Pico e-Paper 2.9

Overview

2.9inch E-Paper E-Ink Display Module For Raspberry Pi Pico, 296×128 Pixels, Black / White, SPI Interface

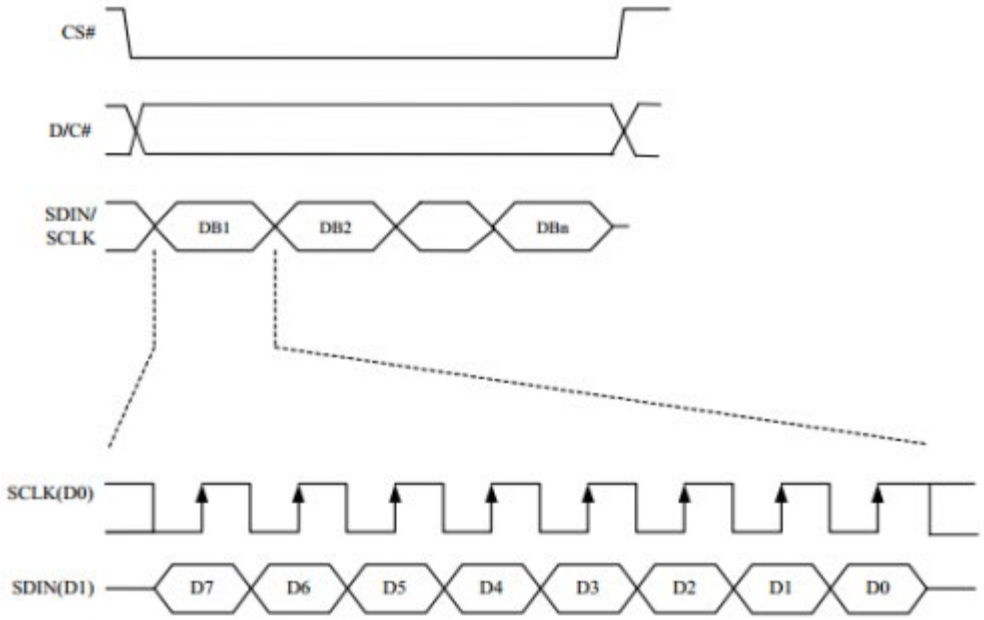
Features

- No backlight keeps displaying last content for a long time even when power down
- Ultra-low power consumption, basically power is only required for refreshing
- SPI interface requires minimal IO pins

Specification

- Operating voltage: 3.3V/5V
- Interface:3-wire SPI, 4-wire SPI
- Outline dimensions: 82.0mm × 38.0mm
- Display size:79.0mm × 36.7mm × 1.05mm
- Dot pitch: 0.138 × 0.138
- Resolution: 296 × 128 pixels
- Display color: Black, white
- Greyscale: 2
- Partial refresh time: 0.3s
- full refresh time: 2s
- Refresh power: 26.4mW(typ.)
- Standby current: <0.01uA(almost none)
- Viewing Angle: >170°

SPI Timing



Note: Different from the traditional SPI protocol, the data line from the slave to the master is hidden since the device only has a display requirement.

- CS is slave chip select, when CS is low, the chip is enabled.
- DC is data/command control pin, when DC = 0, write command, when DC = 1, write data.
- SCLK is the SPI communication clock.
- SDIN is the data line from the master to the slave in SPI communication.

SPI communication has data transfer timing, which is combined by CPHA and CPOL.

1. CPOL determines the level of the serial synchronous clock at an idle state. When CPOL = 0, the level is Low. However, CPOL has little effect on the transmission.
2. CPHA determines whether data is collected at the first clock edge or at the second clock edge of the serial synchronous clock; when CPHL = 0, data is collected at the first clock edge.

- There are 4 SPI communication modes. SPI0 is commonly used, in which CPHL = 0, CPOL = 0.

As you can see from the figure above, data transmission starts at the first falling edge of SCLK, and 8 bits of data are transferred in one clock cycle. Here, SPI0 is in used, and data is transferred by bits, MSB first.

