5.83inch e-Paper HAT (B) User Manual

OVERVIEW

- This is an E-Ink display HAT for Raspberry Pi, 5.83inch, 600x448 resolution, with embedded controller, communicating via SPI interface, supports red, black, and wite three-color display.
- Due to the advantages like ultra-low power consumption, wide viewing angle, clear display without electricity, it is an ideal choice for applications such as shelf label, industrial instrument, and so on.

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
- Compatible with Raspberry Pi Zero/Zero W/Zero WH/2B/3B/3B+
- SPI interface, for connecting with other controller boards like Raspberry/Arduino/Nucleo, etc.

- Comes with development resources and manual (examples for Raspberry Pi /Arduino/STM32)

SPECIFICATIONS

Operating voltage:	3.3V
Interface:	SPI
Outline Dimension:	125.40mm ×99.50mm × 1.18mm
Display size:	118.80mm × 88.26mm
Dot pitch:	0.198 × 0.197
Resolution:	600 × 448
Display color:	Black, White, Red
Grey level:	2
Full refresh:	3.5s
Refresh power:	26.4mW(typ.)
Standby power:	<0.017mW
Viewing angle:	>170°

INTERFACE

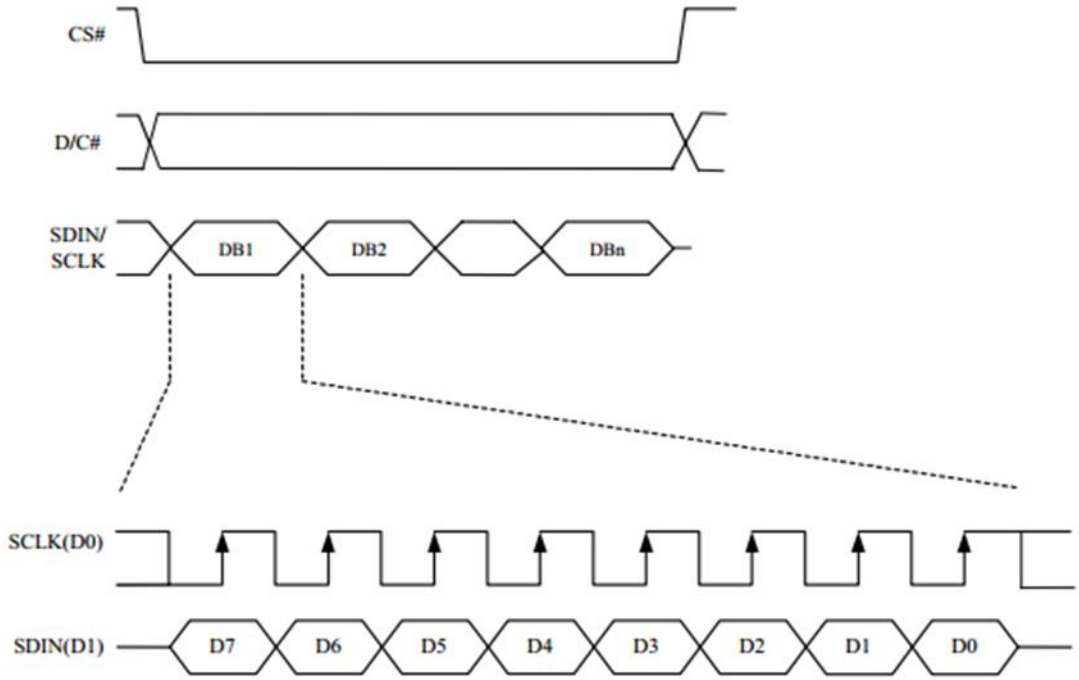
VCC	3.3V
GND	GND
DIN	SPI MOSI pin
CLK	SPI CLK pin
CS	SPI chip selection, low active
DC	Data/Command selection (high for data, lower for command
RST	External reset, low active
BUSY	Busy status output, low active

WORKING PROTOCOL

INTRODUCTION

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. Under sunshine, the E-paper screen still has high visibility with a wide viewing angle of 180 degree. It is the ideal choice for E-reading.

