Getting started with LVGL libraries for LCD displays

Introduction

In the modern context of the automotive industry, it is common to develop more and more complex GUIs even for small LCD displays. To meet this need, a new component, AEK-LCD-LVGL, has been created and added to the AutoDevKit ecosystem.

This new component imports the LVGL graphics library, and it is used with the AEK-LCD-DT028V1 component to develop complex GUIs faster.

The **LVGL** (light and versatile graphics library) is a free, open-source graphics library, written in C language, providing tools to create GUIs with easy-to-use graphics, nice visual effects, and low memory occupation.

LVGL is very powerful as it contains predefined elements, such as buttons, charts, lists, sliders, and images. Creating graphics with animations, anti-aliasing, opacity, and smooth scrolling is simplified with LVGL. The library is compatible with many types of input devices, such as touchpads, mouses, keyboards, and encoders.

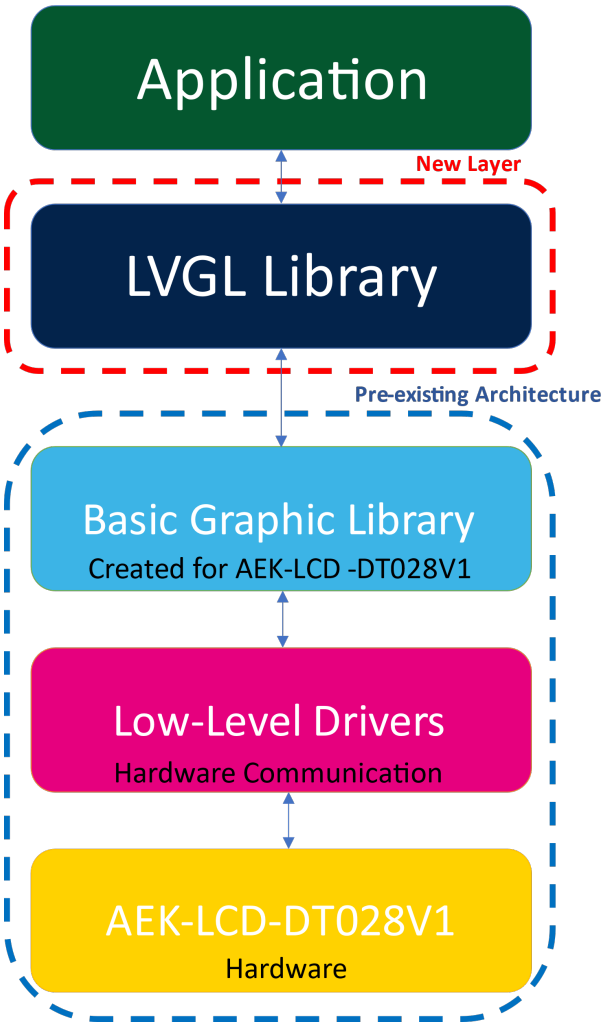
The aim of this user manual is to show how to create an LCD GUI easily, using AutoDevKit.

Note: For more details about LVGL, refer to the official documentation. The source code is available for download from GitHub.

1 AEK-LVGL architecture

The AEK-LCD-LVGL component has been implemented to extend basic graphics provided by the AEK-LCD-DT028V1 LCD touch-screen component.

Figure 1. AEK-LCD-LVGL architecture



The above image shows the LVGL software architecture integrated into AutoDevKit.

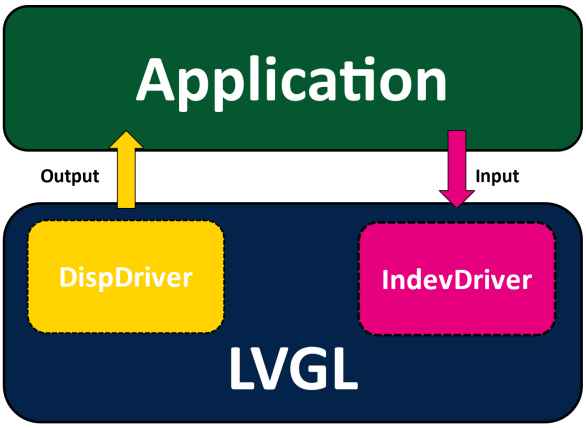
The software architecture is characterized by:

- **An LVGL library**: it implements advanced graphical functions based on the AEK-LCD-DT028V1 basic graphic library:
  - *aek_ili9341_drawPixel*: it prints pixels on the AEK-LCD-DT028V1 LCD;
  - *aek_lcd_get_touchFeedback*:it it detects touch on the AEK-LCD-DT028V1 LCD touch screen;
  - *aek_lcd_read_touchPos*: it gets the coordinates of the touched point;
  - *aek_lcd_set_touchFeedback*: it flags that the touch action is completed.
- **A basic graphic library**: it implements basic graphic functions and calls low-level driver primitives.
- **A low-level driver**: it implements MCU peripherals. In this case, the SPI protocol is used.
- **An AEK-LCD-DT028V1**: LCD evaluation board.

2 LVGL basics

The LVGL library interacts with the AEK-LCD-DT028V1 component through two drivers **DispDriver** and **IndevDriver,** as shown in the image below.

