# Application Note

# BRT_AN_074

# BT81x Simple Color Picker for RGB LED control

**Version 1.0**

**Issue Date: 19-04-2024**

This application note provides several examples of implementing a colour picker using the EVE family of devices. The user can touch an area of an image (often a colour gradient) to select the colour. A BT81x series EVE device is used as the display controller, with an FT9xx MCU as the host controller. The colour selected by the user is displayed on the screen as well as on an RGB LED strip, using the PWM feature of the FT9xx series MCU.



**Bridgetek Pte Ltd (BRT Chip)**
1 Tai Seng Avenue, Tower A, #03-05, Singapore 536464
Tel: +65 6547 4827
Web Site: http://www.brtchip.com

# Table of Contents

# 1 Introduction

## 1.1 Overview

This application note provides several examples of implementing a colour picker or colour wheel using the EVE family of devices.

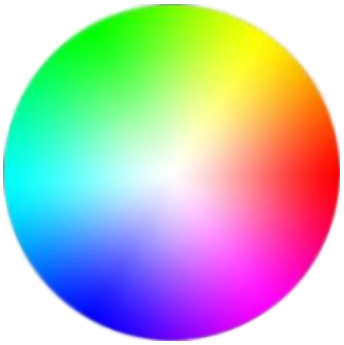


**Figure 1 – Example of a colour picker wheel**

Colour pickers allow the user to select a colour by touching a particular pixel of an image displayed on the screen. They help to make selection of the colour within an application friendly and intuitive for the end user. They are often used in applications such as lighting or where the application designer wants to allow the user to customise colours of on-screen items with a wide selection of colours. In addition, the technique used also allows applications where the end user can test the colour of parts of an image such as a photograph.

The code was developed using a Bridgetek ME817EV with 7" capacitive screen and MM900EV1B module with UMFTPD2A programmer. The selected colour is displayed on the screen as well as on an RGB LED strip, using the PWM feature of the FT9xx series MCU.

## 1.2 Scope

This application note is intended to demonstrate simple techniques for implementing colour pickers on the Bridgetek BT81x series devices. It focuses on the colour picker operation and assumes that the user is familiar with the basic principles of developing touch-enabled applications for EVE and with programming the FT9xx series MCUs.

## 1.3 Compatibility

This application note is based around the Bridgetek EVE BT81x series as it stores the colour picker images in Flash. However, similar techniques could be used on the FT81x / BT88x series provided that the host MCU has sufficient flash to hold the colour picker image, and to load this into the FT81x / BT88x upon power-up. The colour picker technique and code can be used on capacitive or resistive screens, but the multi-touch example is only compatible with capacitive screens (such as BT815 or BT817). Due to the size of the rectangular colour picker and images, the screen layout is best suited to 7" landscape LCD panels but could be easily changed to other screen sizes.

The code example is provided for the FT90x series of MCUs but can also be used on other host MCUs. It can be used on other MCUs provided they have a framework/library code for use with the EVE series devices. Several other MCU platforms are supported by the same code framework from BRT_AN_025.

The PWM feature of the FT9xx is used in this application to control the RGB LED strips. Although the PWM feature on many MCUs operates in a similar way, it may also be necessary to change this part of the code to match the PWM API used by the host MCU. Alternatively, the examples can be

run without the PWM by commenting out the PWM-related code as the on-screen interface shows the colour selected as well as the debug information.

The code was developed on the Bridgetek FT9xx IDE and uses the BRT_AN_025 EVE code framework to provide the EVE API.

## 1.4 Folder Structure

The code package for this application note is provided on Github. Refer to chapter 5 Running the Examples for details of how to use the code.

### 1.4.1 Code

The project for the FT9xx toolchain can be found in folder:

EVE-MCU-BRT_AN_025\examples\BRT_AN_074_Source\FT9XX

The example.c file contains the examples themselves and is the main source of reference for this application note. Code from this file can be ported into projects for other MCU platforms. The other files are part of the BRT_AN_025 code framework for EVE. A define at the top of the example.c file is used to select which of the four examples will be run. Note that the files eve_fonts.c and eve_images.c are intentionally blank as they are intended to replace the files of the same name when adding to the standard BRT_AN_025 example, please refer to chapter 5 Running the Examples for more details.

### 1.4.2 Images and Flash

The following files can be found in folder:

EVE-MCU-BRT_AN_025\examples\BRT_AN_074_Source\images

- Frog.jpg *Original Frog image*
- Colour_Wheel.png *Original circular colour picker image*
- Colour_Picker_Square.png *Original rectangular colour picker image*
- Circular_1555 *Converted files for Example 1*
- Square_1555 *Converted files for Example 2*
- Square_Paletted8 *Converted files for Example 3*
- Frog_Paletted8 *Converted files for Example 4*
- Flash *Flash bin and map files*

**Note:** This code is intended to act as a starting point for customers to create their own application rather than being a complete application or library package. It is necessary that developers of the final application review all parts of the code as part of their product validation. By using any part of this code, the customer agrees to accept full responsibility for ensuring that their final product operates correctly and complies with any operational and safety requirements and accepts full responsibility for any consequences resulting from its use.

The library functions are intended to perform a basic set-up so that the EVE functionality can be demonstrated. If the information provided in this application note and accompanying code differs from the datasheet of the BT81x, MCU or any other associated device, the datasheet of the device should take priority.

# 2 Using an Image as a Colour Picker

The examples presented in this application note are all based around the principle of displaying an image, which is stored as a raw data format in RAM_G, on the screen and allowing the user to select a point on the image. The colour of that pixel is then read and displayed as well as being used to set the colour of an RGB LED strip.

Any image can be used depending on the intended application, but one common use case is to display a colour gradient image which then allows the user to select from a wide range of colours.

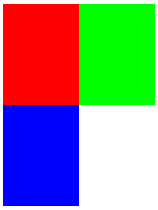
## 2.1 Image Data Format

When EVE displays an image in one of the raw data formats (such as ARGB1555 or RGB332 or RGB565) the image data is read in real time from RAM_G. Depending on the format used, there will be one or more bytes of data in RAM_G for each pixel.

The data array itself looks like one continuous one-dimensional block of values. EVE can combine this data with the knowledge of the format used and the width (leading to how many bytes represent each line of the image) and the height to produce a two-dimensional image.

In the same way, it is possible for the colour picker application to relate a particular pixel on the image back to the byte(s) in the data array in RAM_G which represent that pixel. From this, the colour can be determined.

For example, in an ARGB1555 image which is 6 pixels wide by 8 pixels tall, the image and data may look as shown below:



```
/*('file properties: ', 'resolution ', 6, 'x', 8, 'format ', 'ARGB1555', 'stride ', 12, ' total size ', 96)*/

0,252,0,252,0,252,224,131,224,131,224,131,0,252,0,252,0,252,224,131,224,131,224,131,0,252,0,252,0,252,224,131,224,131
,224,131,0,252,0,252,0,252,224,131,224,131,224,131,31,128,31,128,31,128,255,255,255,255,255,255,31,128,31,128,31,128,
255,255,255,255,255,255,31,128,31,128,31,128,255,255,255,255,255,255,31,128,31,128,31,128,255,255,255,255,255,255,
```

**Figure 2 – Example 6 x 8 image**

The ARGB1555 format uses the following data for each pixel. This includes 5 bits per colour (R, G, B) and an additional alpha bit (A) which allows for transparency of that pixel.

```
ARRR RRGG GGGB BBBB
```

As ARGB1555 uses 2 bytes per pixel and the width and height are known, the relationship between the data and the image can be shown more clearly below.

```
uint8_t data [] = {
0x00, 0xfc, 0x00, 0xfc, 0x00, 0xfc, 0xe0, 0x83, 0xe0, 0x83, 0xe0, 0x83,
0x00, 0xfc, 0x00, 0xfc, 0x00, 0xfc, 0xe0, 0x83, 0xe0, 0x83, 0xe0, 0x83,
0x00, 0xfc, 0x00, 0xfc, 0x00, 0xfc, 0xe0, 0x83, 0xe0, 0x83, 0xe0, 0x83,
0x00, 0xfc, 0x00, 0xfc, 0x00, 0xfc, 0xe0, 0x83, 0xe0, 0x83, 0xe0, 0x83,
0x1f, 0x80, 0x1f, 0x80, 0x1f, 0x80, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0x1f, 0x80, 0x1f, 0x80, 0x1f, 0x80, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0x1f, 0x80, 0x1f, 0x80, 0x1f, 0x80, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
0x1f, 0x80, 0x1f, 0x80, 0x1f, 0x80, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
```

```
}
```

**Figure 3 – Image data formatted to match image rows and columns**

For example,

- Each red pixel is 0x00 0xFC and so the bits are 1111 1100 0000 0000
- Each green pixel is 0xE0 0x83 and so the bits are 1000 0011 1110 0000
- Each blue pixel is 0x1F, 0x80 and so the bits are 1000 0000 0001 1111
- Each white pixel is 0xFF, 0xFF and so the bits are 1111 1111 1111 1111

The alpha bit is 1 as the pixels in the image are not transparent.

**Note:** The data is shown in little endian format as will be seen when converting images with EVE Asset Builder and hence the byte reversal when interpreting it.

## 2.2 Colour Picker with ARGB1555

In a colour picker application, the user can select a pixel within this image by touching the screen over the desired pixel.

In the example below, the touch coordinates reported by the touch register will be X = 12 and Y = 6. It is assumed that the screen is calibrated as this allows the alignment of the touch panel and LCD to be corrected for by EVE and allows EVE to account for any difference in resolution between the touch panel and the LCD which some modules may have.

```
TouchX = 12
TouchY = 6
```

The first step is to remove the offset of their image position on the screen as the image data relates only to the relative position within the image. The image is positioned at coordinate x = 10 and y = 4.

```
Image_X = 10
Image_Y = 4
```

Note that this offset removal should take account of the VERTEX FORMAT of the coordinates used in the VERTEX command used to position the image. If using VERTEX2F(10*16, 4*16) then the above case would have subtracted 10*16 and 4*16 respectively)

Removing this offset leaves the touch coordinates relative to the top-left corner of the image itself

```
Img_touch_X = 12 – 10 = 2
Img_touch_Y = 6 – 4 = 2
```

Using a 0-based coordinate system, this means the touch is on the 3rd cell of the 3rd row. Therefore, there are two complete rows of pixels plus two additional pixels before the required value. Stride means the number of bytes per row of the image and so with ARGB1555 format (2 bytes per pixel) will be 2 * Width

```
Image Width = 6
Image Stride = 2*6 = 12
Image Height = 8

Offset Into Image Data =  (Img_touch_X * 2 + (Img_touch_Y * Image_Width * 2));
                       =  (2 * 2 + (2 * 6 * 2));
                       =   28
```

This pixel is therefore represented by the 28th and 29th bytes in the data.