Working Protocol

This product is an E-paper device adopting the image display technology of Microencapsulated Electrophoretic Display, MED. The initial approach is to create tiny spheres, in which the charged color pigments are suspending in the transparent oil and would move depending on the electronic charge. The E-paper screen display patterns

by reflecting the ambient light, so it has no background light requirement. (**Note that the e-Paper cannot support updating directly under sunlight**).

How to define pixels

In a monochrome picture we define the pixels, 0 is black and 1 is white.
White：□, Bit 1
Black：■ ：Bit 0

- The dot in the figure is called a pixel. As we know, 1 and 0 are used to define the color, therefore we can use one bit to define the color of one pixel, and 1 byte = 8pixels
- For example, If we set the first 8 pixels to black and the last 8 pixels to white, we show it by codes, they will be 16 bit as below:

Pixel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Bit	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Color	■	■	■	■	■	■	■	■	□	□	□	□	□	□	□	□

For computer, the data is saved in MSB format:

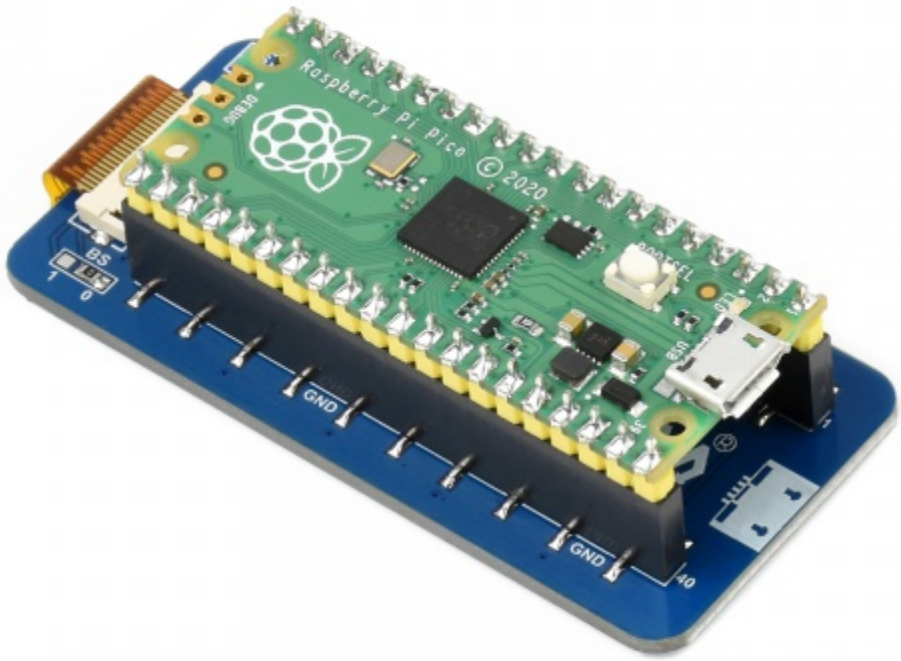
Pixel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Index	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Bit	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
Color	■	■	■	■	■	■	■	■	□	□	□	□	□	□	□	□
Byte	0x00								0xFF							

So we can use two bytes for 16 pixels.

Hardware connection

Please take care of the direction when connecting Pico. A logo of USB port is printed to indicate the directory, you can aslo check the pins.
If you want to connect the board by a 8-pin cable, you can refer to the table below

e-Paper	Pico	Description
VCC	VSYS	Power input
GND	GND	GNd
DIN	GP11	MOSI pin of SPI interface, data transmitted from Master to Slave.
CLK	GP10	SCK pin of SPI interface, clock input
CS	GP9	Chip select pin of SPI interface, Low active
DC	GP8	Data/Command control pin (High: Data; Low: Command)
RST	GP12	Reset pin, low active
BUSY	GP13	Busy pin



Setup Environment

You can refer to the guides of Raspberry Pi: https://www.raspberrypi.org/documentation/pico/getting-started/

Download Demo codes

Open a terminal of Pi and run the following command:

```
sudo apt-get install p7zip-full
cd ~
sudo wget  https://www.waveshare.com/w/upload/2/27/Pico ePaper Code.zip
unzip Pico_ePaper_Code.zip -d Pico_ePaper_Code
cd ~/Pico_ePaper_Code
```

You can also clone the codes from githu

```
sudo git https://github.com/waveshare/Pico_ePaper_Code
cd e-Paper/RaspberryPi_JetsonNano/
```

About the examples

The guides are based on Raspberry Pi.