Figure 2. LVGL drivers



The **DispDriver** is in charge of preparing the buffer image and passing it to the lower layer to display it on the LCD. It uses the following *lv_disp_drv_t* typed structure:

- **draw_buf**: it points to a memory buffer structure in which the LVGL draws.
- **hor_res**: horizontal resolution of the display in pixels.
- **ver_res**: vertical resolution of the display in pixels.
- **flush_cb**: it points to the function used to print the memory buffer to the LCD display.
- **monitor_cb**: it monitors the number of pixels and the time required to display data.

On the other side, **IndevDriver** retrieves the LCD touch information coming from the lower layer. It uses the following *lv_indev_drv_t* typed structure:

- **type**: this field contains the type of the input device. Predefined available macros include:
  - LV_INDEV_TYPE_POINTER (used in our case)
  - LV_INDEV_TYPE_KEYPAD
  - LV_INDEV_TYPE_ENCODER
  - LV_INDEV_TYPE_BUTTON
- **read_cb**: it points to the function used to retrieve touch information.

flush_cb and **read_cb**: are called periodically based, respectively, on the user-defined screen refresh period and touch refresh input. The LVGL library manages refresh times through an internal clock.

Two basic LVGL functions are used for time management:

- **lv_tick_inc(uint32_t x):** the aim of this function is to synchronize the LVGL time with the physical time of the MCU. The tick update has to be set between 1 to 10 milliseconds according to LVGL specification. In our case, we set it to 5 milliseconds.
- **lv_timer_handler(void):** it updates the internal LVGL objects based on the elapsed time.

The physical time is monitored through the programmable interrupt timer (PIT) peripheral of the MCU.

3 Interface between LVGL and the AEK-LCD-DT028V1 component

The interface between AEK-LCD-LVGL and the AEK-LCD-DT028V1 component is implemented by a file named *lcd_lvgl.c* located under the “aek_lcd_lvgl_component_rla” folder.

This file contains functions to:

- initialize the LVGL library,
- manage LVGL internal timer,
- interface the LVGL library with the basic graphic library implemented by the AEK-LCD-DT028V1 component.

The five key functions are explained in the following paragraphs.

3.1 Display Init

The **aek_lcd_lvgl_display_init** function intialize the two LVGL key structures, DispDriver and IndevDriver.

3.1.1 DispDriver

The key goal of the **DispDriver** structure is to hold the drawing buffer for the LVGL.

The DispDriver **draw_buf** field points at the memory buffer structure able to contain up to two different memory buffers. The **draw_buf** field is initialized with the lv_disp_draw_buf_init() function.

Figure 3. draw_buf initialization

```
static lv_disp_draw_buf_t draw_buf;
static lv_color_t buf1[DISP_HOR_RES * DISP_VER_RES / 2];
static lv_color_t buf2[DISP_HOR_RES * DISP_VER_RES / 2];
lv_disp_draw_buf_init(&draw_buf, buf1, buf2, DISP_HOR_RES*DISP_VER_RES/2);
```

In the above code, the DISP_HOR_RES and DISP_VER_RES parameters represent the LCD dimension.

Note: The buffer size must be customized according to the system available memory. The official LVGL guide recommends choosing the size of the drawing buffers of at least 1/10 of the screen size. If a second optional buffer is used, LVGL can tap into one buffer while the data of the other buffer is sent to be displayed in background.

Figure 4. draw_buf initialization

```
static lv_disp_drv_t disp_drv;
lv_disp_drv_init(&disp_drv);
disp_drv.flush_cb = flush;
disp_drv.draw_buf = &draw_buf;
disp_drv.monitor_cb = monitor_cb;
disp_drv.hor_res = DISP_HOR_RES;
disp_drv.ver_res = DISP_VER_RES;
lv_disp_drv_register(&disp_drv);
```

The other parameters of the structure are the screen dimensions, the two functions, *flush* and *monitor_cb,* that we will analyze later. Once filled, the structure has to be registered with the dedicated lv_disp_drv_register() function to set an active display.

3.1.2 IndevDriver

The **IndevDriver** is initialized as follows:

Figure 5. draw_buf initialization

```
static lv_indev_drv_t indev_drv;
lv_indev_drv_init(&indev_drv);
indev_drv.type = LV_INDEV_TYPE_POINTER;
indev_drv.read_cb = my_input_read;
lv_indev_t * my_indev = lv_indev_drv_register(&indev_drv);
```

The key defined fields are the type of device used and the function to manage it. Also in this case, the initialized structure needs to be registered to make the device become active.

3.2 Flush

The **flush** function uses the AEK-LCD-DT028V1 component basic graphic library to draw, on the LCD, the image present in the memory buffer initialized according to the previous paragraph.