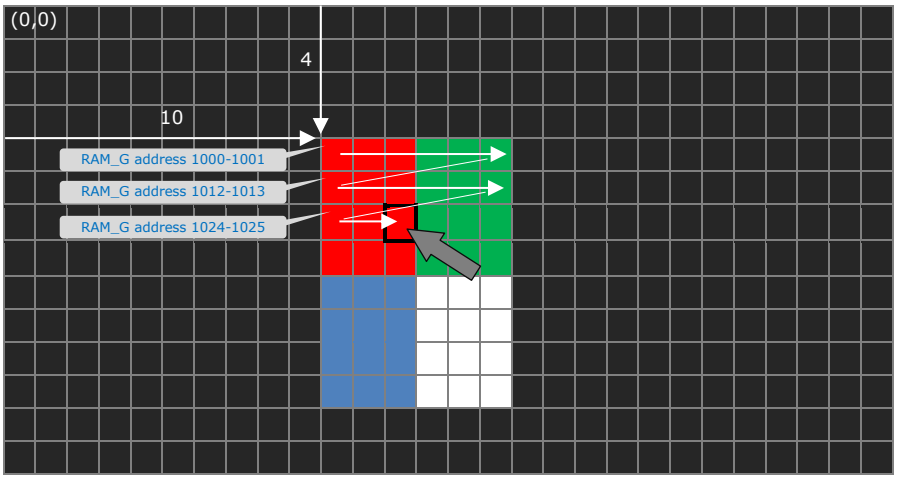
*Note that some formats such as RGB332 use only one byte (3 bits Red, 3 bits Green, 2 bits Blue) and so the above calculation would be as follows if the RGB332 format was used:*

*Offset Into Image Data = (Img_touch_X + (Img_touch_Y * Image_Width));*

*RGB565 has 2 bytes (5 bits Red, 6 bits Green, 5 bits Blue) and so would have the same offset calculation as ARGB1555.*

Assuming that the image data was loaded beginning at RAM_G + 1000, the offset of 1000 must also be added. The actual bytes will therefore be at address 1028 and 1029 in RAM_G.

Doing a 16-bit read of address 1028 will return the 16-bit value which will in this case be red 1111 1100 0000 0000.




| RAM_G Address | Data | RAM_G Address | Data | RAM_G Address | Data | RAM_G Address | Data | RAM_G Address | Data | RAM_G Address | Data | RAM_G Address | Data | RAM_G Address | Data |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1000 | 0x00 | 1012 | 0x00 | 1024 | 0x00 | 1036 | 0x00 | 1048 | 0x1f | 1060 | 0x1f | 1072 | 0x1f | 1084 | 0x1f |
| 1001 | 0xfc | 1013 | 0xfc | 1025 | 0xfc | 1037 | 0xfc | 1049 | 0x80 | 1061 | 0x80 | 1073 | 0x80 | 1085 | 0x80 |
| 1002 | 0x00 | 1014 | 0x00 | 1026 | 0x00 | 1038 | 0x00 | 1050 | 0x1f | 1062 | 0x1f | 1074 | 0x1f | 1086 | 0x1f |
| 1003 | 0xfc | 1015 | 0xfc | 1027 | 0xfc | 1039 | 0xfc | 1051 | 0x80 | 1063 | 0x80 | 1075 | 0x80 | 1087 | 0x80 |
| 1004 | 0x00 | 1016 | 0x00 | 1028 | 0x00 | 1040 | 0x00 | 1052 | 0x1f | 1064 | 0x1f | 1076 | 0x1f | 1088 | 0x1f |
| 1005 | 0xfc | 1017 | 0xfc | 1029 | 0xfc | 1041 | 0xfc | 1053 | 0x80 | 1065 | 0x80 | 1077 | 0x80 | 1089 | 0x80 |
| 1006 | 0xe0 | 1018 | 0xe0 | 1030 | 0xe0 | 1042 | 0xe0 | 1054 | 0xff | 1066 | 0xff | 1078 | 0xff | 1090 | 0xff |
| 1007 | 0x83 | 1019 | 0x83 | 1031 | 0x83 | 1043 | 0x83 | 1055 | 0xff | 1067 | 0xff | 1079 | 0xff | 1091 | 0xff |
| 1008 | 0xe0 | 1020 | 0xe0 | 1032 | 0xe0 | 1044 | 0xe0 | 1056 | 0xff | 1068 | 0xff | 1080 | 0xff | 1092 | 0xff |
| 1009 | 0x83 | 1021 | 0x83 | 1033 | 0x83 | 1045 | 0x83 | 1057 | 0xff | 1069 | 0xff | 1081 | 0xff | 1093 | 0xff |
| 1010 | 0xe0 | 1022 | 0xe0 | 1034 | 0xe0 | 1046 | 0xe0 | 1058 | 0xff | 1070 | 0xff | 1082 | 0xff | 1094 | 0xff |
| 1011 | 0x83 | 1023 | 0x83 | 1035 | 0x83 | 1047 | 0x83 | 1059 | 0xff | 1071 | 0xff | 1083 | 0xff | 1095 | 0xff |

**Figure 4 – Locating the pixel colour value**

## 2.3 Paletted 8

It may be desired to use Paletted 8 format if 8-bit resolution is required for the colour. Paletted 8 allows a smaller number of colours to be defined but at higher resolution. The previously mentioned formats such as RGB332, ARGB1555 and RGB565 have one data array which has values for every pixel (one or two bytes).

Paletted 8 differs from these as the data array does not directly contain the RGB values. Each pixel has one byte which is an index value to a separate array (palette) of colours. A second data array has the colour palette itself which defines a series of colours as 32-bit values.

In this example based on the same image used above, each pixel in the Index array has one byte. Looking up this byte in the LUT gives the colour value for the pixel.

```
uint8_t Index [] =
{
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
```

```
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
}
```

**Figure 5 – Paletted 8 Index data**

In the LUT, each colour in the palette has 4 bytes corresponding to

```
uint8_t LUT [] =
{
//      Blue   Grn    Red    Alpha
        0x00, 0xff, 0x00, 0xff,       // Green with full alpha
        0x00, 0x00, 0xff, 0xff,       // Red with full alpha
        0xff, 0xff, 0xff, 0xff,       // White with full alpha
        0xff, 0x00, 0x00, 0xff        // Blue with full alpha
}
```

**Figure 6 – Paletted 8 Look-Up Table data**

To implement the colour picker here, the initial steps are like those for the other formats above. This allows the pixel on the image to be translated to an Index value. Note that when looking up the index, there is only 1 byte per pixel instead of the 2 bytes per pixel in the previous ARGB1555 example because the Index has one byte per pixel.

In this case, it is assumed that the LUT begins at RAM_G + 0 and the Index begins at RAM_G + 1024.

Determine the position in the index file of this pixel

```
ColourBytesLoc = Image_Data_RAM_G_Paletted8 + (Img_touch_X +
(Img_touch_Y*Image_Width_Paletted8));

= 1024 + (2 + (2 * 6))
= 1024 + 14
= 1038
```

Read the value from the Index array

```
ColourByteIndex = HAL_MemRead8(ColourBytesLoc);
```

The value is 0x01

```
uint8_t Index [] =
{
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x1, 0x1, 0x1, 0x0, 0x0, 0x0,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
        0x3, 0x3, 0x3, 0x2, 0x2, 0x2,
}
```

**Figure 7 – Locating the byte from the Paletted 8 Index**

Then look through the LUT file to find the 32-bit value

```
ColourBytes = HAL_MemRead32(addr_pal_Paletted8 + (ColourByteIndex*4));
ColourBytes = HAL_MemRead32(0 + (0x01*4));
uint8_t LUT [] =

{

//      Blue   Grn    Red    Alpha
        0x00, 0xff, 0x00, 0xff,       // Green with full alpha
```

```
        0x00, 0x00, 0xff, 0xff,      // Red with full alpha
        0xff, 0xff, 0xff, 0xff,      // White with full alpha
        0xff, 0x00, 0x00, 0xff       // Blue with full alpha
}
```

**Figure 8 – Locating the colour value from the Paletted 8 LUT**

This gives the Blue, Green, Red and Alpha values, which is in this case solid red with full alpha.

It may also be desirable to use additional error checking to ensure that the lookup values always remain within the boundaries of the image data. This can be done by limiting the scope of the coordinates to within the area of the image or if the application uses touch areas outside of the colour picker, the pointers to the Index and Look Up Table could be limited to remain within the bounds of the actual data size for the image.

Note that the process of loading the Paletted 8 image also differs from the other raw data formats such as ARGB1555. Paletted 8 is loaded using blending of the alpha, red, green, and blue values and so requires four different Vertex instructions, with each one adding a different layer. It is also important to restore the blending functions and the colour mask afterwards to ensure that other items drawn afterwards in the display list do not get affected and display as expected.

```
// Load the Paletted 8 image of the colour picker
// See BT81x Programmers Guide for details of how to load a Paletted 8 image

EVE_BEGIN(EVE_BEGIN_BITMAPS);
EVE_BLEND_FUNC(EVE_BLEND_ONE, EVE_BLEND_ZERO);

//Draw Alpha channel
EVE_COLOUR_MASK(0,0,0,1);
EVE_PALETTE_SOURCE(Image_Data_RAM_G_Pal8_LUT+3);
EVE_VERTEX2II(Image_X, Image_Y, 0, 0);

//Draw Red channel
EVE_BLEND_FUNC(EVE_BLEND_DST_ALPHA, EVE_BLEND_ONE_MINUS_DST_ALPHA);
EVE_COLOUR_MASK(1,0,0,0);
EVE_PALETTE_SOURCE (Image_Data_RAM_G_Pal8_LUT+2);
EVE_VERTEX2II (Image_X, Image_Y, 0, 0);

//Draw Green channel
EVE_COLOUR_MASK(0,1,0,0);
EVE_PALETTE_SOURCE(Image_Data_RAM_G_Pal8_LUT + 1);
EVE_VERTEX2II(Image_X, Image_Y, 0, 0);

//Draw Blue channel
EVE_COLOUR_MASK(0,0,1,0);
EVE_PALETTE_SOURCE(Image_Data_RAM_G_Pal8_LUT);
EVE_VERTEX2II(Image_X, Image_Y, 0, 0);

// Return the colour mask to all-enabled and the blend function and colour setting
EVE_COLOUR_MASK(1,1,1,1);
EVE_BLEND_FUNC(EVE_BLEND_SRC_ALPHA, EVE_BLEND_ONE_MINUS_SRC_ALPHA);
EVE_COLOUR_RGB(255,255,255);
```

# 3 PWM

This example uses six of the Pulse Width Modulation channels on the FT9xx series MCU. This allows two RGB LED strips (each requiring three PWM channels for the red, green, and blue LEDs) to be driven.

