Topic:	Drawing Text and Images CubeIDE
Author:	MIGA
Date:	10.11.2021

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_cubeide_gettingstarted”. Make sure you have read this document beforehand and executed all the steps to configure your STM32CubeIDE.

For a more detailed explanation of how the microcontroller works, please refer to the STM32F429 Reference Manual (RM0090) by STMicroelectronics.

1. Introduction

In this example package we will use basic functions to draw different items on the display and use the touch input controller event to draw icons on the position we touched 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

```
/* USER CODE BEGIN 2 */
/* Declaration of variables */
Buffer_e activeBuffer;
TouchXY_t *txy;
uint8_t state = RELEASE;

/* Backlight enable */
HAL_GPIO_WritePin(GPIOG, GPIO_PIN_9, GPIO_PIN_SET);
if (BACKLIGHT_EN) {
      HAL_Delay(100);
      HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_SET);
  }
/* Display the logos and save active buffer */
activeBuffer = Display_Startup_Sequence();

/* Paint the background */
DMA2D_Fill_Color(0xFF0066FF, Layer_1, activeBuffer, false);
  DMA2D_Draw_FilledRectangle(0xFF00FF00, 48, 30, 50, 30, Layer_1, activeBuffer, false);
  DMA2D_write_string("Hello World!", 25, 20, 0x00FF0000, &courier_new , Layer_1, activeBuffer, false);
  DMA2D_write_string("I'm italic.", 25, 50, 0x00FF0000, &courier_new_itallic , Layer_1, activeBuffer, false);
  DMA2D_Draw_Y_Line(20, 6, 75, 1, 0xFF000000, Layer_1, activeBuffer);
  DMA2D_Draw_X_Line(5, 45, 210, 1, 0xFF000000, Layer_1, activeBuffer);
  DMA2D_Draw_Line(0, 240, 320, 0, 0xFF00FFFF, Layer_1, activeBuffer);
  DMA2D_Draw_Image(200, 100, 100, 100, 0xFF, DMA2D_NO_MODIF_ALPHA, (uint32_t)image_data_smiley_100px, CM_ARGB8888, Layer_1, activeBuffer, false);

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
  /* Get touch structure address */
  txy = return_touch_struct();
while (1)
{
  /* Check if user touched the display */
  if(txy->event == TOUCH){
          if(state != TOUCH){
              state = TOUCH;
              /* Check if in edge area, if true, change position of smiley */
              if(txy->xPos < 51){
                  txy->xPos = 51;
              }
              if(txy->yPos < 51){
                  txy->yPos=51;
              }
              if(txy->xPos > (LCD_X_PIXEL-51)){
                  txy->xPos = LCD_X_PIXEL-51;
              }
              if(txy->yPos > (LCD_Y_PIXEL-51)){
                  txy->yPos= LCD_Y_PIXEL-51;
              }
              /* Draw smiley around touch event */
              DMA2D_Draw_Image(txy->xPos - 50, txy->yPos - 50, 100, 100, 0x00, DMA2D_NO_MODIF_ALPHA, (uint32_t)image_data_smiley_100px, CM_ARGB8888,
                      Layer_1, activeBuffer, false);
          }
      }
      /* Check if user released the display */
      else if (txy->event == RELEASE) {
          if(state != RELEASE) {
              state = RELEASE;
          }
      }
  /* USER CODE END WHILE */
```

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 hardware abstraction layer (HAL) and the system clocks are initialized. Then all the peripherals are initialized. A few variables are declared for later use.

The 3.5- and 5.0-inch displays need the pin PH6 to be set to high in order to enable the backlight. Those two displays are recognized by the define BACKLIGHT_EN which is defined in the global_Display_Touch_HAL.h file. After that, the display startup sequence, consisting of filling the buffers/layers with the color white and displaying our 2 logos, is started.

Following the short delay of the display sequence the active display buffer is filled with the color 0xFF0066FF (ARGB8888) as background. Then a few elements are drawn on the display. Besides a rectangle, few vertical, horizontal and diagonal lines, two lines of text a smiley face is drawn on the display.