Figure 6. Flush function

```
void flush(lv_disp_drv_t* drv, const lv_area_t* area, lv_color_t*
color_p)
{
	if(area->x2 < 0) return;
	if(area->y2 < 0) return;
	if(area->x1 > DISP_HOR_RES - 1) return;
	if(area->y1 > DISP_VER_RES - 1) return;

	uint16_t x, y;
	for(y = area->y1; y <= area->y2; y++) {
		for(x = area->x1; x <= area->x2; x++) {
			aek_ili9341_drawPixel(AEK_LCD_DEV0,x, y,color_p-
	>full);
			color_p++;
		}
	}
	lv_disp_flush_ready(drv);
}
```

The *flush* function skeleton has been provided by the LVGL function and customized for the LCD screen driver in use (i.e., *aek_ili9341_drawPixel* – pixel drawing). The input parameters are:

- **drv**: the pointer to the DispDriver
- **area**: buffer that contains the specific area that needs to be updated
- **color_p**: buffer that contains the colors to be printed.

3.3 monitor_cb

The **monitor_cb** function is defined in the official LVGL guide and does not require customizations.

Figure 7. monitor_cb function

```
void monitor_cb(lv_disp_drv_t * d, uint32_t t, uint32_t p)
{
      t_saved = t;
}
```

3.4 my_input_read

The my_input_read function is in charge of managing the input coming from the LCD screen at high level. The function skeleton is defined by the LVGL library. The input parameters are:

- **drv**: pointer to the initialized input driver
- **data**: contains the pixel-converted x,y coordinate of touched points

The image below shows the implementation of the *my_input_read* function:

Figure 8. my_input_read function

```
void my_input_read(lv_indev_drv_t* drv, lv_indev_data_t* data)
{
      if(aek_lcd_get_touchFeedback(AEK_LCD_DEV0))
      {
            data->state = LV_INDEV_STATE_PRESSED;
            uint16_t x_value,y_value;
            aek_lcd_read_touch_pos(AEK_LCD_DEV0,&x_value, &y_value);
            aek_ili9341_convert(x_value,y_value, data);
            aek_lcd_set_touchFeedback(AEK_LCD_DEV0);
      }
      else
      {
            data->state = LV_INDEV_STATE_RELEASED;
      }
}
```

3.5 Refresh screen

The aek_lcd_lvgl_refresh_screen function updates the LVGL internal timers.

Note: This function has to be correctly placed in the application code to fulfill the LVGL time constraints.

Figure 9. aek_lcd_lvgl_refresh_screen function

```
int time;
void aek_lcd_lvgl_refresh_screen(void){
	if(enable_tick==1){
	lv_tick_inc(10);
	time++;
	if(time>2){
		lv_timer_handler();
		time = 0;
	}
	enable_tick=0;
	}
}
```

4 AutoDevKit ecosystem

The application development that uses the AEK-LCD-LVGL takes full advantage of the AutoDevKit ecosystem, whose basic components are:

- AutoDevKit Studio IDE installable from www.st.com/autodevkitsw
- SPC5-UDESTK debugging software for Windows or OpenOCD debugger
- AEK-LCD-LVGL drive

4.1 AutoDevKit Studio

AutoDevKit Studio (STSW-AUTODEVKIT) is an integrated development environment (IDE) based on Eclipse designed to assist the development of embedded applications based on SPC5 Power Architecture 32-bit microcontrollers.

The package includes an application wizard to initiate projects with all the relevant components and key elements required to generate the final application source code.

AutoDevKit Studio also features:

- the possibility of integrating other software products from the standard Eclipse marketplace
- free license GCC GNU C Compiler component
- support for industry-standard compilers
- support for multi-core microcontrollers
- PinMap editor to facilitate MCU pin configuration
- integrated hardware and software components, component compatibility checking, and MCU and peripheral configuration tools
- the possibility of creating new system solutions from existing ones by adding or removing compatible function boards
- new code can be generated immediately for any compatible MCU
- high-level application APIs to control each functional component, including the ones for the AEK-LCD-LVGL component.

For more information, refer to UM2623 (in particular, Section 6 and Section 7) or watch the video tutorials.

4.2 AEK_LCD_LVGL component

The AEK-LVGL drivers are provided with the STSW-AUTODEVKIT (from version 2.0.0 on) installation to facilitate the programming phase.

Update your AutoDevKit installation to get the latest version. Once properly installed, select the component named *AEK_LVGL Component RLA*.

4.2.1 AEK_LCD_LVGL component configuration

To configure the component, follow the procedure below.