The following channels were selected due to their availability on the MM900EV1B evaluation board header. The schematic in section 10 shows the connections to the MM900EV1B board.

| FT900 Device Pin | PWM | LED Connection |
|---|---|---|
| Pin 56 | PWM0 | RGB Strip 1 (Red) |
| Pin 57 | PWM1 | RGB Strip 1 (Green) |
| Pin 58 | PWM2 | RGB Strip 1 (Blue) |
| Pin 52 | PWM4 | RGB Strip 2 (Red) |
| Pin 53 | PWM5 | RGB Strip 2 (Green) |
| Pin 54 | PWM6 | RGB Strip 2 (Blue) |

**Note**: The pin assignment may vary depending on which FT9xx device/package is used

**Table 1 – PWM Connections**

```
/* Enable the PWM subsystem */
sys_enable(sys_device_pwm);

gpio_function(56, pad_pwm0); /* PWM0 */
gpio_function(57, pad_pwm1); /* PWM1 */
gpio_function(58, pad_pwm2); /* PWM2 */

gpio_function(52, pad_pwm4); /* PWM4 */
gpio_function(53, pad_pwm5); /* PWM5 */
gpio_function(54, pad_pwm6); /* PWM6 */
```

The PWM signals are used to drive the LEDs through small MOSFET transistors as shown in the Hardware section.

In the multi-touch example, each strip can be controlled individually. In the single-touch examples with only one colour picker selection, the Red, Green, and Blue PWM channels for the second strip are set to the same values are those of the first strip.

The PWM feature of the FT9xx uses a counter which can count to a configurable value before it resets to 0 (which has a maximum of 0xFFFF).

```
/* Initialise the PWM Subsystem... */
 pwm_init(
        1,                      /* Prescaler */
        0xFFFF,                 /* Max count */
        0);                     /* Shots (Infinite) */
```

Each individual channel has a register which then determines the value at which the output is enabled within the overall counter window. By varying this channel value between 0 and the maximum counter value (e.g. 0xFFFF) the pulse width of the signal can be valued between approximately 0% and 100%.

```
 /* Set up PWM 0... */
 /* Set output levels */
 pwm_levels(
        0,                          /* Counter */
        pwm_state_high,             /* Initial State */
        pwm_restore_enable);        /* Rollover State */
 pwm_compare(                       /* Set Compare value */
        0,                          /* Counter */
        0x0);                       /* Compare Value */
```

```
pwm_add_toggle(                 /* Make Counter 0 Toggle PWM0 */
        0,                      /* PWM Channel */
        0);                     /* Counter */
```

The PWM has 65536 possible values (as the max counter is 0xFFFF) whereas the data from the image will have 5-bits depth (if ARGB1555) or 8-bits depth (Paletted 8). The data is therefore shifted to put it in the most significant bits so that the maximum range can be achieved.

One advantage of having the PWM resolution higher than the source data is that calculations could be performed to linearise the LED behaviour or provide a specific dimming curve, or to account for differences in brightness between the red, green, and blue channels for example, or to calibrate the LED strip output compared to the on-screen colours. This was beyond the scope of this simple example which focuses on the colour picker aspect but could be added for a real-world application.

# 4 Preparing the Flash Image

The images used for the four examples are stored in the flash attached to the BT817 on the ME817EV and are copied over into RAM_G these images for the colour picker.

Rectangular Colour Picker Circular Colour Picker Photo of Tree Frog

**Figure 9 – Images used in the examples**

- Example 1 (Circular colour picker) uses image Circular Colour Picker (ARGB1555)
- Example 2 (Multi-Touch Colour Picker) uses image Rectangular Colour Picker (ARGB1555)
- Example 3 (Rectangular colour picker) uses image Rectangular Colour Picker (Paletted8)
- Example 4 (Image colour picker) uses image Photo of Tree Frog (Paletted8)

The following screenshots show the conversion of the files using the EVE Asset Builder.

Note that for ease of running the examples, the images for all four examples were converted and added into a single Flash image and loaded into the Flash device. In a real application, only the associated image which is to be used would be needed.

If the host MCU has sufficient flash available, the image could instead be stored and copied to EVE. This would use the raw file(s) from the individual conversions in section 4.1 below, with the content of the raw file being converted to a data array (for example using the Bin2C feature of EVE Asset Builder).

## 4.1 Image Conversion

First, the Circular Colour Wheel was converted to ARGB1555.

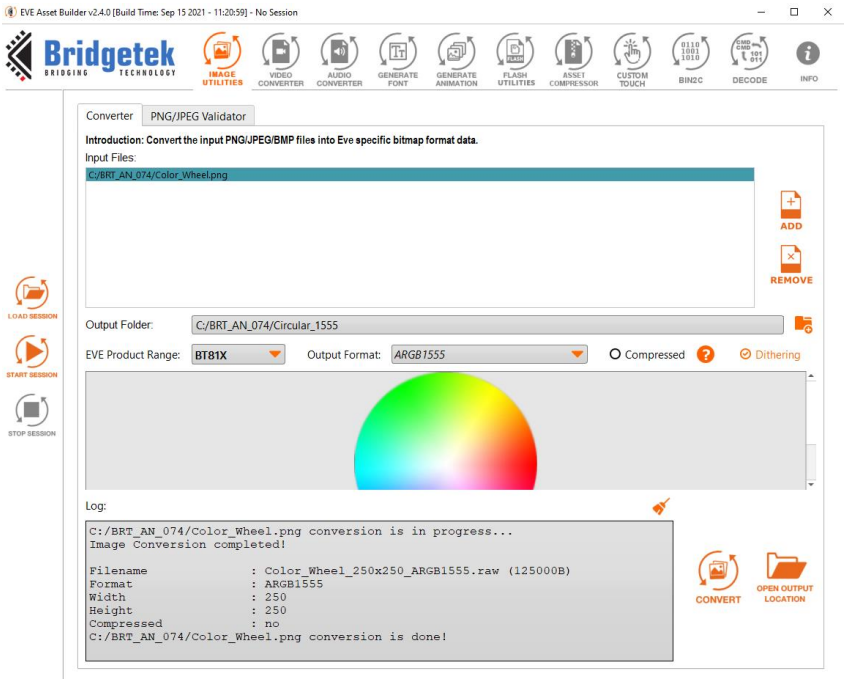



**Figure 10 – Converting the circular colour picker to ARGB1555**

Then, the rectangular colour picker was converted to ARGB1555.

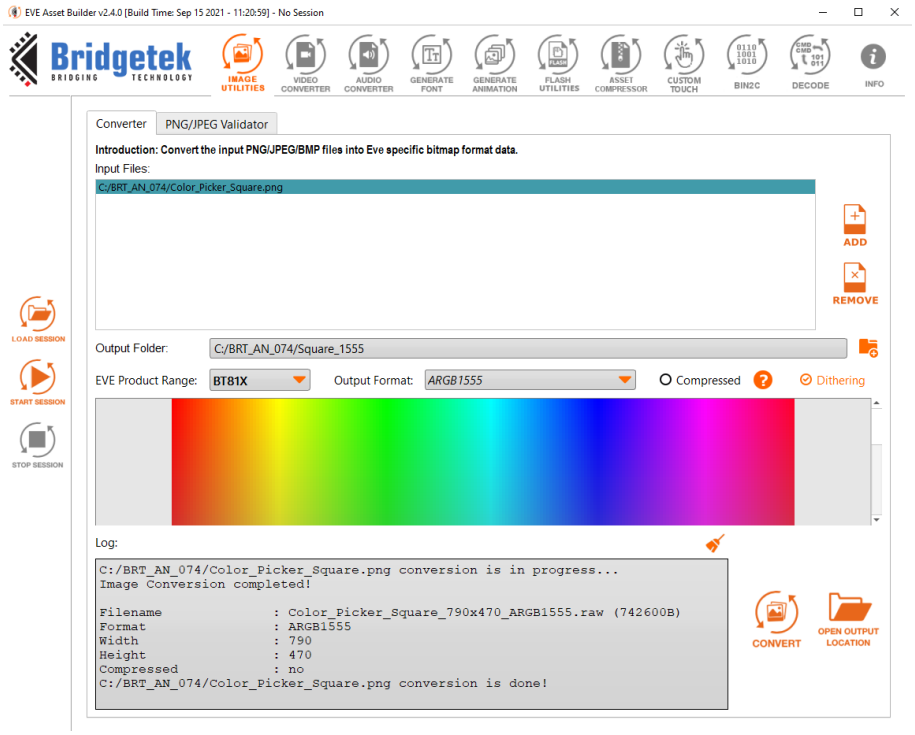



**Figure 11 – Converting the rectangular colour picker to ARGB1555**

The rectangular colour picker was then also converted to Paletted 8 format by selecting this in the Output Format drop-down.

