Topic:	Drawing Text and Images
Author:	JOKI
Date:	05.07.2021

0. Before you start

This document will give you an overview of the source code for the example package Drawing Text and Images. Before you can work with it you need to set up your working environment as explained in the document “ExamplePackage_GettingStarted”. Make sure you have read this document beforehand and executed all the steps to configure your Workbench. Especially, that you have added all the paths to the *Source Location* and *Includes* properties.

For a more detailed explanation of how the microcontroller works, please refer to the STM32F429 Reference Manual (RM0090) and the Standard Peripheral Library documentation provided by STMicroelectronics.

1. Introduction

In this example package we will use basic functions to draw different items on the display. Additional to lines and rectangles, we will draw complete images and write text. The display interface of the STM32F429 microcontroller can work with two layers, which can be merged together to get the final output. But we will only use one layer. Each layer has two memory buffers. You can select in your code, which one shall be the active one. So, while one buffer is displayed, you can fill the other buffer with new image data and switch buffer. These buffers are located on the external SDRAM.

2. Explanation of Example Code

2.1 Main function

main.c

```
29 extern tImage ebssystart_logo_35;
30 extern tImage twoInone_display_logo;
31 extern tFont courier_new;
32 extern tFont courier_new_italic;
33 extern tImage smiley_100px;
34
35/*************************************************
36 * Private Function Prototypes
37 *************************************************/
38 void HW_Init(void);
39 void display_init(void);
40 void gpio_init(void);
41 void status_led_blink(void);
42
43/*****************************************************************
44 *
45 * Function: int main(void)
46 *
47 * Summary: Entry point of display control application
48 *
49 * Parameters:
50 *
51 * Return:
52 *
53 * Revision History:
54 * Creation Date: 08042019
55 *
56 *****************************************************************/
57int main(void) {
58     /* Initialize the peripherals */
59     HW_Init();
60     display_init();
61
62     DMA2D_Fill_Color(0xFF0066FF, Layer_1, Buffer_1);
63     DMA2D_Draw_FilledRectangle(21, 15, 50, 30, 0xFF00FF00, Layer_1, Buffer_1);
64     DMA2D_write_string("Hello World!", 25, 20, 0x00FF0000, &courier_new , Layer_1, Buffer_1, Layer_1, Buffer_1);
65     DMA2D_write_string("I'm italic.", 25, 50, 0x00FF0000, &courier_new_italic , Layer_1, Buffer_1, Layer_1, Buffer_1);
66     DMA2D_Draw_V_Line(20, 6, 75, 1, 0xFF000000, Layer_1, Buffer_1);
67     DMA2D_Draw_X_Line(5, 45, 210, 1, 0xFF000000, Layer_1, Buffer_1);
68     DMA2D_Draw_Line(0, 240, 320, 0, 0xFF00FFFF, Layer_1, Buffer_1);
69     DMA2D_Draw_Image(200, 100, smiley_100px.width, smiley_100px.height, 0xFF, NO_MODIF_ALPHA_VALUE, (uint32_t)smiley_100px.data, CM_ARGB8888, Layer_1, Buffer_1);
70
71     for(;;){
72         status_led_blink();
73     }
74     return 0;
75 }
```

We will start our walk through the code at the main function since this gives a perfect overview of the steps that we will take. At the beginning, the used peripherals are initialized. *HW_Init()* initializes and turns on the display backlight. Also, it sets up the different system clocks and the timer, which counts every microsecond.

Afterwards we initialize the display which also includes the initialization of the SDRAM, because this is where the buffer for the displayed images will be located. The DMA2D is used to send the image data directly from the buffer to the LCD, which speeds up the image manipulation. Part of the display initialization is the drawing of the EBS-SYSTART logo and the logo of our 2in1 display line.

After the initialization process, we start with the various drawing options. We will take a closer look at those functions. The definitions of these functions can be found in *src/driver/d_Display_DMA2D.c*.