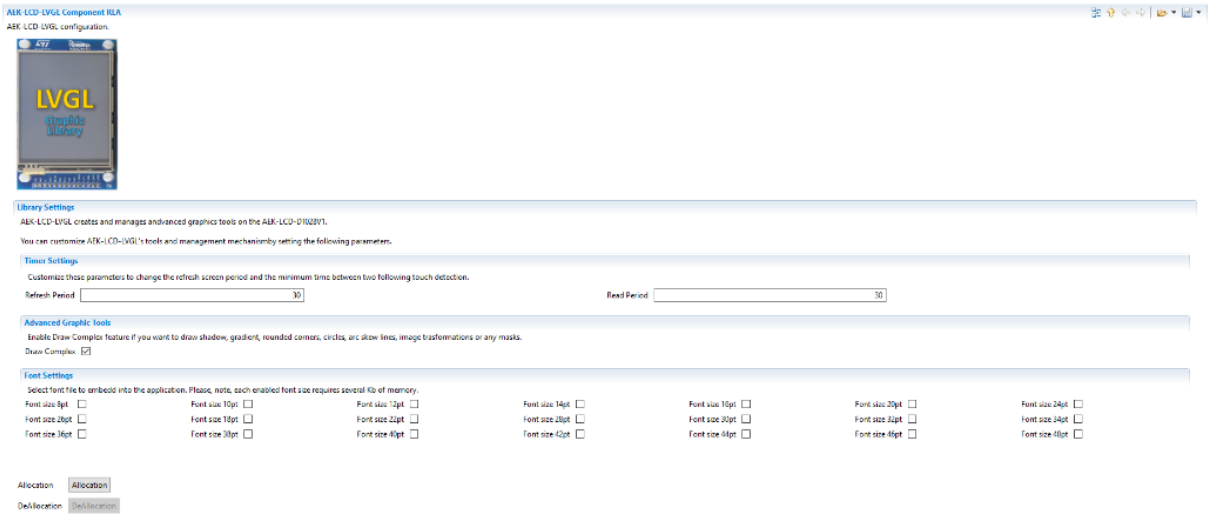
Step 1. Set the **Refr_Period** time. This is the refresh screen period (the recommended value is 30).

Step 2. Set the **Read_Period** time. This the minimum time between two following touch detections (the recommended value is 30).

Step 3. Tick the **Draw Complex** box to enable advanced widget like shadows, gradients, rounded corners, circles, arcs, skew lines, and image transformations.

Step 4. Select the **fonts** that you want to use. Consider that each font requires extra flash memory for the generated application code.

Figure 10. **AEK_LVGL Component RLA configuration**



5 How to create an AutoDevKit project with the AEK-LCD-LVGL component based on SPC58EC

The steps are:

Step 1. Create a new AutoDevKit Studio application for the SPC58EC series microcontroller and add the following components:

- SPC58ECxx Init Package Component RLA
- SPC58ECxx Low Level Drivers Component RLA

Note: Add these components at the beginning, otherwise the remaining components are not visible.

Step 2. Add the following additional components:

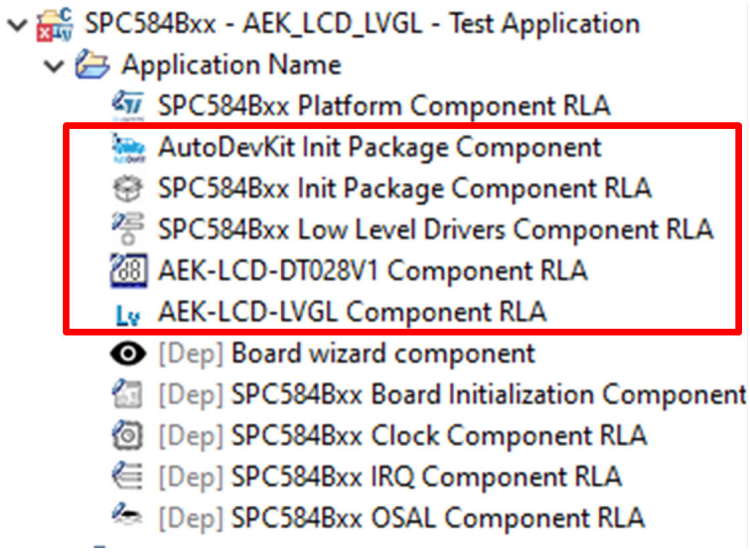
Step 2a. AutoDevKit Init Package Component

Step 2b. SPC58ECxx Platform Component RLA

Step 2c. AEK-LCD-DT028V1 Component RLA (see UM2939 for configuration)

Step 2d. AEK-LCD-LVGL Component RLA

Figure 11. Adding components



Step 3. Click the [Allocation] button in the AEK-LCD-LVGL configuration window. This operation delegates the AEK-LCD-LVGL configuration to AutoDevKit.

Step 4. The allocation has enabled the PIT timer peripheral. You can verify it in the Low-Level Driver component.

Figure 12. PIT enablement (1 of 2)