C codes

The example provided is compatible for several types, you need to modify the main.c file, uncomment the definition according to the actual type of the display you get. For example, if you have the Pico-ePaper-2.13, please modify the main.c file, uncomment line 18(or maybe it is line 19）.

```c
#include "EPD_Test.h"     //Examples

int main(void)
{
    // if(DEV_Module_Init()!=0){
        // return -1;
    // }

    // while(1) {
        // DEV_Delay_ms(10000);
    // }

    // EPD_2in9_V2_test();
    // EPD_2in9bc_test();
    // EPD_2in9b_V3_test();
    // EPD_2in9d_test();

    // EPD_2in13_V2_test();
    // EPD_2in13_V3_test();
    // EPD_2in13bc_test();
    // EPD_2in13b_V3_test();
    // EPD_2in13d_test();

    // DEV_Module_Exit();

    return 0;
}
```

Build the project：

```
cd ~/Pico_ePaper_Code/c
```

Create build folder and add the SDK. ../../pico-sdk is the default path of the SDK, if you save the SDK to other directory, please change it to the actual path.

```
mkdir build
cd build
export PICO_SDK_PATH=../../pico-sdk
```

Run cmake command to generate Makefile file.

```
cmake ..
```

Run the comamnd make to compile the codes.

```
make -j9
```

Code Analysis

C codes

Bottom Hardware Interface

We package the hardware layer for easily porting to the different hardware platforms. DEV_Config.c(.h) in the directory:Pico_ePaper_Code\c\lib\Config

- Data type：

```
#define UBYTE   uint8_t
#define UWORD   uint16 t
#define UDOUBLE uint32_t
```

- Module initialize and exit:

```
void DEV Module Init(void);
void DEV_Module_Exit(void);
Note：
```

```
1.The functions above are used to initialize the display or exit handle.
```

- GPIO Write/Read：

```
void DEV Digital Write(UWORD Pin, UBYTE Value);
UBYTE DEV_Digital_Read(UWORD Pin);
```

- SPI transmit data

```
void DEV_SPI_WriteByte(UBYTE Value);
```

EPD driver

The driver codes of EPD is saved in the directory: Pico_ePaper_Code\c\lib\e-Paper
Open the .h header file, you can check all the functions defined.

- Initialize e-Paper, this function is always used at the beginning and after waking up the display.

```
//2.13inch e-Paper、2.13inch e-Paper  V2、2.13inch e-Paper (D)、2.9inch e-Paper、2.9inch e-Paper (D)
void EPD_xxx_Init(UBYTE Mode); // Mode = 0 fully update, Mode = 1 partial update
//Other types
void EPD_xxx_Init(void);
```

xxx should be changed by the type of e-Paper, For example, if you use 2.13inch e-Paper (D), to fully update, ti shoule be EPD_2IN13D_Init(0) and EPD_2IN13D_Init(1) for partial update;

- Clear: this function is used to clear the display to white.

```
void EPD_xxx_Clear(void);
```

xxx should be changed by the type of e-Paper, For example, if you use 2.9inch e-Paper (D), it should be EPD_2IN9D_Clear();

- Send the image data (one frame) to EPD and display

```
//Bicolor version
void EPD_xxx_Display(UBYTE *Image);
```

```
//Tricolor version
void EPD_xxx_Display(const UBYTE *blackimage, const UBYTE *ryimage);
```

There are several types which are different from others

```
//Paritial update for 2.13inch e-paper (D)、2.9inch e-paper (D)
void EPD_2IN13D_DisplayPart(UBYTE *Image);
void EPD_2IN9D_DisplayPart(UBYTE *Image);
//For 2.13inch e-paper V2, you need to first useEPD_xxx_DisplayPartBaseImage to display a static background
and then partial update by the function EPD_xxx_DisplayPart()
void EPD_2IN13_V2_DisplayPart(UBYTE *Image);
void EPD_2IN13_V2_DisplayPartBaseImage(UBYTE *Image);
```