2.2 Function: DMA2D_Fill_Color

```
179 void DMA2D_Fill_Color(uint32_t color, Layer_e layer, Buffer_e buffer){
180     DMA2D_InitStruct.DMA2D_Mode = DMA2D_R2M;
181     if(PIXEL_FORMAT == LTDC_Pixelformat_RGB565){
182         uint16_t color16 = (uint16_t) color;
183         DMA2D_InitStruct.DMA2D_CMode = DMA2D_RGB565;
184         DMA2D_InitStruct.DMA2D_OutputBlue = (0x001F & color16);
185         DMA2D_InitStruct.DMA2D_OutputGreen =(0x07E0 & color16) >> 5;
186         DMA2D_InitStruct.DMA2D_OutputRed  = (0xF800 & color16) >> 11;
187     }
188     else if(PIXEL_FORMAT == LTDC_Pixelformat_RGB888){
189         DMA2D_InitStruct.DMA2D_CMode = DMA2D_RGB888;
190         DMA2D_InitStruct.DMA2D_OutputBlue = (0x0000FF & color);
191         DMA2D_InitStruct.DMA2D_OutputGreen =(0x00FF00 & color) >> 8;
192         DMA2D_InitStruct.DMA2D_OutputRed  = (0xFF0000 & color) >> 16;
193     }
194     else if(PIXEL_FORMAT == LTDC_Pixelformat_ARGB8888){
195         DMA2D_InitStruct.DMA2D_CMode = DMA2D_ARGB8888;
196         DMA2D_InitStruct.DMA2D_OutputBlue = (0x000000FF & color);
197         DMA2D_InitStruct.DMA2D_OutputGreen =(0x0000FF00 & color) >> 8;
198         DMA2D_InitStruct.DMA2D_OutputRed  = (0x00FF0000 & color) >> 16;
199         DMA2D_InitStruct.DMA2D_OutputAlpha =(0xFF000000 & color) >> 24;
200     }
201     DMA2D_InitStruct.DMA2D_OutputMemoryAdd = LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE;
202     DMA2D_InitStruct.DMA2D_OutputOffset = 0;
203     DMA2D_InitStruct.DMA2D_NumberOfLine = LCD_Y_PIXEL;
204     DMA2D_InitStruct.DMA2D_PixelPerLine = LCD_X_PIXEL;
205
206     DMA2D_InitAndTransfer();
207 }
```

This function fills the whole screen with a single color. The *color* parameter is a 32-bit integer in an ARGB-format (alpha value, red, green, blue). With *layer* and *buffer* you specify which buffer of which layer you want to fill with the color. The parameters can be *Layer_1* or *Layer_2* and *Buffer_1* or *Buffer_2*. Those are macros which represent the number 0 or 1. You can see, how these parameters affect the output memory address in line 201.