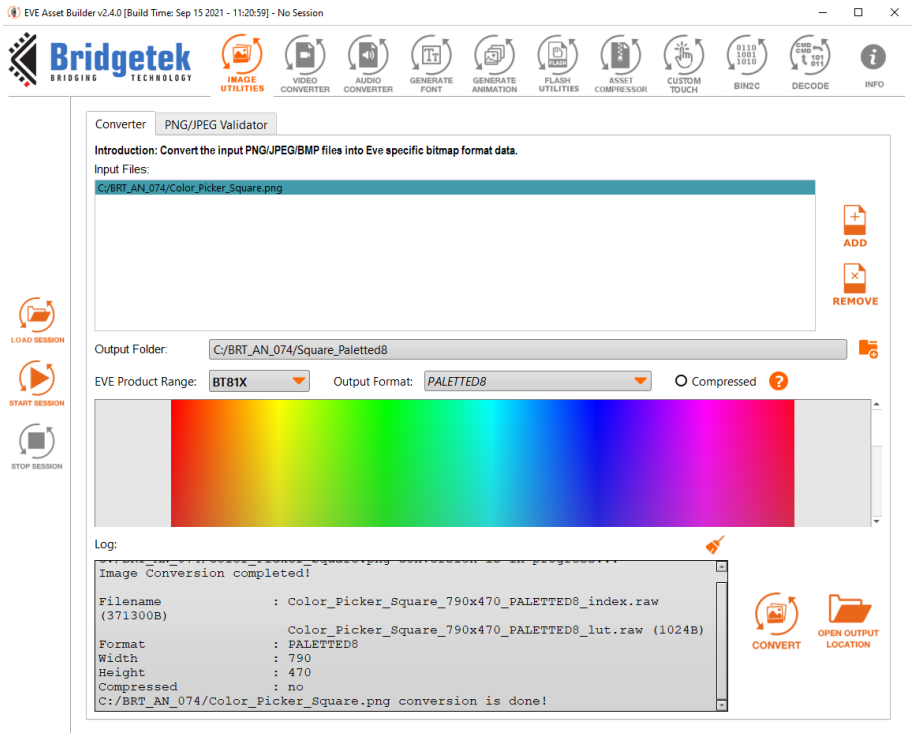



**Figure 12 – Converting the rectangular colour picker to Paletted8**

Finally, the image of the Tree Frog was converted to Paletted 8 format.

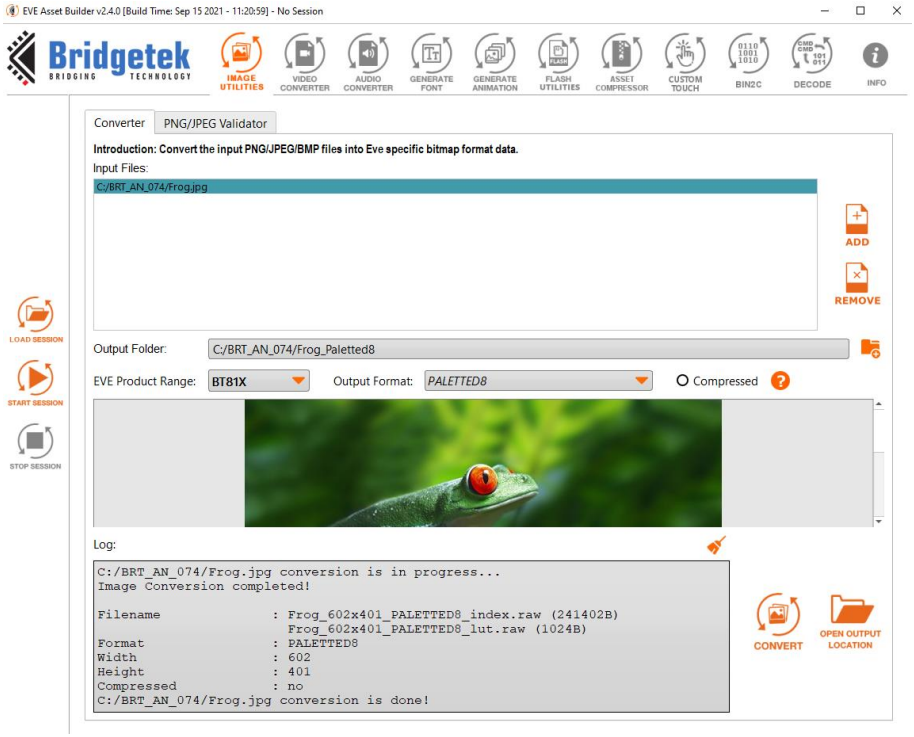



**Figure 13 – Converting the Tree Frog image to Paletted8**

## 4.2 Flash Image

Once the images were converted, the Flash Utilities option in EVE Asset Builder was used to create a flash image.

First, the converted files were added to the list in the Generate Flash Image tab of the Flash Utilities. To do this, select the correct EVE Chip type (in this case BT817/8) and this will select the correct BLOB file which is shown first in the list. Then, the raw files of the converted images were added.

For ARGB1555, there is just one raw file in the output folder which contains the converted image data. For the Paletted 8 images, there is the LUT (Look Up Table) file and the Index file. The raw version of each of these was added to the list.

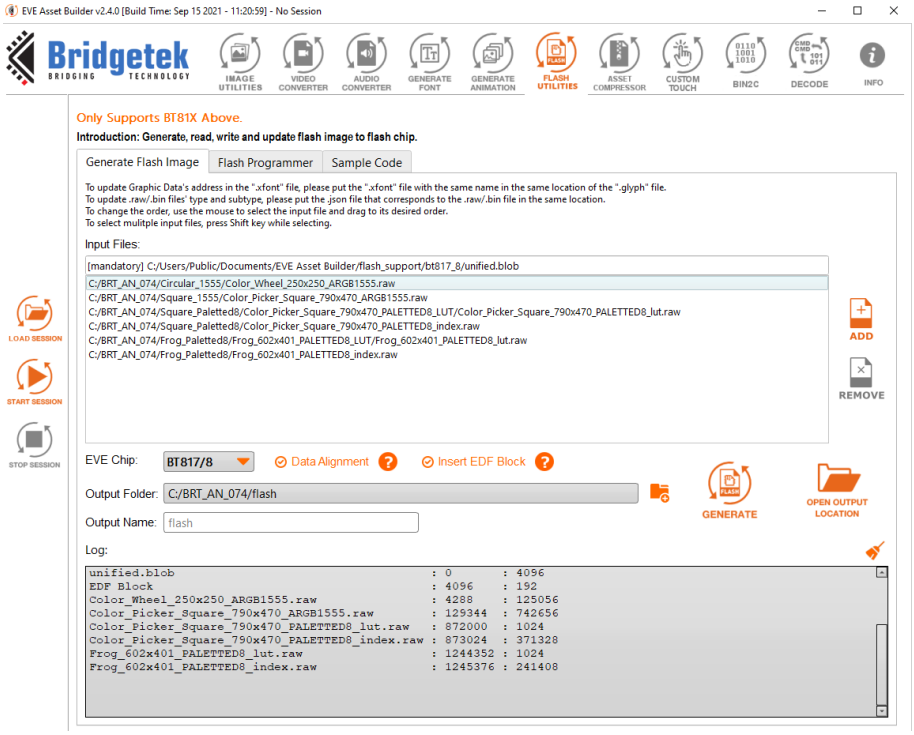



**Figure 14 – Creating the Flash Image**

After generating the flash file, the map file can be viewed in the Log window and is also present in the output folder alongside the bin file. The map file shows the name, starting address in flash, and size of each file in flash, and can be used as a reference when locating files in Flash.

Note that the latest version of EAB has a new EDF block which contains a summary of the files and so the host MCU can read this instead of having the addresses and lengths from the map file coded into the application.

| Asset | Start Addr in Flash | Length |
|---|---|---|
| unified.blob | 0 | 4096 |
| EDF Block | 4096 | 192 |
| Colour_Wheel_250x250_ARGB1555.raw | 4288 | 125056 |
| Colour_Picker_Square_790x470_ARGB1555.raw | 129344 | 742656 |
| Colour_Picker_Square_790x470_PALETTED8_lut.raw | 872000 | 1024 |
| Colour_Picker_Square_790x470_PALETTED8_index.raw | 873024 | 371328 |
| Frog_602x401_PALETTED8_lut.raw | 1244352 | 1024 |
| Frog_602x401_PALETTED8_index.raw | 1245376 | 241408 |

**Figure 15 – Flash MAP file**

## 4.3 Programming the Flash

The Flash can now be programmed. There are several ways to do this.

If using the ME817EV, the module has an on-board FT4222H which can be enabled using DIP switches and can be used as the USB-SPI interface for Flash programming. Please refer to the ME817EV datasheet for details.

Alternatively, as shown below, the EVE module can be disconnected from the MM900EV1B and an MPSSE adapter (such as USBC-HS-MPSSE-5V-3.3V-500-SPR) can be used to connect the BT81x to the PC and the flash can be programmed. Once completed, the USB-SPI adapter can be disconnected and the EVE module can be connected again to the MM900EV1B.

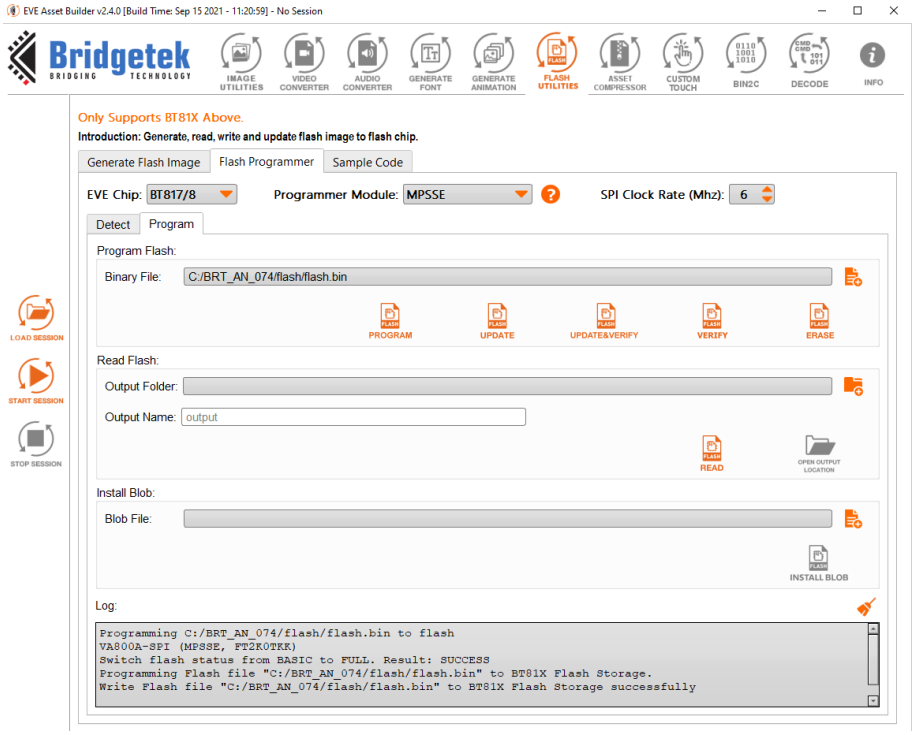



**Figure 16 – Programming the Flash**

## 4.4 Copying the Image from Flash

When each example is run, the code uses the FLASHREAD command to copy the data from Flash to the RAM_G. The code in the ConnectToFlash(); function puts the flash into full mode before the flash is used. In this example below, the image of the frog is used which is in paletted8 format.