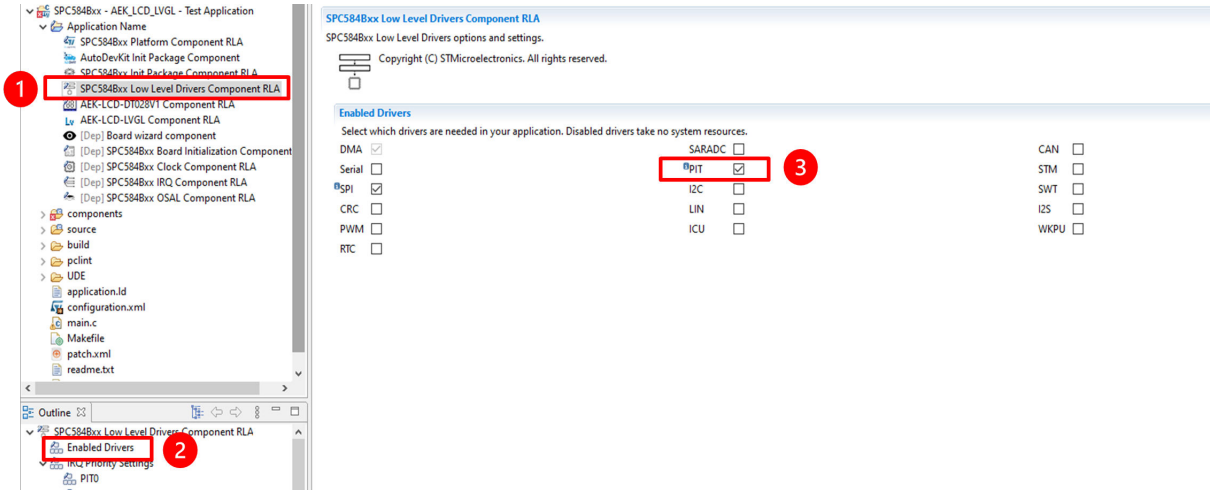
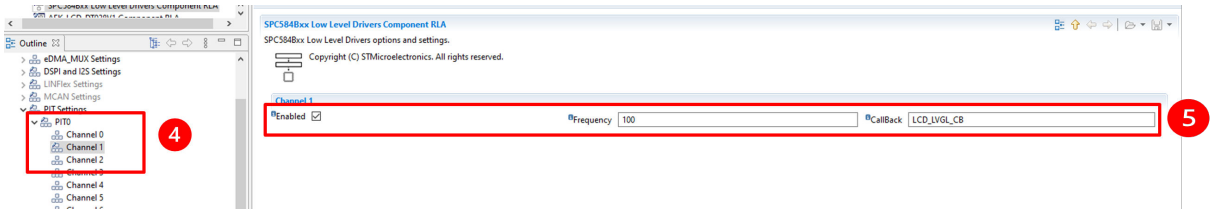


Figure 13. PIT enablement (2 of 2)



Step 5. Generate and build the application using the appropriate icons in AutoDevKit Studio. The project folder is then populated with new files, including *main.c*. The component folder is populated then with AEK-LCD-DT028V1 and AEK-LCD-LVGL drivers.

Step 6. Open the *main.c* file and include *AEK-LCD-DT028V1.h* and *AEK_LCD_LVGL.h* files.

Figure 14. *main.c* file

```
#include "components.h"
#include "aek_lcd_dt028v1.h"
#include "aek_lcd_lvgl.h"
```

Step 7. In the main.c file, after the irqIsrEnable() function, insert the following mandatory functions:

Figure 15. Mandatory functions to insert in the main.c file

```
int main(void)

{

        componentsInit();
        /* Enable Interrupts */
        irqIsrEnable();
       /* initialize AEK-LCD-DT028V1 */
        aek_ili9341_init(AEK_LCD_DEV0);
        // Init LCD and LVGL
        aek_lcd_lvgl_init(AEK_LCD_DEV0);
        //* Start PIT0 driver
        pit_lld_start(&PITD1, pit0_config);
        // Enable PIT0 Channel 1
        pit_lld_channel_start(&PITD1,PIT0_CHANNEL_CH1);
        //call simple function from the example provided by the library

        /* Application main loop.*/
        for ( ; ; )
        {
              aek_lcd_lvgl_refresh_screen();
```

1 2 3

Step 8. In the *main*.c, copy and paste an example from the LVGL library taken from the official guide and insert it in the *main()*.

Figure 16. Example from the LVGL library to insert in the main.c file

```
void lv_example_get_started_1(void)
{
    lv_obj_t * btn = lv_btn_create(lv_scr_act());                    /*Add a button the current screen*/
    lv_obj_set_pos(btn, 10, 10);                                     /*Set its position*/
    lv_obj_set_size(btn, 120, 50);                                   /*Set its size*/
    lv_obj_add_event_cb(btn, btn_event_cb, LV_EVENT_ALL, NULL);      /*Assign a callback to the button*/

    lv_obj_t * label = lv_label_create(btn);                         /*Add a label to the button*/
    lv_label_set_text(label, "Button");                             /*Set the labels text*/
    lv_obj_center(label);
}

int main(void)

{

       [. . .]                                                  /*Other code*/

       lv_example_get_started_1()

       for ( ; ; ){ [. . .] }                                   /*Other code*/
}
```

1

Step 9. Save, generate, and compile the application.

Step 10. Open the boardview editor provided by AutoDevKit This provides a graphical point-to-point guide on how to wire the boards.

Step 11. Connect the AEK-LCD-DT028V1 to a USB port on your PC using a mini-USB to USB cable.

Step 12. Launch SPC5-UDESTK-SW and open the debug.wsx file in the AEK-LCD-LVGL– Application /UDE folder.

Step 13. Run and debug your code.

6 Available demos for AEK-LVGL

There are several demos provided with the AEK-LCD-LVGL component:

- SPC582Bxx_RLA AEK_LCD_LVGL Test Application
- SPC58ECxx_RLA AEK-LCD_LVGL Test Application
- dual screen AVAS demo - SPC58ECxx_RLA_MainEcuForIntegratAVASControl - Test Application

Note: More demos might become available with new AutoDevKit releases.