After the canvas is filled with the elements and before the main while-loop starts the main function retrieves the address (pointer) of the Touch Event structure, which can be used to determine touch events on the display area. Now the while loop starts and every time the user touches the display, a smiley is drawn around the touchpoint.

2.2 Function: DMA2D_Fill_Color

```
113 void DMA2D_Fill_Color(uint32_t color, Layer_e layer, Buffer_e buffer, bool waitForVsync){
114   uint32_t clr;
115   hdma2d.Instance = DMA2D;
116   hdma2d.Init.Mode = DMA2D_R2M;
117   if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB565){
118     hdma2d.Init.ColorMode = DMA2D_RGB565;
119     clr =   0xff << 24                              |
120             ((color >> 10) & 0x1f) * 0xff / 0x1f  << 16 |
121             ((color >> 4) & 0x3f) * 0xff / 0x3f  << 8   |
122             (color & 0x1f) * 0xff / 0x1f;
123   }
124   else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB888){
125     hdma2d.Init.ColorMode = DMA2D_RGB888;
126     clr = (0xff*color+127)/255;
127   }
128   else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB1555){
129     hdma2d.Init.ColorMode = DMA2D_ARGB1555;
130     clr =   ((color >>15) * 255) << 24              |
131             ((color >> 9) & 0x1f) * 0xff / 0x1f  << 16  |
132             ((color >> 4) & 0x1f) * 0xff / 0x1f  << 8   |
133             (color & 0x1f) * 0xff / 0x1f;
134   }
135   else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB8888){
136     hdma2d.Init.ColorMode = DMA2D_ARGB8888;
137     clr=color;
138   }
139
140   hdma2d.Init.OutputOffset = 0;
141
142   HAL_DMA2D_DeInit(&hdma2d);
143   HAL_DMA2D_Init(&hdma2d);
144   if(waitForVsync){
145   while(hltdc.Instance->CDSR & LTDC_CDSR_HSYNCS);
146   }
147   HAL_DMA2D_Start(&hdma2d, clr, LCD_START_ADDR + LCD_COUNT_BUFFER*layer*LCD_BUFFER_SIZE+buffer*LCD_BUFFER_SIZE, LCD_X_PIXEL, LCD_Y_PIXEL);
148   while(hdma2d.Instance->CR & DMA2D_CR_START);
149 }
```

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 147. The waitForVsync value, determines whether to wait for vertical sync of the display or not. If true, the display waits until the LTDC_CDSR_HSYNC bit is in reset state, which signals the start of the vertical sync.

2.3 Function: DMA2D_Draw_FilledRectangle

```
151 void DMA2D_Draw_FilledRectangle(uint32_t color, uint16_t xpos, uint16_t ypos, uint16_t width, uint16_t height, Layer_e layer, Buffer_e buffer, bool waitForVsync){
152   uint32_t clr;
153   hdma2d.Instance = DMA2D;
154   hdma2d.Init.Mode = DMA2D_R2M;
155   if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB565){
156     hdma2d.Init.ColorMode = DMA2D_RGB565;
157     clr =   0xff << 24              |
158             ((color >> 10) & 0x1f) * 0xff / 0x1f  << 16 |
159             ((color >> 4) & 0x3f) * 0xff / 0x3f  << 8   |
160             (color & 0x1f) * 0xff / 0x1f;
161   }
162   else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB888){
163     hdma2d.Init.ColorMode = DMA2D_RGB888;
164     clr = (0xff*color+127)/255;
165   }
166   else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB1555){
167     hdma2d.Init.ColorMode = DMA2D_ARGB1555;
168     clr =   ((color >>15) * 255) << 24              |
169             ((color >> 9) & 0x1f) * 0xff / 0x1f  << 16  |
170             ((color >> 4) & 0x1f) * 0xff / 0x1f  << 8   |
171             (color & 0x1f) * 0xff / 0x1f;
172   }
173   else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB8888){
174     hdma2d.Init.ColorMode = DMA2D_ARGB8888;
175     clr=color;
176   }
177
178   hdma2d.Init.OutputOffset = LCD_X_PIXEL-width;
179   HAL_DMA2D_DeInit(&hdma2d);
180   HAL_DMA2D_Init(&hdma2d);
181   if(waitForVsync){
182   while(hltdc.Instance->CDSR & LTDC_CDSR_HSYNCS);
183   }
184   HAL_DMA2D_Start(&hdma2d, clr, LCD_START_ADDR+ LCD_COUNT_BUFFER*layer*LCD_BUFFER_SIZE+buffer*LCD_BUFFER_SIZE +xpos*LCD_BPP + ypos*LCD_X_PIXEL*LCD_BPP, width, height);
185   HAL_DMA2D_PollForTransfer(&hdma2d, 10);
186 }
187
```