2.3 Function: DMA2D_Draw_FilledRectangle

```
390 void DMA2D_Draw_FilledRectangle(uint16_t xpos, uint16_t ypos, uint16_t length, uint16_t height, uint32_t color, Layer_e layer, Buffer_e buffer){
391     if(xpos >= LCD_X_PIXEL || ypos >= LCD_Y_PIXEL)
392         return;
393     DMA2D_InitStruct.DMA2D_Mode = DMA2D_R2M;
394     if(PIXEL_FORMAT == LTDC_Pixelformat_RGB565){
395         uint16_t color16 = (uint16_t) color;
396         DMA2D_InitStruct.DMA2D_CMode = DMA2D_RGB565;
397         DMA2D_InitStruct.DMA2D_OutputBlue = (0x001F & color16);
398         DMA2D_InitStruct.DMA2D_OutputGreen =(0x07E0 & color16) >> 5;
399         DMA2D_InitStruct.DMA2D_OutputRed  = (0xF800 & color16) >> 11;
400     }
401     else if(PIXEL_FORMAT == LTDC_Pixelformat_RGB888){
402         DMA2D_InitStruct.DMA2D_CMode = DMA2D_RGB888;
403         DMA2D_InitStruct.DMA2D_OutputBlue = (0x0000FF & color);
404         DMA2D_InitStruct.DMA2D_OutputGreen =(0x00FF00 & color) >> 8;
405         DMA2D_InitStruct.DMA2D_OutputRed  = (0xFF0000 & color) >> 16;
406     }
407     else if(PIXEL_FORMAT == LTDC_Pixelformat_ARGB8888){
408         DMA2D_InitStruct.DMA2D_CMode = DMA2D_ARGB8888;
409         DMA2D_InitStruct.DMA2D_OutputBlue = (0x000000FF & color);
410         DMA2D_InitStruct.DMA2D_OutputGreen =(0x0000FF00 & color) >> 8;
411         DMA2D_InitStruct.DMA2D_OutputRed  = (0x00FF0000 & color) >> 16;
412         DMA2D_InitStruct.DMA2D_OutputAlpha =(0xFF000000 & color) >> 24;
413     }
414     DMA2D_InitStruct.DMA2D_OutputMemoryAdd = LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + xpos * LCD_BPP;
415     if(xpos + length > LCD_X_PIXEL) length = LCD_X_PIXEL - xpos;
416     DMA2D_InitStruct.DMA2D_OutputOffset = LCD_X_PIXEL - length;
417     if(ypos + height > LCD_Y_PIXEL) height = LCD_Y_PIXEL - height;
418     DMA2D_InitStruct.DMA2D_NumberOfLine = height;
419     DMA2D_InitStruct.DMA2D_PixelPerLine = length;
420
421     DMA2D_InitAndTransfer();
422 }
```

This function draws a filled rectangle with a given width, height and color at a specified position. You may notice, that only those pixels which belong to the rectangle are being changed, while the rest of the display will still be filled with the previous image. This is because of how the x- and y-position are used to calculate the output memory address. Additionally, we use the length of the rectangle to determine the number of pixels, that shall be filled, in every line and the offset between two consecutive lines. The height of the rectangle is used for the number of lines, which shall be drawn.

2.4 Function: DMA2D_write_string

```
uint16_t DMA2D_write_string(char* string, uint16_t xpos, uint16_t ypos, uint32_t color, tFont* font, Layer_e layer, Buffer_e buffer, Layer_e bglayer, Buffer_e bgbuffer){
    uint16_t x_offset = 0, x_add;
    while(*string != '\0'){
        x_add = DMA2D_write_char(*string, xpos + x_offset, ypos, color, font, layer, buffer, bglayer, bgbuffer);
        if(x_add < 0) return x_offset;
        x_offset += x_add;
        string++;
    }
    return x_offset;
}
```

This function writes a string. You must give the position, the color and the font for the text you want to write. Since the letters have a transparent background, you can use another layer and another buffer to change the background of your text. The font is stored in a *tFont* struct. This struct contains an array of addresses to another array, which stores each pixel of the letter to be drawn. You can see the structure of those arrays in *src/fonts/Courier_New.c*. In this example package we provide the font Courier New in normal and italic with a size of 16 pixels. If you want to use another font family or another size you have to generate your own font files. Fortunately, you don't have to do it all by yourself. We recommend the free tool "lcd-image-converter" by riuson (https://sourceforge.net/projects/lcd-image-converter/), which can automatically generate a C file with the chosen font.

Note, how the wanted *tFont* struct is declared as *extern* on the top of the main file.

2.5 Drawing Lines

Our driver also implements three functions to draw a line.

- DMA2D_Draw_Y_Line
- DMA2D_Draw_X_Line
- DMA2D_Draw_Line

The first two functions are basically the same as *DMA2D_Draw_FilledRectangle* since horizontal and vertical lines are like thin rectangles. The last function (*DMA2D_Draw_Line*) draws a line between any start- and endpoint.