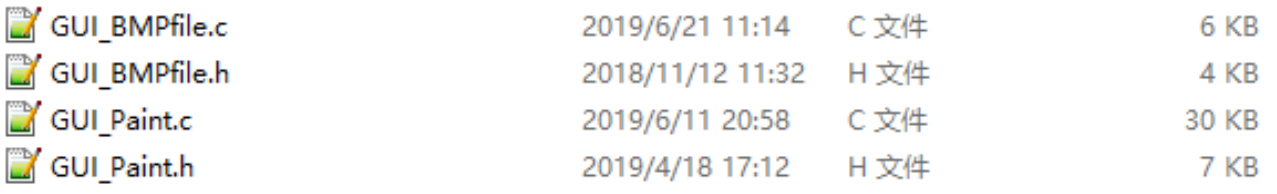
- Enter sleep mode

```
void EPD_xxx_Sleep(void);
```

Note, You should only hardware reset or use initialize function to wake up e-Paper from sleep mode
xxx is the type of e-Paper, for example, if you use 2.13inch e-Paper D, it should be EPD_2IN13D_Sleep();

Application Programming Interface

We provide basic GUI functions for testing, like draw point, line, string, and so on. The GUI function can be found in the directory: RaspberryPi_JetsonNano\c\lib\GUI\GUI_Paint.c(.h)



The fonts used can be found in directory: RaspberryPi_JetsonNano\c\lib\Fonts

font8.c	2018/7/4 17:24	C 文件	18 KB
font12.c	2018/7/4 17:24	C 文件	27 KB
font12CN.c	2018/3/6 15:52	C 文件	6 KB
font16.c	2018/7/4 17:24	C 文件	49 KB
font20.c	2018/7/4 17:24	C 文件	65 KB
font24.c	2018/7/4 17:24	C 文件	97 KB
font24CN.c	2018/3/6 16:02	C 文件	28 KB
fonts.h	2018/10/29 14:04	H 文件	4 KB

- Create a new image, you can set the image name, width, height, rotate angle and color.

```
void Paint NewImage(UBYTE *image, UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
Parameters:
          image : Name of the image buffer, this is a pointer;
          Width : Width of the image;
          Height: Height of the image;
          Rotate: Rotate angle of the Image;
          Color : The initial color of the image;
```