From the map file shown earlier in this section, the start addresses and sizes can be found.

| | | |
|---|---|---|
| Frog_602x401_PALETTED8_lut.raw | 1244352 | 1024 |
| Frog_602x401_PALETTED8_index.raw | 1245376 | 241408 |

First, the Look Up Table is copied from Flash address 1244352 to RAM_G + 0. The size copied is 1024 bytes.

```
// copy Look Up Table (LUT) from flash to RAM
EVE_LIB_BeginCoProList();
EVE_CMD_FLASHREAD(0, 1244352  , 1024);
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();
```

Then, the Index data is copied from Flash address 1245376 to RAM_G + 1024 (directly after the LUT copied above). The size copied is 241408 bytes.

```
// copy Index from flash to RAM
EVE_LIB_BeginCoProList();
EVE_CMD_FLASHREAD(1024, 1245376   , 241408);
EVE_LIB_EndCoProList();
EVE_LIB_AwaitCoProEmpty();
```

After this copying is done, the data is now in RAM_G and can be used to render the image via a display list on the screen.

# 5 Running the Examples

To run the examples on an FT9xx MCU, Load the project (from the path below within the Github download) into the FT9xx Toolchain

EVE-MCU-BRT_AN_025\examples\BRT_AN_074_Source\BRT_AN_074_Color_Picker_Examples

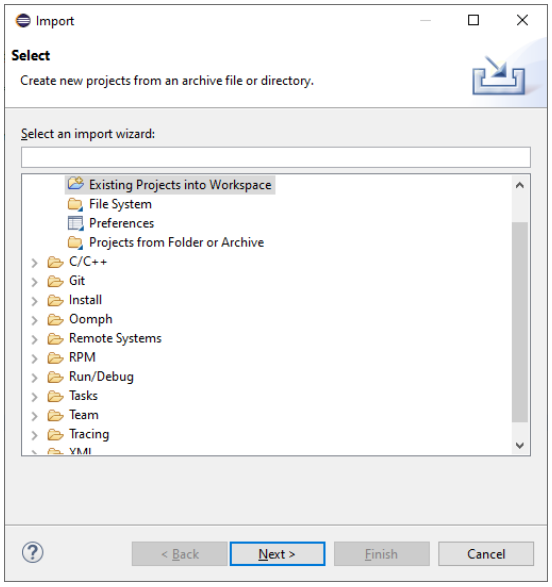



**Figure 17 – Add existing project to workspace**

Select the provided project folder as shown below.

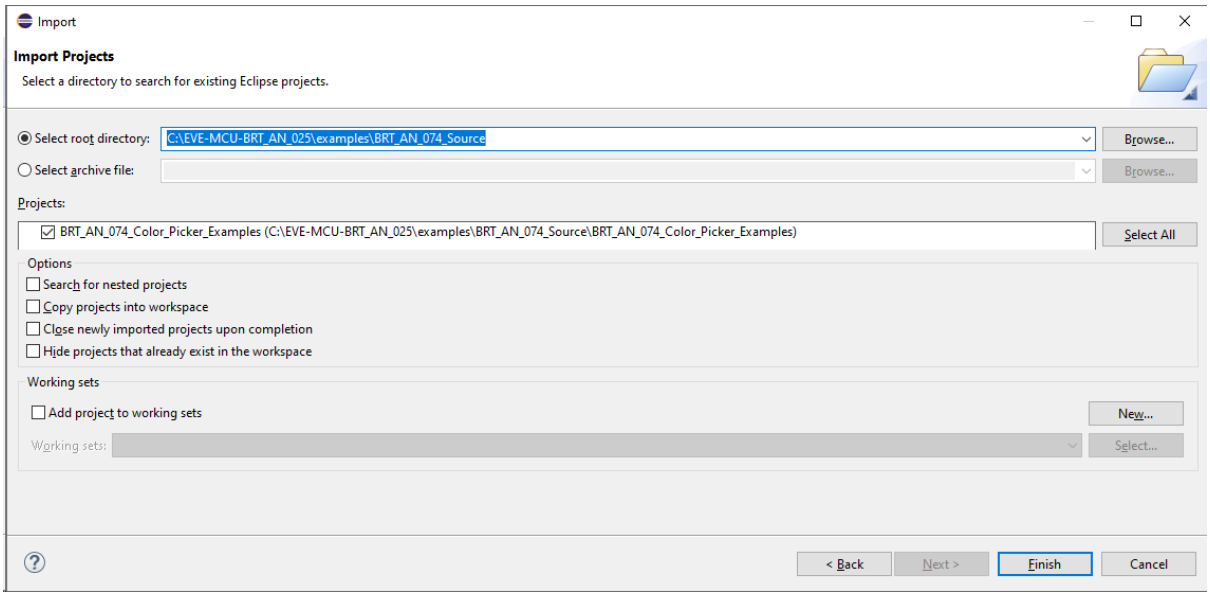



**Figure 18 – Select the project folder**

The project will then appear in the list as shown.

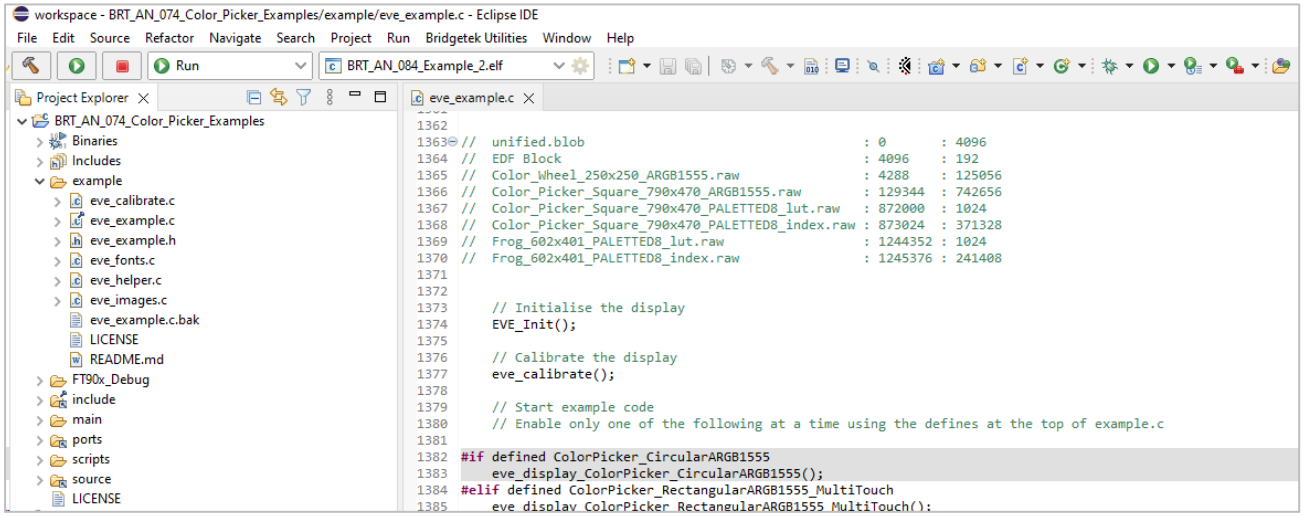



**Figure 19 – Project loaded in the FT9xx IDE**

Before building the project, select the example to be run using the defines at the top of the example.c file.

Note that the flash must also have been loaded beforehand using the process in section 4.3.

```
// Select which example to run. Un-comment only ONE of the examples at any time

#define ColourPicker_CircularARGB1555

//#define ColourPicker_RectangularARGB1555_MultiTouch
//#define ColourPicker_RectangularPaletted8
//#define ColourPicker_ImagePaletted8
```

# 6 Example 1 – Circular Colour Picker (ARGB1555)

In this example, a circular colour picker is created with a single colour picker cursor. This can be used on capacitive or resistive screens.

Features include:

- Circular colour picker image
- Cursor to indicate current selection on the colour gradient
- Status Panel to display the selected colour
- Debug Information is also printed to show the cursor position/tag and the colour values
- Red Warning Ring illuminates if the user touch goes outside the colour picker area
- PWM Output to drive the LED Strip to indicate the colour selected

Some illustrations of the colour picker is shown below. As this colour picker has only one touch point / cursor the RGB LED strip around the top half of the display (PWM 0, 1, 2) and the RGB LED strip around the bottom half of the display (PWM 4, 5, 6) are both driven to the same colour.