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. The waitForVsync value, determines whether to wait for vertical sync of the display or not. If true, the display waits until the LTDC_CDSR_HSYNC bit is in reset state, which signals the start of the vertical sync.

2.4 Function: DMA2D_write_string

```
void DMA2D_write_string(char* string, uint16_t xpos, uint16_t ypos, uint32_t color, tFont* font, Layer_e layer, Buffer_e buffer, bool waitForVsync){
  uint16_t x_offset = 0, x_add;
  uint32_t clr;
  hdma2d.Instance = DMA2D;
  hdma2d.Init.Mode = DMA2D_M2M_BLEND;
  if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB565){
    hdma2d.Init.ColorMode = DMA2D_RGB565;
    clr =   0xff << 24              |
            ((color >> 10) & 0x1f) * 0xff / 0x1f  << 16 |
            ((color >> 4) & 0x3f) * 0xff / 0x3f  << 8   |
            (color & 0x1f) * 0xff / 0x1f;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_RGB565;
  }
  else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB888){
    hdma2d.Init.ColorMode = DMA2D_RGB888;
    clr = (0xff*color+127)/255;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_RGB888;
  }
  else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB1555){
    hdma2d.Init.ColorMode = DMA2D_ARGB1555;
    clr =   ((color >>15) * 255) << 24              |
            ((color >> 9) & 0x1f) * 0xff / 0x1f  << 16  |
            ((color >> 4) & 0x1f) * 0xff / 0x1f  << 8   |
            (color & 0x1f) * 0xff / 0x1f;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_ARGB1555;
  }
  else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB8888){
    hdma2d.Init.ColorMode = DMA2D_ARGB8888;
    clr=color;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_ARGB8888;
  }
  hdma2d.LayerCfg[1].InputOffset = 0;
  hdma2d.LayerCfg[1].InputColorMode = DMA2D_INPUT_A8;
  hdma2d.LayerCfg[1].AlphaMode = DMA2D_NO_MODIF_ALPHA;
  hdma2d.LayerCfg[1].InputAlpha = clr;
  hdma2d.LayerCfg[0].AlphaMode = DMA2D_NO_MODIF_ALPHA;
  while(*string != '\0'){
    if(xpos >= LCD_X_PIXEL || ypos >= LCD_Y_PIXEL) return;
    hdma2d.Init.OutputOffset = LCD_X_PIXEL - font->chars[*string - 0x20].image->width;
    hdma2d.LayerCfg[0].InputOffset = LCD_X_PIXEL - font->chars[*string - 0x20].image->width;
    HAL_DMA2D_DeInit(&hdma2d);
    HAL_DMA2D_Init(&hdma2d);
    HAL_DMA2D_ConfigLayer(&hdma2d, 0);
    HAL_DMA2D_ConfigLayer(&hdma2d, 1);

    if(waitForVsync){
    while(hltdc.Instance->CDSR & LTDC_CDSR_HSYNCS);
    }
    HAL_DMA2D_BlendingStart(
        &hdma2d,
        (uint32_t)font->chars[*string-0x20].image->data,
        LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + (xpos + x_offset) * LCD_BPP,
        LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + (xpos + x_offset) * LCD_BPP,
        font->chars[*string - 0x20].image->width,
        font->chars[*string - 0x20].image->height);

    while(hdma2d.Instance->CR & DMA2D_CR_START){}

    x_add = font->chars[*string- 0x20].image->width;
    if(x_add < 0) return;
    x_offset += x_add;
    string++;
  }
}

void LTDC_setLayerAlpha(Layer_e layer, uint8_t alpha){
  HAL_LTDC_SetAlpha(&hltdc, alpha, layer);
}
```