COMMUNICATION PROTOCOL



CS: Slave chip select, when CS is low, the chip is enabled.

DC: Data/command control pin, when DC=0. Write command, when DC=1, write data.

SCLK: SPI communication clock

SDIN: 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 idle state, when CPOL =0, the level is Low. However, CPOL has little effect to the transmission.

2) CPHA determines whether data is collected at the first clock edge or at the second clock edge of serial synchronous clock; when CPHA=0, data is collected at the first clock edge, otherwise, it is collected at the second clock edge.

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

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

HOW TO USE

On the wiki page, demo code are provide, which are based on Raspberry Pi, Arduino and STM32.

WORKING WITH RASPBERRY PI

For Raspberry Pi, there are three demo codes. One is based on BCM2835 libraries, one on wiringPi and one on Python.

To use these demo code, you require to necessary libraries. For how to install the libraries, please refer to: Libraries_Installation_for_RPi

HARDWARE CONNECTION

Here is the connection between Raspberry Pi 3 B/B+ and e-Paper. (BCM2835 code number)

e-Paper	Pi 3 B/B+
3.3V	3.3V
GND	GND
DIN	MOSI
CLK	SCLK
CS	CE0
DC	25(BCM)
RST	17(BCM)
BUSY	24(BCM)

EXPECTED RESULT

After the corresponding libraries installed, you can copy the relative programs into your Raspberry Pi, and then enter the directory.

- **BCM2835:** Execute the command: **make**, to compile the code and generate a file epd. Execute the command: **sudo ./epd**, the program will run.
- **WiringPi:** Execute the command: **make**, to compile the code and generate a file epd. Execute the command: **sudo ./epd**, the program will run.
- **Python:** Execute the command: **sudo python main.py**

After execute the command above, image will be displayed on the screen.

Note: The refresh speed of this module is slow (about 14s), and it will flicker for several times during refreshing. Please be patient.

WORKIGN WITH ARDUINO

HARDWARE CONNECTION

This connection is for Arduino UNO. If you use other Arduino board like mega2560 which pins have some different, you need to change the connection according to real board.

e-Paper	Arduino UNO
3.3V	3.3V
GND	GND
DIN	D11
CLK	D13
CS	D10
DC	D9
RST	D8
BUSY	D7

EXPECTED RESULT

- Copy the libraries file of Arduino demo code to the libraries folder which is under the installation directory of Arduino IDE, it is C:\users\username\documents\arduino\libraries. Please choose the real path you installed.
- Open the project and upload it to board.
- E-Paper will display the image as code.

Note: The refresh speed of this module is slow (about 14s), and it will flicker for several times during refreshing. Please be patient.

WORKING WITH STM32

- Board: Open103Z (STM32F103ZE)
- Libraries: HAL
- Software: Compiled with Keil v5

HARDWARE CONNECTION

e-Paper	STM32F103ZE
3.3V	3.3V
GND	GND
DIN	PA7(MOSI)
CLK	PA5(SCK)
CS	PA4
BUSY	PA3
DC	PA2
RST	PA1

EXPECTED RESULT

- Open the Keil project (epd-demo.uvprojx) which is under the MDK-ARM directory.
- Then click **Build** to compile the project.
- Flash to board by clicking **Download**

  After reset, the e-paper will display image.

Note: The refresh speed of this module is slow (about 14s), and it will flicker for several times during refreshing. Please be patient.

CODE ANALYSIS

Here, we will analyze the driving code and take the demos for Raspberry Pi based on WiringPi library as examples.

HARDWARE INTERFACE FUNCTION(EPDIF)

The functions of driver code like **DigitalWrite**, **DigitalRead**, **SendCommand**, **SendData**, **DelayMs** call the interface functions which are provided by hardware device (epdif.h, epdif.cpp). To respectively implements the function that Control IO level, Read IO level, Send SPI Command, Send SPI Data and Delay for Millisecond. If you want to port the demo code, you need to implement all the interfaces of epdif (e-paper display interface) according to the corresponding hardware device.

Note that Raspberry Pi uses hardware chip select while transmitting SPI data, so we needn't set the CS pin to LOW before transmitting data, and the code will set it automatically while transmitting. However, for Arduino and STM32, etc. you need to explicitly set the CS pin Low by coding to start the SPI transmission of module.