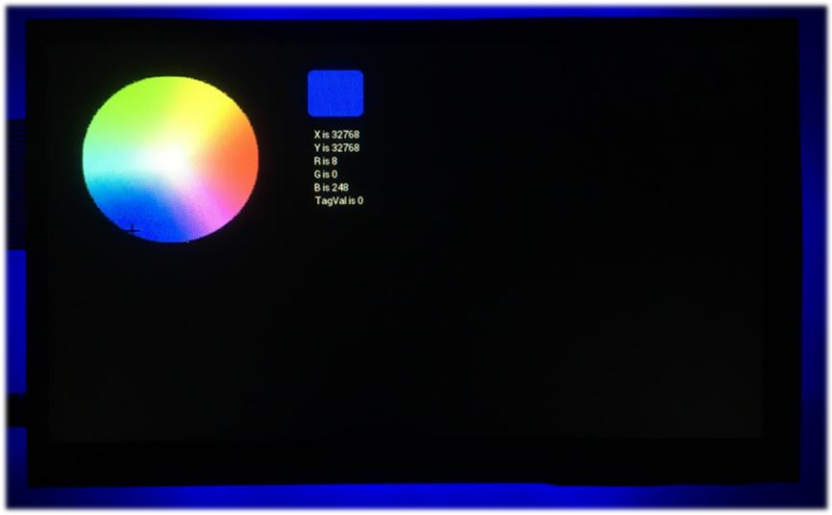



**Figure 20 – Example 1 – Selecting a blue colour**

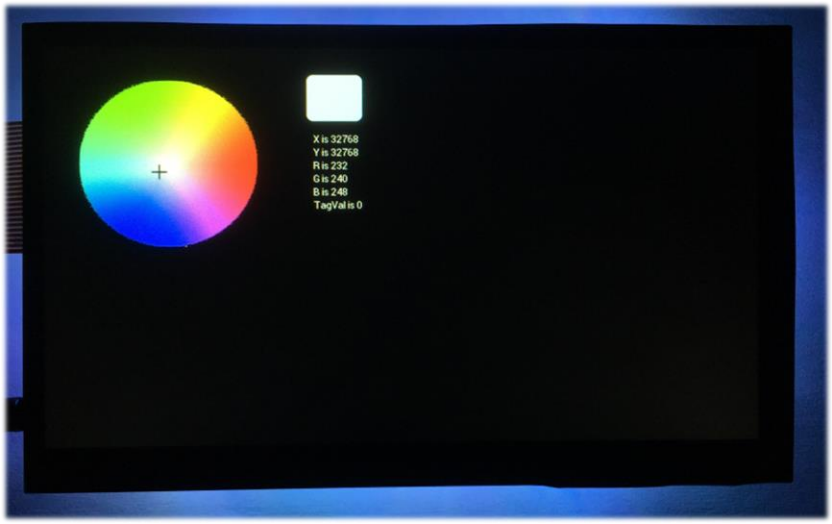



**Figure 21 – Example 1 – Selecting a white colour**

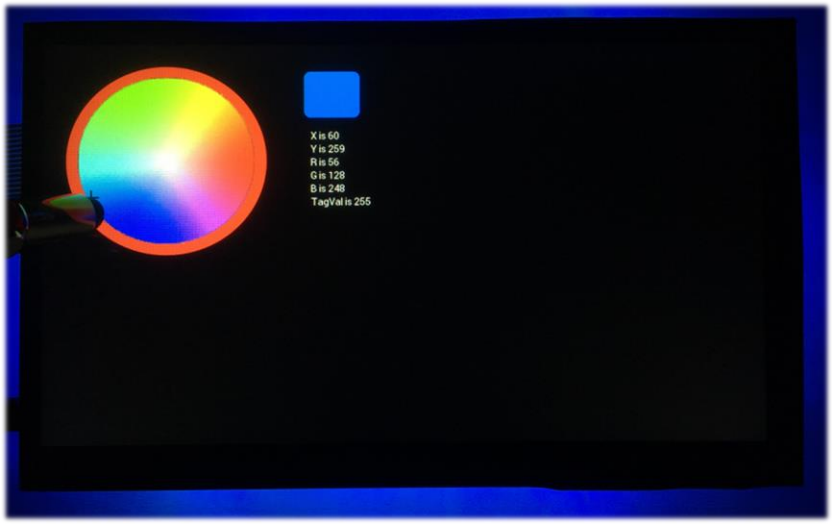



**Figure 22 – Example 1 – Red warning ring for touch out of range**

The code begins by setting up the BT81x device, and then initialising the PWM and flash. It then copies the image of the circular colour picker to RAM_G.

It then uses a constant loop consisting of:

- Create a display list to render the colour picker image, the cursor, and the colour indicator panel and debug print information
- Check the touch tag and registers to see if the tagged area of the colour picker is being touched
- If a touch is present in the colour picker area
    - The touch coordinates are transformed to be relative to the image
    - The touch information is used to find the colour value of the pixel being touched
    - The PWM values are updated
    - The variables for the colour value and cursor coordinates are updated to be used the next time the screen is rendered

For the ARGB1555 colour picker operation please refer to section 2.2

This example uses an invisible tagged circle, which is slightly smaller than the colour picker circle, for the touch sensing as the colour picker image itself is a square shape and it is desired to limit the tagged area to the actual colour circle.

# 7 Example 2 – Multi-Touch Colour Picker (ARGB1555)

In this example, a larger rectangular colour picker is created with two colour picker cursors for capacitive 7" screens on the BT815/7. These can be used simultaneously due to the multi-touch functionality of the BT815/7.

Features include:

- Large rectangular colour picker image
- Two colour pickers A and B which can be moved simultaneously
- Cursors to indicate current position of each picker A and B
- Dual Status Panel to display the selected colour for each cursor A and B
- Debug Information is also printed to show the cursor position/tag and the colour values
- PWM Output x6 to drive two LED Strips (A above the screen and B below the screen)

Some illustrations of the colour picker is shown below. The selections on the A and B cursors are used to control the colour LED strip above (PWM 0, 1, 2) and below (PWM 4, 5, 6) the screen respectively, allowing different top and bottom colours to be chosen.

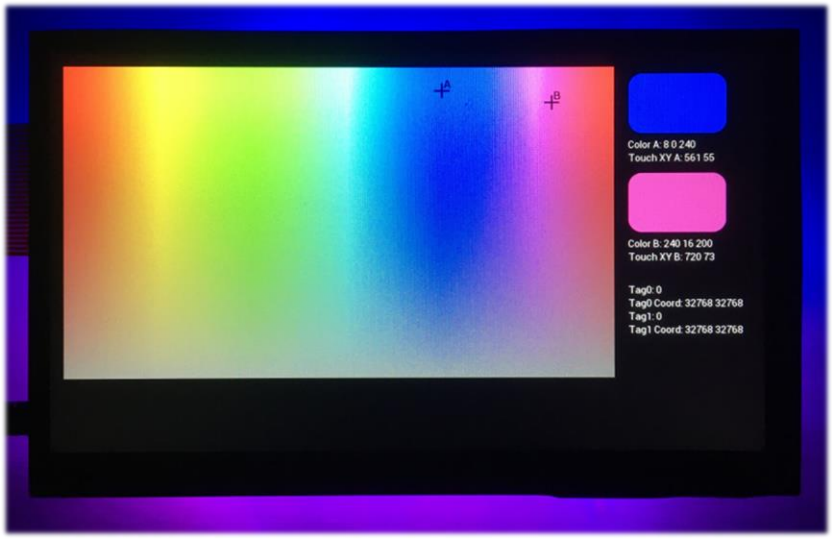



**Figure 23 – Example 2 – Selecting Blue and Magenta**

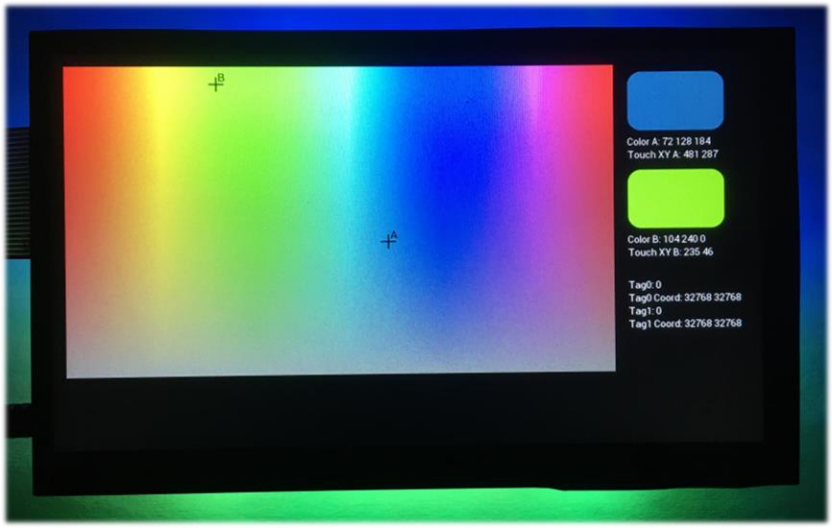



**Figure 24 – Example 2 – Selecting Blue and Green**

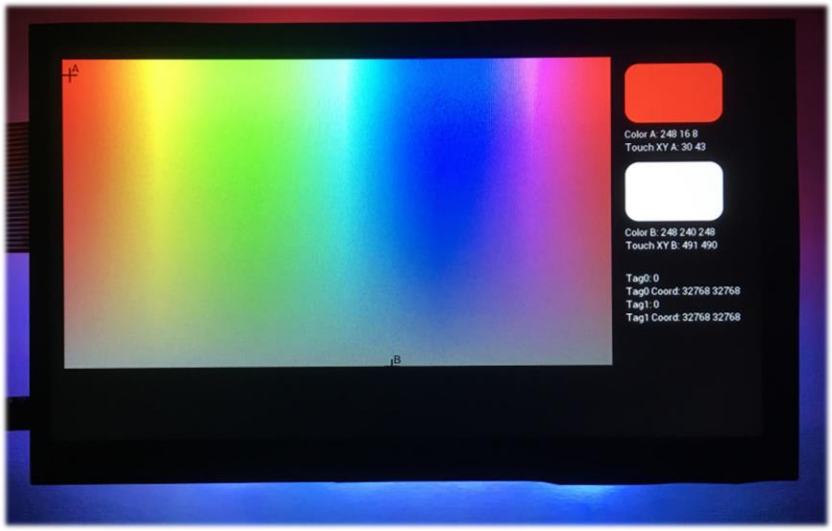



**Figure 25 – Example 2 – Selecting Red and White**

This example works along similar principles to Example 1 but has some differences in the touch.

From a user perspective, the use of two touch points requires that the cursors are dragged by positioning the finger over the active area (an invisible circle drawn around the cursor).

```
// Disable writes to the R, G and B components to make the circle invisible
EVE_COLOUR_MASK(0,0,0,255);
EVE_BEGIN(EVE_BEGIN_POINTS);
EVE_POINT_SIZE(40*16);                      // Set size to 40 pixels
EVE_TAG_MASK(1);                            // Enable tagging of the following items
EVE_TAG(10);
EVE_VERTEX2F(Point0_X*16, Point0_Y*16);     // Draw invisible circle tagged 10
EVE_TAG(11);
EVE_VERTEX2F(Point1_X*16, Point1_Y*16);     // Draw invisible circle tagged 11
EVE_TAG_MASK(0);                            // Prevent further items being tagged
```

Example 1 moves the cursor immediately to wherever the user taps the screen but as there are now two cursors, the user drags each cursor so that the application knows which one the user wants to move.

In addition to the small cursor displayed for each point A and B (which are created by drawing short lines across the current touch point), letters A and B are added by placing the bitmap cell of these characters from the built-in fonts in the BT81x.

From a code perspective, one key point to note that that the tag registers 0 and 1 are not permanently linked to the cursor A or B. The tag registers and their associated coordinates are associated with the first and second touches and depend on the order of the touch. The first touch that the user makes to the screen will be in Tag 0 and associated X and Y coordinates. If they add a second touch whilst holding the first, this will be reflected in Tag 1. The same would apply for third, fourth and fifth touches to Tag 2, 3 and 4 if these were also used.