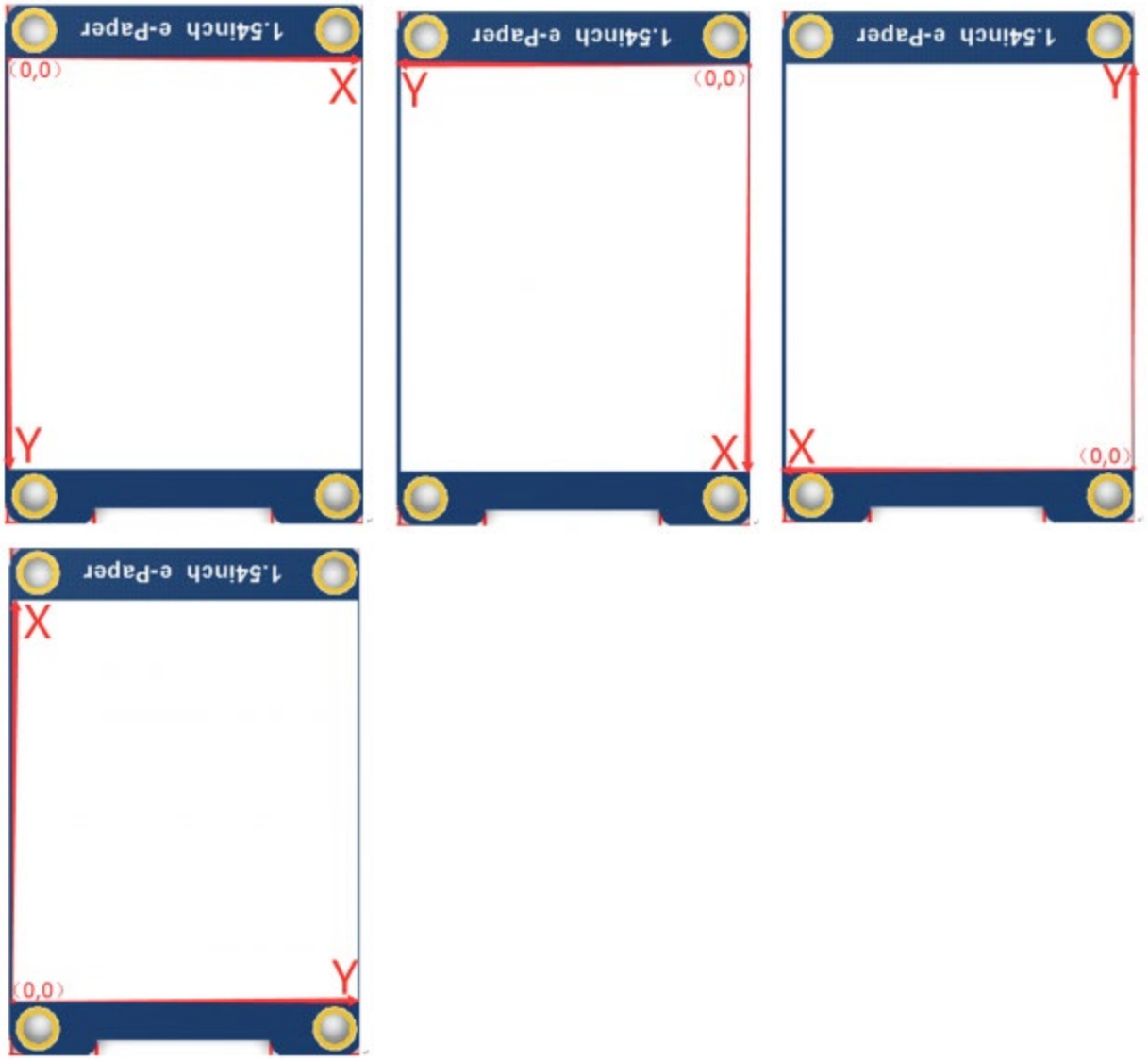
- Select image buffer: You can create multiple image buffers at the same time and select the certain one and drawing by this function.

```
void Paint SelectImage(UBYTE *image)
Parameters:
          image: The name of the image buffer, this is a pointer;
```

- Rotate image: You need to set the rotate angle of the image, this function should be used after Paint_SelectImage(). The angle can 0, 90, 180, 270

```
void Paint_SetRotate(UWORD Rotate)
Parameters:
          Rotate: Rotate angle of the image, the parameter can be ROTATE_0, ROTATE_90, ROTATE_180,
ROTATE_270.
```

【Note】Afer rotating, the place of the first pixel is different, we take 1.54inch e-paper as example



- Image mirror: This function is used to set the image mirror.

```
void Paint SetMirroring(UBYTE mirror)
Parameters:
          mirror: Mirror type if the image, the parameter can be MIRROR_NONE、MIRROR_HORIZONTAL、
MIRROR_VERTICAL、MIRROR_ORIGIN.
```

- Set the position and color of pixels: This is the basic function of GUI, it is used to set the position and color of a pixel in the buffer.

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
Parameters:
          Xpoint: The X-axis value of the point in the image buffer
          Ypoint: The Y-axis value of the point in the image buffer
          Color : The color of the point
```

- Clear display: To set the color of the image, this function always be used to clear the display.