SENDCOMMAND, SENDDATA

SendCommand and SendData are functions which are used to send command and data to module separately.

Difference: When send command, the send command function will set D/C pin to Low, with this, the SPI data send to module will execute as command. When send data, the send data function will set D/C pin to High, then the SPI data sent will take as common data. Usually, parameters and image data will follow the commands.

RESET

Module will reset if RST pin is Low. After power on or the module is in sleep mode, you can low this pin to restart module. After restart module, you should also use **Init** function to initial the module, otherwise it couldn't work normally.

INIT

After powering on the module, initialization function (**Init**) will configure the parameters of module. It can also wake up module from sleep mode. Process of initialization: reset --> power setting --> panel setting --> booster soft start --> power on --> PLL control --> temperature calibration --> VCOM and data interval setting --> TCON setting --> TCON resolution --> VCM DC setting.

For more about the parameters setting, please refer to the datasheet.

SETLUT

Look-up table is stored in module, the table is provided by us and it may different because of different production batch. If the table change, we will update demo code as soon as possible.

DISPLAYFRAME

DisplayFrame is used to send a frame to the module, and the screen will refresh and display it.

Process: Send command data start transmission 1 --> Send data of an image to display --> Refresh the screen.

The image data: 4bits = 1 pixel. It could only display black, white and red. 0000b stands for a black pixel, 0011b stand for white pixel and 0100b stand for red pixel.

For example:

0x00：2 pixels■■

0x03：2 pixels■□

0x30：2 pixels□■

0x33：2 pixels□□

0x04：2 pixels■■

0x43：2 pixels■□

0x66：the state of pixels is uncertain

Note: This display **cannot** support partial refresh and the refresh speed of this module is slow (about 14s), and it will flicker for several times during refreshing. Please be patient.

SLEEP

Module can be set in sleep mode to reduce the consumption.

Process: power off --> deep sleep

If you want to wake up the module from sleep mode, you need to give a LOW pulse to RST pin. Then maybe you need to reconfigure the parameter of power (According to the batches, some of them need to reconfigure, some needn't). If you want to wake up module,

you had better use the Init function instead of Reset. Reset function and relative commands will be executed while executing the Init function.

DISPLAY AN IMAGE

There are two ways to display a picture on module.

Display directly: Read the data of pictures with library functions and decode it. Then convert it to arrays and send to module. About how to implement it, you can refer to the python examples of Raspberry Pi. (The C demo doesn't display pictures directly).

Display indirectly: Converting pictures to relative arrays on PC and save as c file. Then you can use the c file on your project. Here we will talk about how to convert a picture to array.

1) Open a picture with drawing tool comes with Windows system. Reset the size to 600x448.

2) As the module could only display three colors. The demo code uses the method that separate B/W/R picture to two, one is only has black and white part and another have red and white part, 33600 bytes per picture. In this case we need to convert the picture to monochrome bitmap before converting it to array. There is a monochrome bitmap on examples pack for demonstration (raspberrypi/python/monocolor.bmp)
   process: File --> Save as --> BMP picture--> Monochrome Bitmap

3) Separate red and black pixel: You can use PS, Select-->Color Range. Choose the red part and delete, then save the picture to get B/W picture. With the same operate to get the red one.

4) Use Image2Lcd.exe software to convert picture to C file. Open picture on software, and configure:
   Output data type: C language array
   Scanning mode: vertical scanning
   Output gray: 4-level-grayscale
   Maximum width and height: 640 and 384
   Include the data of Image Header: Don't check
   Inverse color: Check (Check: the white on image will be inversed to 1, and black is inversed to 0)

5) Click "Save", to generate .c file. Copy the corresponding array into your project, and you can display picture by calling this array.

Note:

The module decodes the image data as: 4 bits = 1 pixel, but due to Gray scale unsupported, a monochrome array (1 bit = 1 pixel) is enough. The code will send 0x3 if it read set bit of array, and it will send 0x0 if it read reset bit. For example: code get the byte 0xAA, it will send 0x30303030 in fact.