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 e.g., *src/fonts/Courier_New.c*. The waitForVsync value, determines whether to wait for vertical sync of the display or not. If true, the display waits until the LTDC_CDSR_HSYNC bit is in reset state, which signals the start of the vertical sync.

In this example package we provide the font Courier New in normal and italic with a size of 26 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: You have to include `#include "driver/d_Fonts.h"` in order to have the tFont typedef available in your font 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, uint32_t alpha, uint32_t alpha_mode, uint32_t addr_image, uint32_t source_format, Layer_e layer, Buffer_e buffer, bool waitForVsync){
  if(xpos >= LCD_X_PIXEL || ypos >= LCD_Y_PIXEL) return;

  /* Blend existing Layer contents with new image */
  hdma2d.Instance = DMA2D;
  hdma2d.Init.Mode = DMA2D_M2M_BLEND;
  if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB565){
    hdma2d.Init.ColorMode = DMA2D_OUTPUT_RGB565;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_RGB565;
  }
  else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_RGB888){
    hdma2d.Init.ColorMode = DMA2D_OUTPUT_RGB888;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_RGB888;
  }
  else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB1555){
    hdma2d.Init.ColorMode = DMA2D_ARGB1555;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_ARGB1555;
  }
  else if(PIXEL_FORMAT == LTDC_PIXEL_FORMAT_ARGB8888){
    hdma2d.Init.ColorMode = DMA2D_OUTPUT_ARGB8888;
    hdma2d.LayerCfg[0].InputColorMode = DMA2D_INPUT_ARGB8888;
  }
  hdma2d.Init.OutputOffset = LCD_X_PIXEL-xsize;

  hdma2d.LayerCfg[1].InputOffset = 0;
  hdma2d.LayerCfg[1].InputColorMode = source_format;
  hdma2d.LayerCfg[1].AlphaMode = alpha_mode;
  hdma2d.LayerCfg[1].InputAlpha = alpha;

  hdma2d.LayerCfg[0].InputOffset = LCD_X_PIXEL-xsize;
  hdma2d.LayerCfg[0].AlphaMode = DMA2D_NO_MODIF_ALPHA;
  hdma2d.LayerCfg[0].InputAlpha = 255;

  HAL_DMA2D_DeInit(&hdma2d);
  HAL_DMA2D_Init(&hdma2d);
  HAL_DMA2D_ConfigLayer(&hdma2d, 0);
  HAL_DMA2D_ConfigLayer(&hdma2d, 1);
  if(waitForVsync){
  while(hltdc.Instance->CDSR & LTDC_CDSR_HSYNCS);
  }
  HAL_DMA2D_BlendingStart(&hdma2d, addr_image,
      LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + xpos * LCD_BPP
     ,LCD_START_ADDR + LCD_COUNT_BUFFER * layer * LCD_BUFFER_SIZE + buffer * LCD_BUFFER_SIZE + ypos * LCD_X_PIXEL * LCD_BPP + xpos * LCD_BPP, xsize, ysize);
  while(hdma2d.Instance->CR & DMA2D_CR_START);
}
```

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 `DMA2D_REPLACE_ALPHA`, 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 `DMA2D_NO_MODIF_ALPHA`. The third mode, `DMA2D_COMBINE_ALPHA`, 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.
The waitForVsync value, determines whether to wait for vertical sync of the display or not. If true, the display waits until the LTDC_CDSR_HSYNC bit is in reset state, which signals the start of the vertical sync.