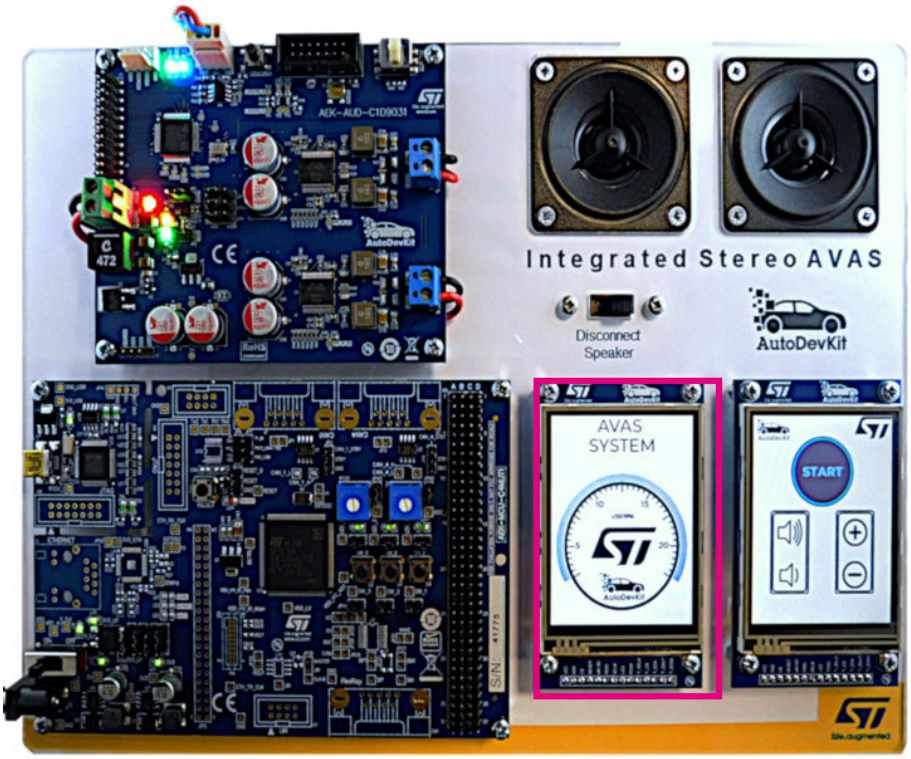
7 Advanced application example - dual screen AVAS demo

An advanced application has been implemented using LVGL. This application draws a car gauge for engine rpms in a display and manages the related gauge animations.

The implemented AVAS application is based on the AEK-AUD-C1D9031 board and simulates the car engine sound at low speeds to warn pedestrians of an electric vehicle approaching.

In the demo, we simulate the acceleration and deceleration (i.e., an increase/decrease of rpms) of a car engine and its volume through a control panel implemented on the LCD screen of the AEK-LCD-DT028V1.

Figure 17. AEKD-STEREOAVAS kit



We have extended the demo by adding a second AEK-LCD-DT028V1 LCD and using the LVGL library to create a speedometer to gauge the engine rpm values.

7.1 LVGL widgets used

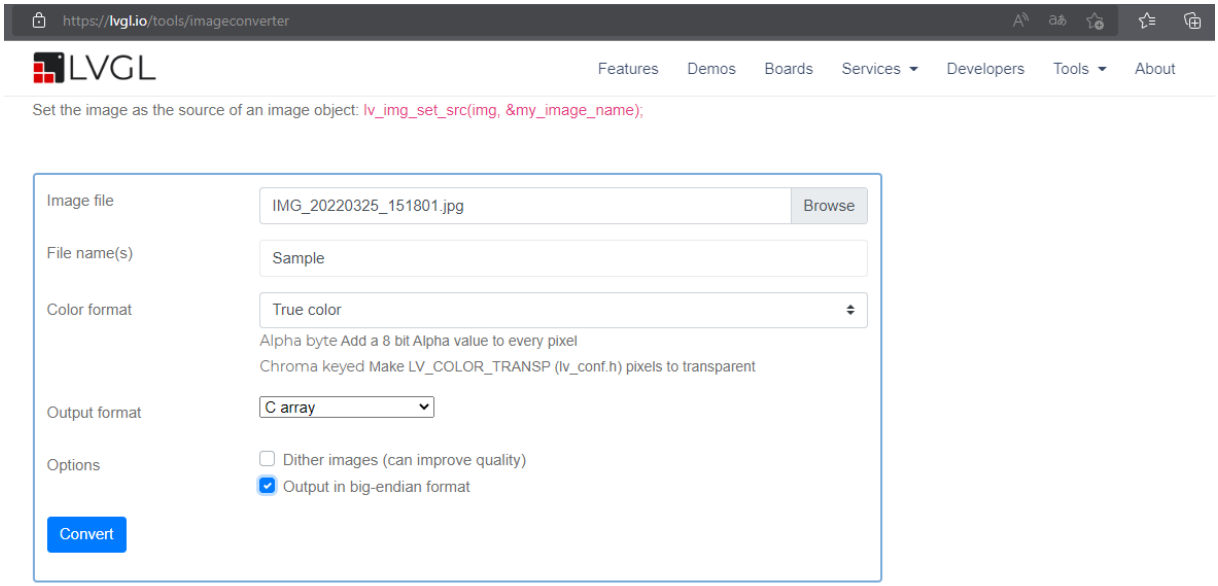
To develop the dual screen AVAS demo, we have used the following LVGL widgets:

- An **image** used as a tachometer background
- An **arc** used as a tachometer indicator
- An **animation** that updates the arc value according to the engine rpm

7.1.1 An LVGL image widget

To use an image with the LVGL library, convert it into a C array by using a free online converter.