```
void Paint_Clear(UWORD Color)
Parameters:
         Color: The color of the image
```

- Color of the windows: This function is used to set the color of windows, it always used for updating partial areas like displaying a clock.

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color)
Parameters:
         Xpoint: The X-axis value of the start point in the image buffer
         Ypoint: The Y-axis value of the start point in the image buffer
         Xend: The X-axis value of the end point in the image buffer
         Yend: The Y-axis value of the end point in the image buffer
         Color: The color of the windows
```

- Draw point: Draw a point at the position （Xpoint, Ypoint） of image buffer, you can configure the color, size, and the style.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, DOT_STYLE Dot_Style)
Parameters:
         Xpoint: X-axis value of the point.
         Ypoint: Y-axis value of the point.
         Color: Color of the point
         Dot Pixel: Size of the point, 8 sizes are available.
                   typedef enum {
                            DOT_PIXEL_1X1  = 1,         // 1 x 1
                            DOT PIXEL 2X2  ,            // 2 X 2
                            DOT_PIXEL_3X3  ,            // 3 X 3
                            DOT PIXEL 4X4  ,            // 4 X 4
                            DOT PIXEL 5X5  ,            // 5 X 5
                            DOT_PIXEL_6X6  ,            // 6 X 6
                            DOT PIXEL 7X7  ,            // 7 X 7
                            DOT_PIXEL_8X8  ,            // 8 X 8
                  } DOT_PIXEL;
         Dot Style: Style of the point, it define the extednded mode of the point.
                  typedef enum {
                     DOT FILL AROUND  = 1,
                     DOT_FILL_RIGHTUP,
                  } DOT_STYLE;
```

- Draw line: Draw a lin from (Xstart, Ystart) to (Xend, Yend) in image buffer, you can configure the color, width and the style.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color, LINE_STYLE Line_Style ,
LINE STYLE Line Style)
Parameters:
          Xstart: Xstart of the line
          Ystart: Ystart of the line
          Xend: Xend of the line
          Yend: Yend of the line
          Color: Color of the line
          Line width: Width of the line, 8 sizes are available.
                    typedef enum {
                              DOT_PIXEL_1X1  = 1,         // 1 x 1
                              DOT PIXEL 2X2  ,            // 2 X 2
                              DOT_PIXEL_3X3  ,            // 3 X 3
                              DOT_PIXEL_4X4  ,            // 4 X 4
                              DOT PIXEL 5X5  ,            // 5 X 5
                              DOT_PIXEL_6X6  ,            // 6 X 6
                              DOT PIXEL 7X7  ,            // 7 X 7
                              DOT_PIXEL_8X8  ,            // 8 X 8
                    } DOT_PIXEL;
           Line Style: Style of the line, Solid or Dotted.
                    typedef enum {
                              LINE_STYLE_SOLID = 0,
                              LINE STYLE DOTTED,
                    } LINE_STYLE;
```

- Draw rectangle: Draw a rectangle from (Xstart, Ystart) to (Xend, Yend) , you can configure the color, width, and style.

```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color, DOT_PIXEL
Line width, DRAW FILL Draw Fill)
Parameters:
          Xstart: Xstart of the rectangle.
          Ystart: Ystart of the rectangle.
          Xend: Xend of the rectangle.
          Yend: Yend of the rectangle.
          Color: Color of the rectangle
          Line_width: The width of the edges. 8 sizes are available.
                    typedef enum {
                              DOT_PIXEL_1X1  = 1,         // 1 x 1
                              DOT_PIXEL_2X2  ,            // 2 X 2
                              DOT PIXEL 3X3  ,            // 3 X 3
                              DOT_PIXEL_4X4  ,            // 4 X 4
                              DOT_PIXEL_5X5  ,            // 5 X 5
```

```
                    DOT PIXEL 6X6  ,          // 6 X 6
                    DOT_PIXEL_7X7  ,          // 7 X 7
                    DOT_PIXEL_8X8  ,          // 8 X 8
          } DOT PIXEL;
     Draw_Fill: Style of the rectangle, empty or filled.
          typedef enum {
                    DRAW FILL EMPTY = 0,
                    DRAW_FILL_FULL,
          } DRAW_FILL;
```