2.6 Function: DMA2D_Draw_Image

```
void DMA2D_Draw_Image(uint16_t xpos, uint16_t ypos, uint16_t xsize, uint16_t ysize, uint8_t alpha, uint32_t alpha_mode, uint32_t addr_image, uint32_t source_format, Layer_e layer, Buffer_e buffer){
    if(xpos >= LCD_X_PIXEL || ypos >= LCD_Y_PIXEL) return;

    DMA2D_InitStruct.DMA2D_Mode = DMA2D_M2M_BLEND;
    if(PIXEL_FORMAT == LTDC_Pixelformat_RGB565){
        DMA2D_InitStruct.DMA2D_CMode = DMA2D_RGB565;
        DMA2D_BGInitStruct.DMA2D_BGCM = CM_RGB565;
    }
    else if(PIXEL_FORMAT == LTDC_Pixelformat_RGB888){
        DMA2D_InitStruct.DMA2D_CMode = DMA2D_RGB888;
        DMA2D_BGInitStruct.DMA2D_BGCM = CM_RGB888;
    }
    else if(PIXEL_FORMAT == LTDC_Pixelformat_ARGB8888){
        DMA2D_InitStruct.DMA2D_CMode = DMA2D_ARGB8888;
        DMA2D_BGInitStruct.DMA2D_BGCM = CM_ARGB8888;
    }

    DMA2D_InitStruct.DMA2D_NumberOfLine = ysize;
    DMA2D_InitStruct.DMA2D_PixelPerLine = xsize;
    DMA2D_InitStruct.DMA2D_OutputOffset = LCD_X_PIXEL-xsize;
    DMA2D_InitStruct.DMA2D_OutputMemoryAdd = LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + xpos * LCD_BPP;

    DMA2D_FGInitStruct.DMA2D_FGCM = source_format;
    DMA2D_FGInitStruct.DMA2D_FGMA = addr_image;
    DMA2D_FGInitStruct.DMA2D_FGO = 0;
    //REPLACE_ALPHA_VALUE -> Replaces alpha value of image with parameter alpha;
    //COMBINE_ALPHA_VALUE -> Combines alpha value from image with parameter alpha;
    //NO_MODIF_ALPHA_VALUE -> Keeps alpha value from image, ignores parameter alpha;
    DMA2D_FGInitStruct.DMA2D_FGPFC_ALPHA_MODE = alpha_mode;
    DMA2D_FGInitStruct.DMA2D_FGPFC_ALPHA_VALUE = alpha;

    DMA2D_BGInitStruct.DMA2D_BGMA = LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + xpos * LCD_BPP;
    DMA2D_BGInitStruct.DMA2D_BGO = LCD_X_PIXEL-xsize;

    DMA2D_InitAndTransfer();
}
```

The last function, we want to introduce to you, is the function *DMA2D_Draw_Image* which can be used to draw any pixel image. Similar to fonts, the pixel image must be stored in an array of (A)RGB-pixels. With *alpha* and *alpha_mode* you can individually change the alpha value for a whole image. When you choose the alpha mode *REPLACE_ALPHA_VALUE*, the alpha values for each pixel in your image will be replaced with the parameter *alpha*. If you want to

keep the alpha values of your image file, you need to choose the mode *NO_MODIF_ALPHA_VALUE*. The third mode, *COMBINE_ALPHA_VALUE*, replaces all alpha values in the image with the original value multiplied with the parameter *alpha* divided by 255. The pixel array of the image and the dimensions of the image are stored in a *tImage* struct. The according C file can again be generated with the free tool “lcd-image-converter”. Take care, that the size of the image you want to draw doesn’t exceed the dimensions of your display.
If you have an image which matches the dimensions of the display exactly, you can use the function *DMA2D_Fill_Image()* instead, as this function won’t need the wanted position and the size of your image.

3. Ideas for Exercise Project

Here, we want to give you some suggestions how you could modify our example code and make your own little project.

After reading the guide for our example package *Touch Input and UART* you should know how to react on a touch input. You could try to write a code, which generates a blank canvas. And when you touch the display, an icon like our smiley appears on the touch location.