLK PLL input parameter, the value should be 1~63. (i.e. value 0 is forbidden).

\*/

```
LCD_RegisterWrite(0x01,0x80); // PLL Enable
```

```
Delay_ms(100);
```

```
}
```

## Appendix 2 : SDRAM initialization

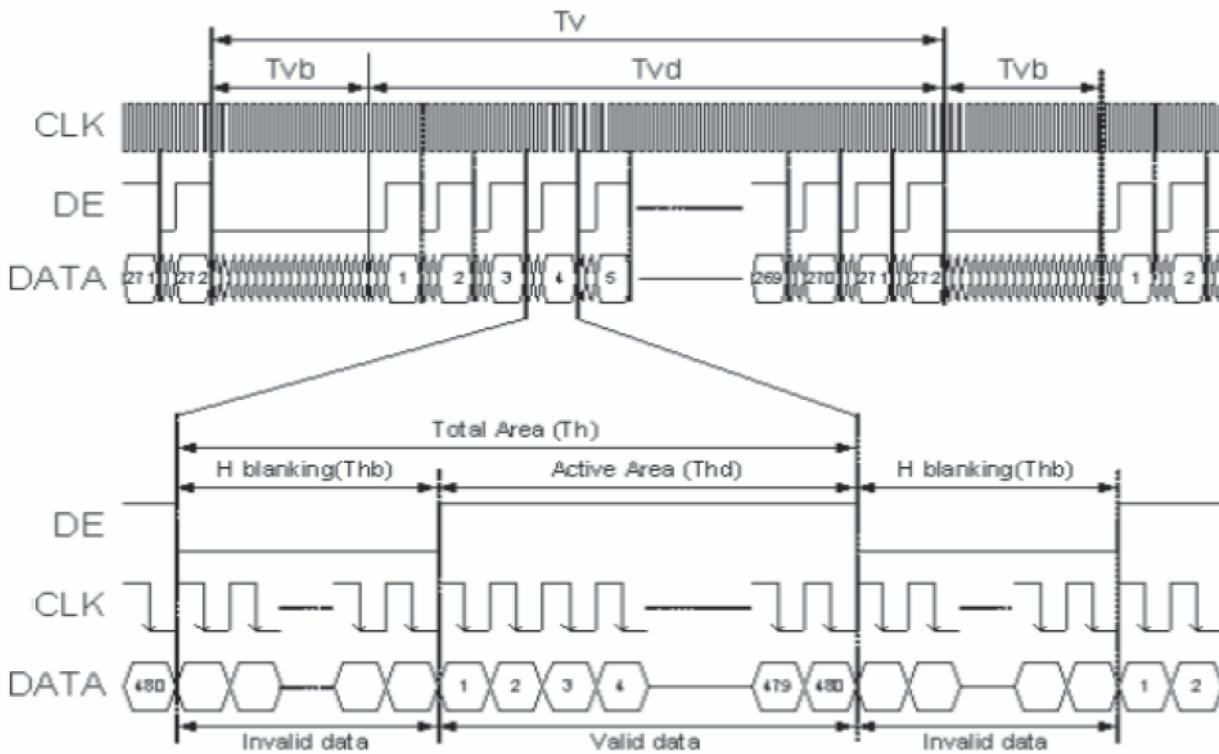
```
void RA8871_SDRAM_initial(void)
{
    unsigned short    Auto_Refresh;
    Auto_Refresh=(64*DRAM_FREQ*1000)/(4096) - 2;
    LCD_RegisterWrite(0xe0,0x08);
    LCD_RegisterWrite(0xe1,0x02);
    LCD_RegisterWrite(0xe2,Auto_Refresh);
    LCD_RegisterWrite(0xe3,Auto_Refresh>>8);
    LCD_RegisterWrite(0xe4,0x01);
    Check_SDRAM_Ready();
}
```

**Appendix 3 : LCD initialization:**

AT043TN25 timing:

Parameter Symbol	Parameter Symbol	Value			Unit
		Min.	Typ.	Max.	
CLK frequency	fclk	7	9	12	MHZ
DEV period time	Tv	277	288	400	H
DEV display area	Tvd	272			H
DEV blanking	Tvb	5	16	128	H
DEH period time	Th	520	525	800	CLK
DEH display area	Thd	480			CLK
DEH blanking	Thb	40	45	320	CLK