Figure 18. **LVGL image widget**



Note: When converting the image remember to tick the box of the Big-Endian format.

In the dual screen AVAS demo, the C array representing the tachometer image has been named *Gauge*. The image widget has been customized as follows:

Figure 19. **Tachometer background image**

```
LV_IMG_DECLARE(Gauge);
lv_obj_t * img1 = lv_img_create(lv_scr_act());
lv_img_set_src(img1, &Gauge);
lv_obj_align(img1, LV_ALIGN_CENTER, 0, 40);
lv_obj_set_size(img1, 200, 200);
```

Where:

- **lv_img_declare**: is used to declare an image called Gauge.
- **lv_img_create**: is used to create an image object and attach it to the current screen.
- **lv_img_set_src**: this is the image obtained from the LVGL converter previously shown (it is recommended to use the jpg format).
- **lv_obj_align:** is used to align the image to the center with a given offset.
- **lv_obj_set_size**: is used to set the image size.

Note: For more details about how to manage an image with the LVGL library, refer to the official documentation.

7.1.2 An LVGL arc widget

A multicolored arc has been created to show the engine instantaneous rpms. The multicolored arc consists of two contiguous colors, red and blue, respectively.

Figure 20. AVAS tachometer

The following code shows how to create an arc:

Figure 21. AVAS tachometer

```
/*Create an Arc*/
lv_obj_t * arc = lv_arc_create(lv_scr_act());
lv_arc_set_rotation(arc, 90);
lv_arc_set_bg_angles(arc, 60, 240);
lv_arc_set_value(arc, 0);
lv_obj_set_size(arc, 225, 225);
lv_obj_remove_style(arc, NULL, LV_PART_KNOB);
lv_obj_clear_flag(arc, LV_OBJ_FLAG_CLICKABLE);
lv_obj_align(arc, LV_ALIGN_CENTER, 0, 40);
```

Where:

- **lv_arc_create**: creates an arc object.
- **lv_arc_set_rotation:** sets the arc rotation.
- **lv_arc_set_bg_angles**: sets the maximum and minimum arc value in degrees.
- **lv_arc_set_value**: sets the arc initial value at zero.
- **lv_obj_set_size:** sets the arc dimensions.
- **lv_obj_remove_style**: removes the arc final pointer.

- **lv_obj_clear_flag**: sets the arc as not clickable.
- **lv_obj_align**: aligns the arc to the center with a specified offset.

Note: For more details about the arc, refer to LVGL documentation.

7.1.3 Widget associated animation

A specific arc animation function is created and passed to the LVGL engine to display rpm changes.
The function code is the following:

Figure 22. arc_animation function

```
uint16_t arc_animation(lv_obj_t * arc, uint16_t delay, uint16_t start, uint16_t end,
uint32_t speed){
	//Setting animation parameters
	lv_anim_t a;
	//Delay to next step
	uint16_t animation_time = lv_anim_speed_to_time(speed, start, end);
	lv_anim_init(&a);
	lv_anim_set_var(&a, arc);
	lv_anim_set_exec_cb(&a, set_angle);
	lv_anim_set_values(&a, start, end);
	lv_anim_set_time(&a, animation_time);
	lv_anim_path_overshoot(&a);
	lv_anim_set_delay(&a, delay);
	lv_anim_start(&a);
	return animation_time;
}
```

Where:

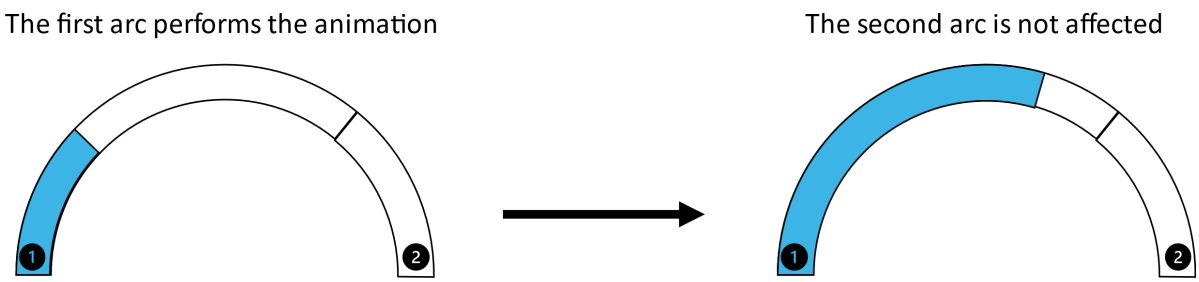
- **arc**: is the pointer to the current arc widget
- **delay**: is the delay time before the animation starts
- **start**: is the initial arc position
- **end**: is the final arc position
- **speed**: is the animation speed in unit/secs.

