Bitmapping a BMP/PNG Image to a Character Array and Displaying the Image on Pervasive Display's - 2.66" E ink Display - Aurora -25(V430)

Scope

This configuration has been tested successfully against

ESP32 Devkit V1 for the screens up to 2.66”

This configuration may also suit other boards with the Arduino form-factor,
with the size of the screen adapted to the available SRAM of the MCU.

Software

If not already done, follow the instructions ***HERE*** to get the correct
setup required to display an image on the 2.66” E ink display from Pervasive Display.

Notes

After having completed the setup from the linked document in the Software section,
you should be able to display the images found in the example.ino files in Arduino IDE.
Next, we will now configure our project to be able to display custom images through
bit mapping images to C/C++ character arrays.

Step 1:

Download the github repository located here. After downloading the files, open up the
EPD_sub_5_inch.ino file. Start off by declaring your board much like in the guide for connecting
the EXT3 board. Also you will need to correctly declare valid pins for your board. (See below)
Then run the program to make sure that the project is flashing correctly to ESP32.
The program should display an image for around 5 seconds, and then display a second image.

```
16  #define EPD_154 0 //1.54 inch PDi EPD (iTC)
17  #define EPD_213 1 //2.13 inch PDi EPD (iTC)
18  #define EPD_266 2 //2.66 inch PDi EPD (iTC)
19  #define EPD_271 3 //2.71 inch PDi EPD (iTC)
20  #define EPD_287 4 //2.87 inch PDi EPD (iTC)
21  #define EPD_370 5 //3.70 inch PDi EPD (iTC)
22  #define EPD_420 6 //4.20 inch PDi EPD (iTC)
23  #define EPD_437 7 //4.37 inch PDi EPD (iTC)
24
25  long image_data_size[] = { 2888, 2756, 5624, 5808, 4736, 12480, 15000, 10560 };  //followed by the index above
26
27  // Please define which size of PDi's EPD you will work with from one of the definitions above
28  #define PDI_EPD_Size EPD_266
29
```

For the code below, you will have to make sure that the pins are aligned correctly with the configuration that you are using for your specific project.

```
#else   // Valid pins for Arduino board, like M0 Pro
#define SCL_PIN 18      // EXT3 board J4 pin 2 SCK
#define BUSY_PIN 15      // EXT3 board J4 pin 3 BUSY
#define DC_PIN 2       // EXT3 board J4 pin 4 D/C
#define RESET_PIN 4    // EXT3 board J4 pin 5 RST (RESET)
#define SDA_PIN 23      // EXT3 board J4 pin 7 MOSI
#define CS_PIN 21       // EXT3 board J4 pin 9 ECSM (EPD CS Master)
//EXT3 board J4 pin 1 connects 3V3
//EXT3 board J4 pin 10 connects GND
```

Next, locate this chunk of code:

```
#elif(PDI_EPD_Size==2)   //2.66"
    #include "image_data\2.66\image_266_296x152_BW.c"
```

Step 2:

Directly underneath this code, delete/comment out the following lines.

#include "image_data\2.66\image_266_296x152_BWR.c"
#define BWR_blackBuffer (uint8_t *) & image_266_296x152_BWR_blackBuffer
#define BWR_redBuffer (uint8_t *) & image_266_296x152_BWR_redBuffer

So now your code should look like this:

```
#elif(PDI_EPD_Size==2)   //2.66"
    #include "image_data\2.66\image_266_296x152_BW.c"

    #define BW_monoBuffer        (uint8_t *) & image_266_296x152_BW_mono
    #define BW_0x00Buffer        (uint8_t *) & image_266_296x152_BW_0x00
```

Step 3:

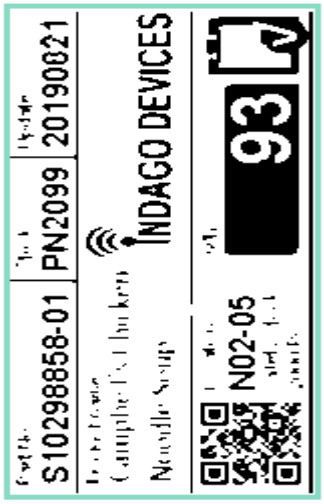
Next, open up the file in the directory above, image_266_296x152_BW.c
This is the file that contains the arrays for whichever image you are going to be displaying on your screen. First, delete the code that sends the second image:

```
// Send 2nd image data for black, white and red colors
sendIndexData( 0x10, BWR_blackBuffer, image_data_size[PDI_EPD_Size] );  //First frame: black frame where 1=black and 0=white or color pixel
sendIndexData( 0x13, BWR_redBuffer, image_data_size[PDI_EPD_Size] );    //Second frame: color frame where 1=color and 0=white or black pixel
sendIndexData( 0x12, &register_data[0], 1 );    //Display Refresh
while( digitalRead( BUSY_PIN ) != HIGH );
```

This will make it easier for debugging because you will only be focused on being able to Send one image to your board. Once you can correctly send a desired image to your board, you can go back and add the code to send multiple images to the board.

Step 4:

Within the image_266_296x152_BW.c file, delete all the characters within the **image_266_296x152_BW_mono** array, but do not delete the array declaration, because once we correctly bitmap an image, we will paste the converted image into the array declaration. Now, we will take a black and white image of your choice to be updated on the display. I used the image found here. Since we already have the array for this image, we will open this image up in an editor (Photoshop,Gimp,etc.) and change the image to your liking. Then, rotate the image within the editor so that it has this orientation



Save the image as a .bmp or .png file. Depending on the
photo editor that you are using, one of these types of files will be needed for the next step. While doing the next step,
If the .bmp does not work, try it with the .png instead, and vice versa.

Step 5:

Open up the website found here. Select the image that you edited in the previous step and upload it to the application.
Within the image settings, make sure you have the canvas size set appropriately(for the 2.66" display, 152x296).
Also you will most likely need to invert the image colors. You should now see your image in the preview like this

Next, go the output portion of the application and make sure that output format is set to Arduino Code, and the Draw mode is set
To Horizontal - 1 bit per pixel. Now generate the code. Take the array that was generated and paste it into the array that we talked
about in step 4.

Step 6:

After saving the array, now run the EPD_sub_5_inch.ino program and the display should be outputting the image that you created
within your editor.

Desired Outcome

Here, you will see a basic program that runs through 4 images.

To get over the air flashing of your ESP32, follow the guide found here.

Lastly, here are some useful links concerning the ESP32/Epaper display.

Options

Large screens 9.7" and 12.2" require panelCSS to be defined and connected.

If the second memory is populated on the EXT3 board, lashCSS needs to be defined and connected. The library will keep lashCSS HIGH to avoid any interferences with the other devices connected to the SPI bus.

Created 10/12/2021 Joe Harrison, NCSU