**Parallel DE mode RGB input timing table**



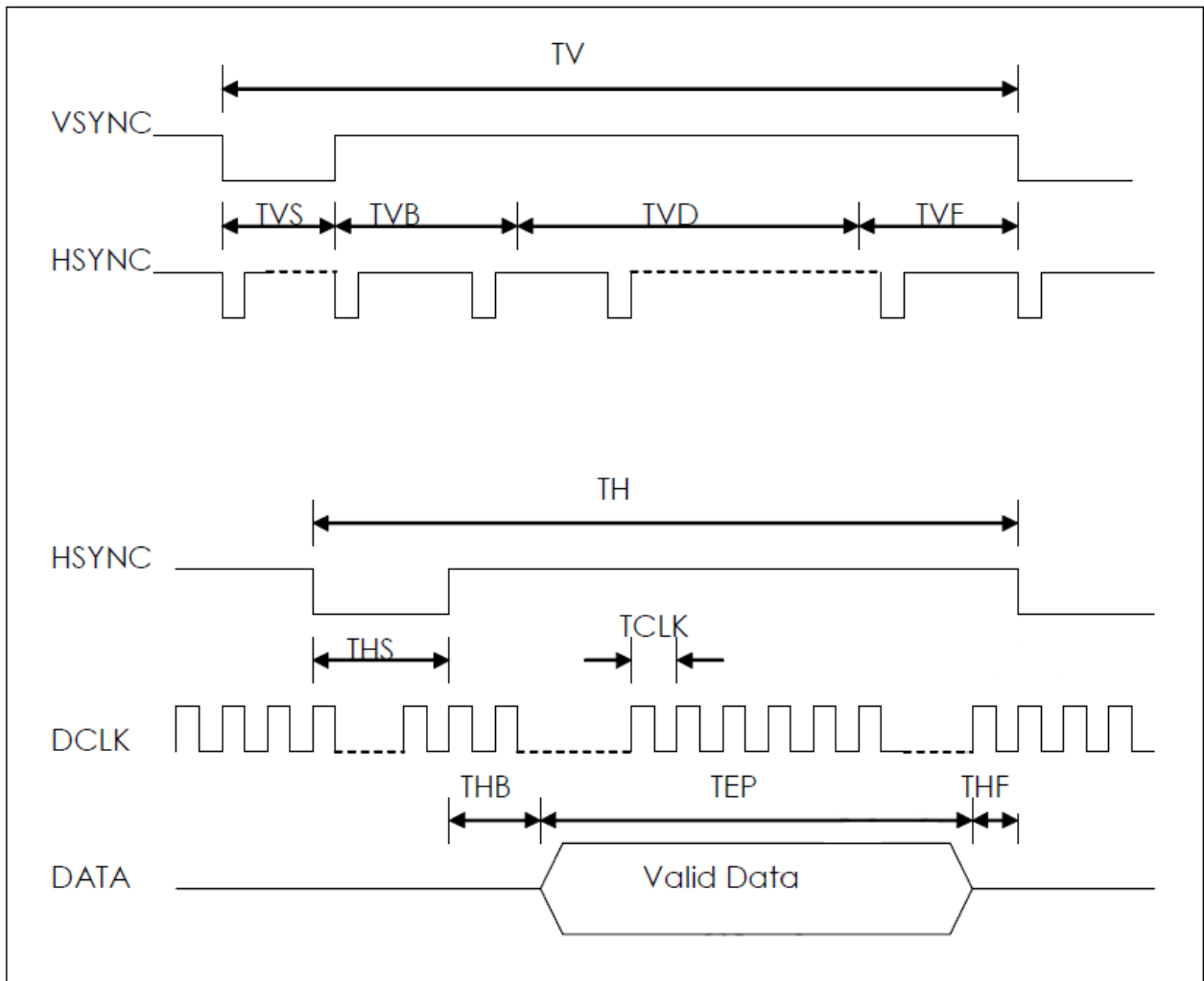
**Timing Diagram**

**AT043TN25 LCD Initial:**

```
Select_LCD_DE_Mode(); // XVSYNC & XHSYNC in idle state.
PCLK_Falling();
VSCAN_T_to_B();
PDATA_Set_RGB();
HSYNC_Low_Active();
VSYNC_Low_Active();
DE_High_Active();
delay_us(100);
/**[14h][15h][1Ah][1Bh]**/
LCD_HorizontalWidth_VerticalHeight(480,272); //LCD Resolution 480x272 °
/**[16h][17h]**/
LCD_Horizontal_Non_Display(38);// 480x272 , HSYNC Back Porch=38 °
/**[18h]**/
LCD_HSYNC_Start_Position(20);// 480x272 HSYNC Front Porch = 20 °
/**[19h]**/
LCD_HSYNC_Pulse_Width(20);// 480x272 , HSYNC Pulse Width=5~30,set 20 °
/**[1Ch][1Dh]**/
LCD_Vertical_Non_Display(4);// 480x272 , VSYNC Back Porch =4 °
/**[1Eh]**/
LCD_VSYNC_Start_Position(15);// 480x272 , VSYNC Front Porch =15 °
/**[1Fh]**/
LCD_VSYNC_Pulse_Width(3);//480x272 , VSYNC Pulse Width =3 °
```