Also note that the TOUCHn_XY or the TOUCH_TAGn_XY registers can be used. One advantage of the TOUCH_TAGn_XY register is that it holds the coordinates used to check the Tag value. This helps to avoid the tag lookup and the current coordinates having a mismatch if the user moves the touch across the screen quickly whereby the tag could be identified as tag 10 but the coordinate could be read slightly later and the touch could be outside the item tagged with 10.

```
Tag0 = HAL_MemRead8(EVE_REG_TOUCH_TAG);
Tag0_Coord = HAL_MemRead32(EVE_REG_CTOUCH_TOUCH0_XY);
//Tag0_Coord = HAL_MemRead32(EVE_REG_TOUCH_TAG_XY);
Tag0_Coord_X = (Tag0_Coord >> 16);
```

```
Tag0_Coord_Y = (Tag0_Coord & 0x0000FFFF);

Tag1 = HAL_MemRead8(EVE_REG_TOUCH_TAG1);
Tag1_Coord = HAL_MemRead32(EVE_REG_CTOUCH_TOUCH1_XY);
//Tag1_Coord = HAL_MemRead32(EVE_REG_TOUCH_TAG1_XY);
Tag1_Coord_X = (Tag1_Coord >> 16);
Tag1_Coord_Y = (Tag1_Coord & 0x0000FFFF);
```

The application must check each touch register (0 and 1 in this case to support up to 2 simultaneous touches) in turn and then determine if the touch is on the tagged area. If so, the coordinates for that tag can be recorded against that touch point. Therefore, after the code section below, the Point0_X and Point0_Y (coordinates of touch cursor A) and Point1_X and Point1_Y (coordinates of touch cursor B) will have been updated if a touch was present on them.

```
// If there is a touch on Tag 0
if(Tag0_Coord != 0x80008000)
{
	if(Tag0 == 10)
	{
		Point0_X = Tag0_Coord_X;
		Point0_Y = Tag0_Coord_Y;
	}
	else if (Tag0 == 11)
	{
		Point1_X = Tag0_Coord_X;
		Point1_Y = Tag0_Coord_Y;
	}
}

// If there is a touch on Tag 1
if(Tag1_Coord != 0x80008000)
{
	if(Tag1 == 10)
	{
		Point0_X = Tag1_Coord_X;
		Point0_Y = Tag1_Coord_Y;
	}
	else if (Tag1 == 11)
	{
		Point1_X = Tag1_Coord_X;
		Point1_Y = Tag1_Coord_Y;
	}

}
```

Once the coordinates for each tag are known, the same techniques as detailed in section 2.2 can be used to determine the colour. The updated coordinates are also stored so that the centre coordinate of the cursor can be updated the next time round the while loop, where the screen will be re-drawn.

# 8 Example 3 – Rectangular Colour Picker (Paletted8)

This example focuses on the use of the Paletted 8 format. The same larger rectangular colour picker is created as in Example 2. However, in this case Paletted 8 format is used instead and only a single touch is implemented.

Paletted 8 offers a greater depth of 8 bits per colour but also has a more limited number of colours. Therefore, the benefit over the 5-bit ARGB1555 may vary depending on the actual image used and in some cases the simpler ARGB1555 image format may offer visually good results.
Features include:

- Large rectangular colour picker image
- Single colour picker cursor
- Cursor with crossed lines to indicate current position of the selected colour
- Status Panel to display the selected colour
- Debug Information is also printed to show the cursor position/tag and the colour values
- PWM Output to drive the LED Strip to indicate the colour selected

Some illustrations of the colour picker is shown below.

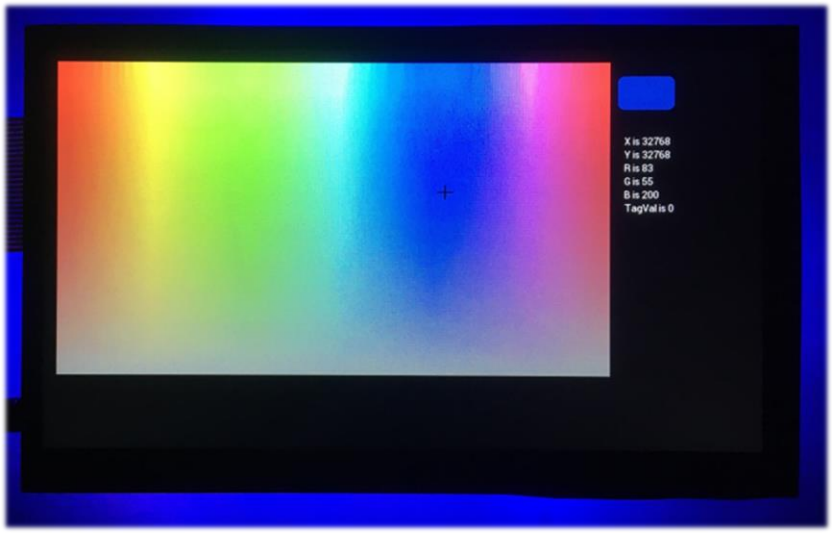



**Figure 26 – Example 3 – Selecting Blue shade on Paletted 8**

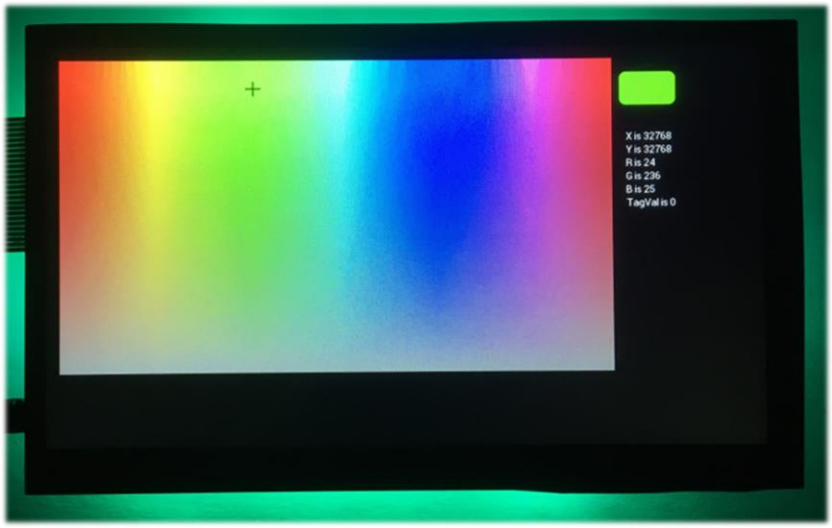



**Figure 27 – Example 3 – Selecting Green shade on Paletted 8**

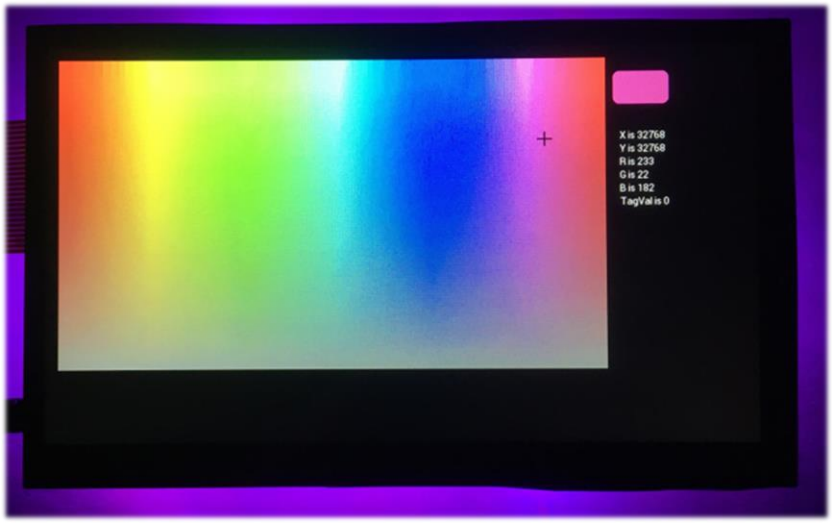



**Figure 28 – Example 3 – Selecting Magenta shade on Paletted 8**


**Figure 29 – Example 3 – Selecting Orange shade on Paletted 8**

This example differs from the earlier ones in the way that the image data is looked up once the touch point has been determined. Please refer to section 2.3 for details of how the Paletted 8 colour lookup is performed.

The touch tag is applied to a non-visible rectangle drawn into the alpha buffer and so, by only updating the coordinates when a valid tag is detected, the scope of the cursor movement can be restricted to the area of the colour picker gradient.

In other cases, it may be necessary or recommended to add error checking to ensure that the lookup coordinates for the image data are within the bounds of the image itself. This will ensure that the colour data lookup is always within the array of data.

More comprehensive error prevention can be added but one simple step is to check and limit the coordinates to within the active area as shown below.

```
// Remove any offset from the top-left position of image
Img_touch_X = TouchX - Image_X;
Img_touch_Y = TouchY - Image_Y;

// Make sure that any coordinates obtained by the code above
// are restricted to within the colour picker area
if(Img_touch_X < Image_X)
```

```
        Img_touch_X = Image_X;
else if (Img_touch_X > (Image_X + Image_Width_Paletted8))
        Img_touch_X = Image_X + Image_Width_Paletted8;

if(Img_touch_Y < Image_Y)
        Img_touch_Y = Image_Y;
else if (Img_touch_Y > (Image_Y + Image_Height_Paletted8))
        Img_touch_Y = Image_Y + Image_Height_Paletted8;
```

# 9 Example 4 – Image Colour Picker (Paletted8)

This example is closely based on Example 3 but with a different image. It is intended to show the use of a general photo with Paletted 8 format. This could be used in lighting applications where a scene is set based on colours from a photo or could be part of an application where the user sets a colour theme based on a photo or where it is required to test colours in parts of an image and retrieve their colour components.

Features include:

- Checking the colour of pixels in a photo
- Single colour picker cursor
- Cursor to indicate current position of the selected colour
- Status Panel to display the selected colour
- Debug Information is also printed to show the cursor position/tag and the colour values
- PWM Output to drive the LED Strip to indicate the colour selected

Some illustrations of the colour picker is shown below. An image of a Tree Frog was used, and areas of the image were shown selected using the cursor. Note that a small red circle is added to the screenshot to highlight the cursor position within the screenshot photos.