Note: For more details about the used animation functions, refer to LVGL documentation.

Considering that the complete arc consists of two contiguous arches, we had to manage the animation function properly. For this purpose, let us analyze two different scenarios:

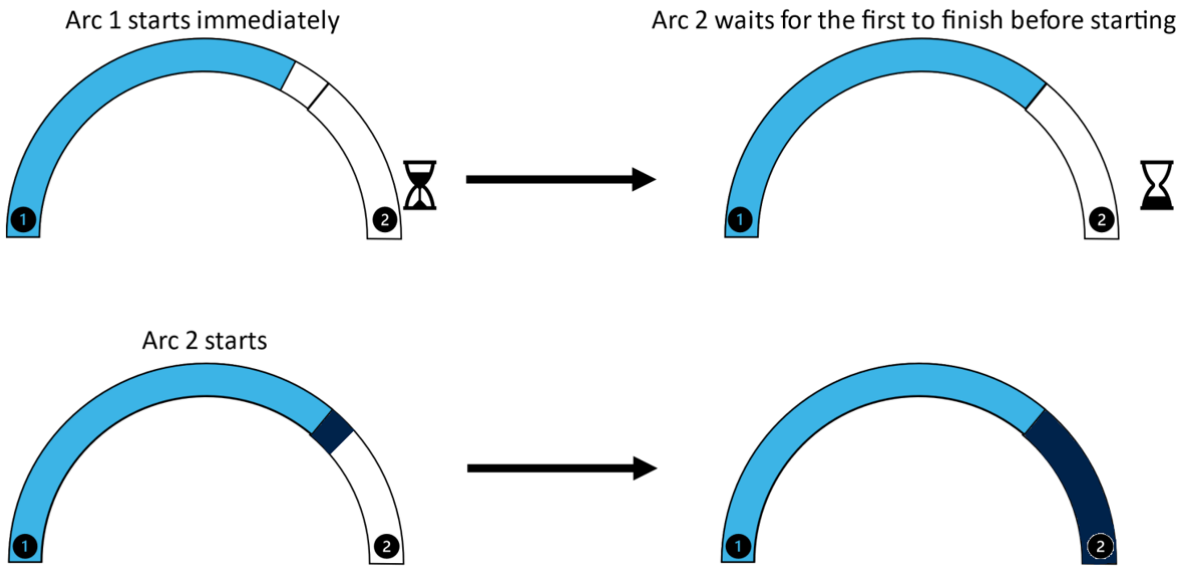
1. *Case: the animation involves one arc*
   In this simple case, we assign a single animation to the arc.

Figure 23. Arc animation - one arc



2. *Case: the animation involves two arches*
In this case, the animation of the second arc starts at the end of the animation of the first one.

Figure 24. Arc animation - two arches



A specific LVGL function (*lv_anim_speed_to_time*) computes the animation time. This execution time is used to compute the delay of the second arc animation.

Figure 25. Delay of the second arc

```
delay = arc_animation(arcB, 0, arc_blue_value, 100, speed);
arc_animation(arcR, delay, arc_red_value, 50, speed);
```

7.2 Dual core implementation

In the dual screen AVAS demo, display and audio playback tasks are concurrently executed in a real-time embedded system. To overcome a possible loss of system responsiveness, we have decided to use two different cores: one dedicated to the display and one to the audio playback.

The AEK-MCU-C4MLIT1 board hosts an SPC58EC80E5 microcontroller with a dual core processor, the best fit for the above described case.

In detail:

- **Core 2**: It is the first to start, it initializes the library and then execute the application code.
- **Core 0**: It calls the *aek_lcd_lvgl_refresh_screen()* function within the main loop, in order to update continuously the display and read the touch input.

Figure 26. SPC58EC80E5 microcontroller core initialization

```
int main(void)
{
   componentsInit();
   const FontObject_t *font_12pt = (const FontObject_t *)&DejaVuSansMono_12pt_7b;

   /* Enable Interrupts */
   irqIsrEnable();
   .
   .
   .
   .
   .

   /* Activate core0 */
   runCore0();
}

int main_core0(void){
  /* Enable Interrupts */
  irqIsrEnable();
  //Initialize LVGL
  lv_init();
  //Initialize Display and LVGL Drivers
  aek_lcd_lvgl_init(AEK_LCD_DEV1);
  //* Start PIT0 driver
  pit_lld_start(&PITD1, pit0_config);
  // Enable PIT0 Channel 1
  pit_lld_channel_start(&PITD1,PIT0_CHANNEL_CH1);

  //lv_example_arc_2();
  draw_gauge();
  arcB = arc_blue();
  arcR = arc_red();

  for(;;){
  aek_lcd_lvgl_refresh_screen();
  }
}
```

Note: The PIT functions and the aek_lcd_lvgl_refresh_screen() must be placed in the same core.

Revision history

Table 1. **Document revision history**

Date	Revision	Changes
04-Oct-2023	1	Initial release.

Contents

List of figures