**LQ035NC111 Timing :**

Signal	Item	Symbol	Min	Typ	Max	Unit
Dclk	Frequency	Tosc	-	156	-	ns
	High Time	Tch	-	78	-	ns
	Low Time	Tcl	-	78	-	ns
Data	Setup Time	Tsu	12	-	-	ns
	Hold Time	Thd	12	-	-	ns
Hsync	Period	TH	-	408	-	Tosc
	Pulse Width	THS	5	30	-	Tosc
	Back-Porch	Thb	38	Tosc		Thb
	Display Period	TEP	-	320	-	Tosc
	Hsync-den time	THE	36	68	88	Tsoc
	Front-Porch	Thf	-	20	-	Tosc
Vsync	Period	Tv	-	262	-	TH
	Pulse Width	Tvs	1	3	5	TH
	Back-Porch	Tvb	-	15	-	TH
	Display Period	Tvd	-	240	-	TH
	Front-Porch	Tvf	2	4	-	TH



Horizontal and Vertical Input Timing Diagram

● Digital Parallel RGB

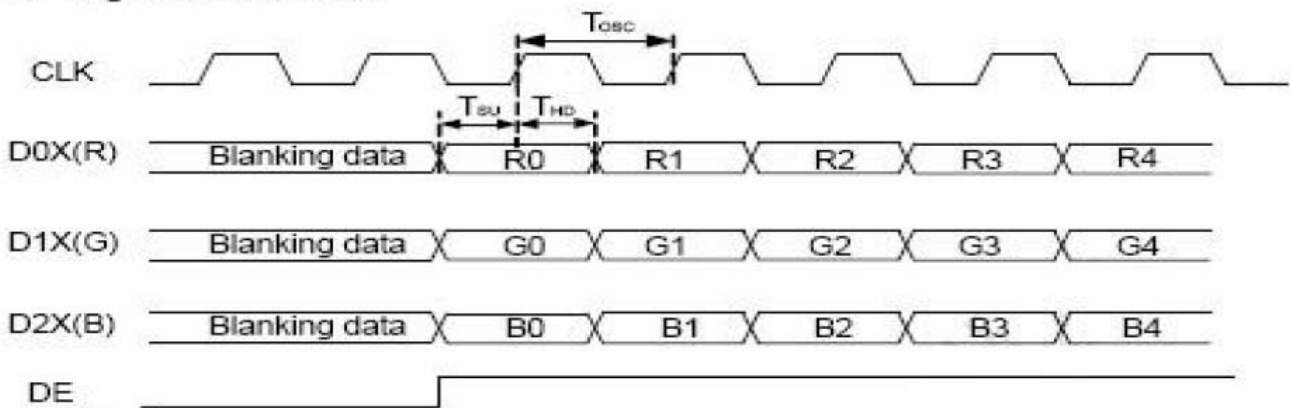


Figure4 CLK, DATA and DE waveforms in Digital Parallel RGB

**LQ035NC111 LCD Initial:**

```
Select_LCD_Sync_Mode(); // Enable XVSYNC, XHSYNC, XDE.
DE_PCLK_Rising();
VSCAN_T_to_B();
PDATA_Set_RGB();
HSYNC_Low_Active();
VSYNC_Low_Active();
DE_Low_Active();
delay_us(100);
/**[14h][15h][1Ah][1Bh]**/
LCD_HorizontalWidth_VerticalHeight(320,240); //LCD Resolution 320x240 ◦
/**[16h][17h]**/
LCD_Horizontal_Non_Display(38); //320x240 ◦ HSYNC Back Porch=38 ◦
/**[18h]**/
LCD_HSYNC_Start_Position(20); //320x240 HSYNC Front Porch = 20 ◦
/**[19h]**/
LCD_HSYNC_Pulse_Width(20); //320x240 ◦ HSYNC Pulse Width=5~30, set 20 ◦
/**[1Ch][1Dh]**/
LCD_Vertical_Non_Display(4); // 320x240 ◦ VSYNC Back Porch =4 ◦
/**[1Eh]**/
LCD_VSYNC_Start_Position(15); // 320x240 ◦ VSYNC Front Porch =15 ◦
/**[1Fh]**/
LCD_VSYNC_Pulse_Width(3); // 320x240 ◦ VSYNC Pulse Width =3 ◦
```

#### Appendix 4 : 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:

`Write_Layer(1);`//memory address switch to layer1.Reference Page.5~6

`Show_Layer(1);`//LCD show layer1 data. Reference Page.5~6

+

//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