- Draw circle: Draw a circle in image buffer, use (X_Center Y_Center) as center and Radius as radius. You can configure the color, width of line and the style of circle.

```
void Paint DrawCircle(UWORD X Center, UWORD Y Center, UWORD Radius, UWORD Color, DOT PIXEL Line width,
DRAW_FILL Draw_Fill)
Parameters:
     X Center: X-axis of center
     Y_Center: Y-axis of center
     Radius：Radius of circle
     Color: Color of the circle
     Line_width: The width of arc, 8 sizes are available.
          typedef enum {
                    DOT_PIXEL_1X1  = 1,       // 1 x 1
                    DOT_PIXEL_2X2  ,          // 2 X 2
                    DOT PIXEL 3X3  ,          // 3 X 3
                    DOT_PIXEL_4X4  ,          // 4 X 4
                    DOT PIXEL 5X5  ,          // 5 X 5
                    DOT_PIXEL_6X6  ,          // 6 X 6
                    DOT_PIXEL_7X7  ,          // 7 X 7
                    DOT PIXEL 8X8  ,          // 8 X 8
          } DOT_PIXEL;
     Draw_Fill: Style of the circle: empty or filled.
          typedef enum {
                    DRAW_FILL_EMPTY = 0,
                    DRAW_FILL_FULL,
          } DRAW_FILL;
```

- Show Ascii character: Show a characeter in (Xstart, Ystart) position, you can configure the font, foreground and the background.

```
void Paint DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii Char, sFONT* Font, UWORD Color Foreground,
UWORD Color_Background)
Parameters:
     Xstart: Xstart of the character
     Ystart: Ystart of the character
```

```
        Ascii Char : Ascii char
        Font:  five fonts are avaialble :
                font8 : 5*8
                font12 : 7*12
                font16 : 11*16
                font20 : 14*20
                font24 : 17*24
        Color_Foreground: foreground color
        Color_Background: background color
```

- Draw string: Draw string at (Xstart Ystart) , you can configure the fonts, foreground and the background

```
void Paint DrawString EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* Font, UWORD
Color_Foreground, UWORD Color_Background)
Parameters:
        Xstart: Xstart of the string
        Ystart: Ystart of the string
        pString : String
        Font: five fonts are available:
                font8 : 5*8
                font12 : 7*12
                font16 : 11*16
                font20 : 14*20
                font24 : 17*24
        Color_Foreground: foreground color
        Color_Background: background color
```

- Draw Chiness string: Draw Chinese string at (Xstart Ystart) of image buffer. You can configure fonts (GB2312), foreground and the background.

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* font, UWORD
Color Foreground, UWORD Color Background)
Parameters:
        Xstart: Xstart of string
        Ystart: Ystart of string
        pString : string
        Font: GB2312 fonts, two fonts are available
                font12CN : ascii 11*21, Chinese 16*21
                font24CN : ascii 24*41, Chinese 32*41
        Color Foreground: Foreground color
        Color_Background: Background color
```

- Draw number: Draw numbers at (Xstart Ystart) of image buffer. You can select font, foreground and the background.

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, int32_t Nummber, sFONT* Font, UWORD Color_Foreground, UWORD
Color Background)
Parameters:
          Xstart: Xstart of numbers
          Ystart: Ystart of numbers
          Nummber : numbers displayed. It support int type and 2147483647 are the maximum supported
          Font: Ascii fonts, five fonts are available:
                    font8 : 5*8
                    font12 : 7*12
                    font16 : 11*16
                    font20 : 14*20
                    font24 : 17*24
          Color_Foreground: foreground
          Color_Background: background
```

- Display time: Display time at (Xstart Ystart) of image buffer, you can configure fonts, foreground, and background.

  This function is used for partial update. Note that some of the e-Paper doesn't support partial update and you cannot use partial update all the time,which will cost ghots problem and destroy the display.

```
void Paint DrawTime(UWORD Xstart, UWORD Ystart, PAINT TIME *pTime, sFONT* Font, UWORD Color Background, UWORD
Color_Foreground)
Parameters:
          Xstart: Xstart of time
          Ystart: Ystart of time
          pTime : Structure of time
          Font: Ascii font, five fonts are avaialble
                    font8 : 5*8
                    font12 : 7*12
                    font16 : 11*16
                    font20 : 14*20
                    font24 : 17*24
          Color_Foreground: foreground
          Color_Background: background
```