One enhancement to this example or indeed any of the others may be to change the type of cursor used. This could include the use of thicker lines or dynamically changing the colour to ensure good contrast with the image behind. The crossed lines could be replaced with a dropper symbol like that used in the colour-picker feature of many paint programs which could be converted in a format with transparent background and alpha support (such as ARGB1555).

This example uses almost identical code to Example 3 except for loading a different image along with the associated addresses in Flash and the width and height.


**Figure 30 – Example 4 – Selecting green area of background**

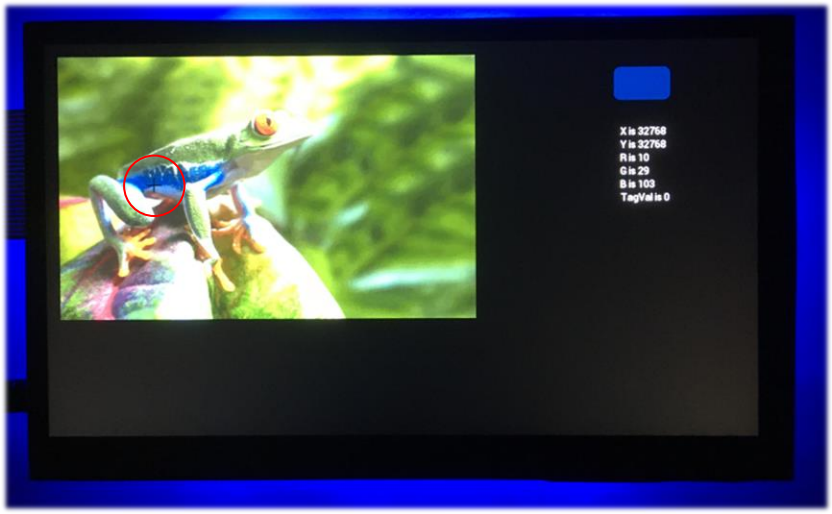



**Figure 31 – Example 4 – Selecting Blue colour on the side of the frog**

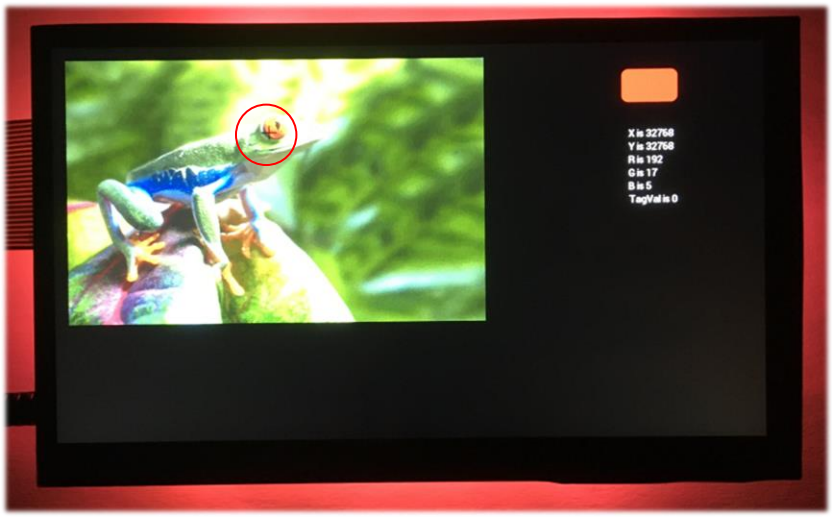



**Figure 32 – Example 4 – Selecting the Red area of the frog's eye**

# 10 Hardware

The hardware is quite simple as the MM900EV1B and ME817EV contain most of the circuitry needed. The only addition is for the RGB LED strip and associated transistors to drive the LEDs.
The 12V LED strip tape contains groups of three RGB LEDs connected in series along with series resistors.

As the outputs from the MM900EV1B are at 3.3V logic levels, a small MOSFET is used to drive each channel of the 12V LED tape by sinking the cathode of the LEDs to GND. The LED strip is powered from a separate 12V supply with its GND common to the rest of the circuit.

The ME817EV EVE board is connected to the header J2 on the MM900EV1B. The MM900EV1B can receive power via its USB micro-B connector from a USB power supply or through a JST connector. The ME817EV and display are powered directly from a 5V supply via the connection on the ME817EV itself to ensure sufficient current for the display and backlight.

The LED tape used here is the original type with just standard 5050 RGB LEDs and series resistors but other types of output device could be used such as outputting DMX from a UART or controlling intelligent LED strip which has LED drivers on-board and can be controlled by a data stream.

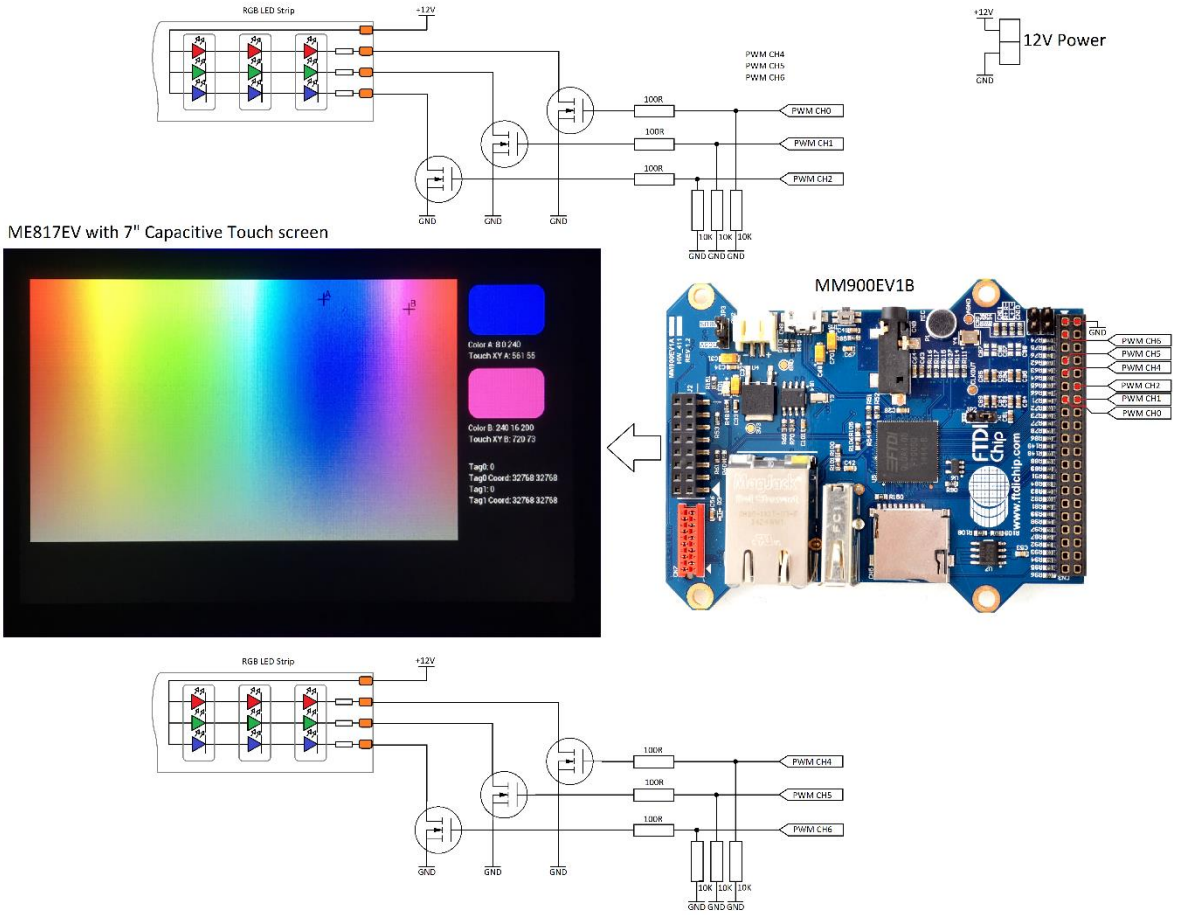



**Figure 33 – Schematic showing the LED connections**

# 11 Conclusion

This application note has provided an introduction to creating a colour picker on the Bridgetek BT81x series devices using raw images displayed from RAM_G. The four examples included help to illustrate the technique and its practical implementation.

It has also provided a simple example of using six PWM channels on the FT9xx series MCU to control RGB LEDs.

There are many variations and enhancements which could be made to create a final application. The user interface out with the colour picker is intentionally basic to focus on the colour picker aspects but the colour picker could be used as part of a more fully featured and attractive user interface.

A variety of other images can be used to meet the needs of a wide range of applications such as lighting control or allowing the user to pick colour schemes for a user interface based on a photo.

# 12 Contact Information

Refer to https://brtchip.com/contact-us/ for contact information

### Distributor and Sales Representatives

Please visit the Sales Network page of the Bridgetek Web site for the contact details of our distributor(s) and sales representative(s) in your country.

# Appendix A– References

## Document References

| | |
|---|---|
| BRT_AN_074 | Source code for this application note |
| BRT_AN_025 | EVE Framework used in this application note |
| BRT_AN_025_EVE Portable MCU Example | |
| BT81x | BT81x product page |
| BT81x | BT81x Programmers Guide |
| BT81x | BT81x Datasheet |
| ME817EV | ME817EV BT817 development module |
| EVE Asset Builder | EVE asset Builder tool |
| MM900EV1B | FT90x evaluation board |
| FT900 Toolchain | FT90x Eclipse-based toolchain |

*Credits: Red-eyed tree frog image used in example 4 from the Microsoft Word Stock Images*

## Acronyms and Abbreviations

| Terms | Description |
|---|---|
| EVE | Embedded Video Engine |
| MCU | Microcontroller |
| BT81x | Latest version of the EVE family with enhanced feature set |
| LCD | Liquid Crystal Display |
| SPI | Serial Peripheral Interface |
| UART | Universal Asynchronous Receiver Transmitter for serial data transfer |
| PWM | Pulse Width Modulation |

# Appendix B – List of Tables & Figures

## List of Tables

NA

## List of Figures

# Appendix C– Revision History

Document Title: BRT_AN_074 BT81x Simple Color Picker for RGB LED control
Document Reference No.: BRT_000361
Clearance No.: BRT#212
Product Page: http://brtchip.com/i-ft8/
Document Feedback: Send Feedback

| Revision | Changes | Date |
|---|---|---|
| 1.0 | Initial release | 19-04-2024 |