# 5 Multi-label layout

## 5.1 experimental goals

In this lesson, you will learn how to draw a multi-label layout, which is divided into three layouts.

The reference code path for this experiment： CanMV\04-GUI\multi_layout.py

## 5.2 experimental procedure

The factory firmware of the module has been integrated with the lvgl graphical library. If you have downloaded other firmware, please burn it back to the factory firmware before experimenting.

1. Import the relevant libraries and initialize the lvgl, register the display interface, and register the input interface.
2. Create a new button page class, Page_Buttons, that will add buttons automatically on each click.

```
class Page_Buttons:
    def __init__(self, app, page):
        self.app = app
        self.page = page

        # counter button
        self.counter_btn = lv.btn(page)
        self.counter_btn.set_size(100,60)
        self.counter_btn.align(page, lv.ALIGN.CENTER, 0, 0)
        self.counter_label = lv.label(self.counter_btn)
        self.counter_label.set_text('Count')
        self.counter_btn.set_event_cb(self.on_counter_btn)
        self.counter = 0

    def on_counter_btn(self, obj, event):
        if event == lv.EVENT.CLICKED:
            self.counter += 1
            self.counter_label.set_text(str(self.counter))
```

3. Create a new slider page class, Page_Slider, which displays the current slider value in real time.

```
class Page_Slider:
    def __init__(self, app, page):
        self.app = app
        self.page = page

        # slider
        self.slider = lv.slider(page)
        self.slider.align(page, lv.ALIGN.CENTER, 0, -10)
        self.slider_label = lv.label(page)
        self.slider_label.align(self.slider, lv.ALIGN.OUT_LEFT_MID, -10, 0)
        self.slider.set_event_cb(self.on_slider_changed)
        self.on_slider_changed(None)

    def on_slider_changed(self, obj=None, event=-1):
        self.slider_label.set_text(str(self.slider.get_value()))
```

4. Create a new animation class, Anim.

```
class Anim(lv.anim_t):
    def __init__(self, obj, val, size, exec_cb, path_cb, time=500, playback = False,
ready_cb=None):
        super().__init__()
        lv.anim_init(self)
        lv.anim_set_time(self, time, 0)
        lv.anim_set_values(self, val, val+size)
        if callable(exec_cb):
            lv.anim_set_custom_exec_cb(self, exec_cb)
        else:
            lv.anim_set_exec_cb(self, obj, exec_cb)
        lv.anim_set_path_cb(self, path_cb )
        if playback: lv.anim_set_playback(self, 0)
        if ready_cb: lv.anim_set_ready_cb(self, ready_cb)
        lv.anim_create(self)
```

5. Create a new chart animation class, AnimatedChart.

```
class AnimatedChart(lv.chart):
    def __init__(self, parent, val, size):
        super().__init__(parent)
        self.val = val
        self.size = size
        self.max = 2000
        self.min = 500
        self.factor = 100
        self.anim_phase1()

    def anim_phase1(self):
        Anim(
            self,
            self.val,
            self.size,
            lambda a, val: self.set_range(0, val),
            lv.anim_path_ease_in,
            ready_cb=lambda a:self.anim_phase2(),
            time=(self.max * self.factor) // 100)

    def anim_phase2(self):
        Anim(
            self,
            self.val+self.size,
            -self.size,
            lambda a, val: self.set_range(0, val),
            lv.anim_path_ease_out,
            ready_cb=lambda a:self.anim_phase1(),
            time=(self.min * self.factor) // 100)
```

6. Create a new chart animation page class Page_Chart, the main function is to display chart animation effects.

```
class Page_Chart():
    def __init__(self, app, page):
        self.app = app
        self.page = page
        self.chart = AnimatedChart(page, 100, 1000)
        self.chart.set_width(page.get_width() - 100)
        self.chart.set_height(page.get_height() - 30)
        self.chart.align(page, lv.ALIGN.CENTER, 0, 0)
        self.series1 = self.chart.add_series(lv.color_hex(0xFF0000))
        self.chart.set_type(self.chart.TYPE.POINT | self.chart.TYPE.LINE)
        self.chart.set_series_width(3)
        self.chart.set_range(0,100)
        self.chart.init_points(self.series1, 10)
        self.chart.set_points(self.series1, [10,20,30,20,10,40,50,80,95,80])
        self.chart.set_x_tick_texts('a\nb\nc\nd\ne', 2,
lv.chart.AXIS.DRAW_LAST_TICK)
        self.chart.set_x_tick_length(10, 5)
        self.chart.set_y_tick_texts('1\n2\n3\n4\n5', 2,
lv.chart.AXIS.DRAW_LAST_TICK)
        self.chart.set_y_tick_length(10, 5)
        self.chart.set_div_line_count(3, 3)
        self.chart.set_margin(30)

        self.slider = lv.slider(page)
        self.slider.align(self.chart, lv.ALIGN.OUT_RIGHT_TOP, 10, 0)
        self.slider.set_width(30)
        self.slider.set_height(self.chart.get_height())
        self.slider.set_range(10, 200)
        self.slider.set_value(self.chart.factor, 0)
        self.slider.set_event_cb(self.on_slider_changed)

    # Create a slider that controls the chart animation speed
    def on_slider_changed(self, obj=None, event=-1):
        self.chart.factor = self.slider.get_value()
```

7. New main page class Screen_Main, the main function is to use the tabview control to manage multiple pages.

```
class Screen_Main(lv.obj):
    def __init__(self, app, *args, **kwds):
        self.app = app
        super().__init__(*args, **kwds)

        self.tabview = lv.tabview(self)
        self.tabview.set_style(lv.tabview.STYLE.BG, lv.style_plain_color)
        self.page_buttons = Page_Buttons(self.app, self.tabview.add_tab('Button'))
        self.page_slider = Page_Slider(self.app, self.tabview.add_tab('Slider'))
        self.page_chart = Page_Chart(self.app, self.tabview.add_tab('Chart'))
```

8. Initialize the main screen and load all views.

```
screen_main = Screen_Main(lv.obj())
lv.scr_load(screen_main)
```

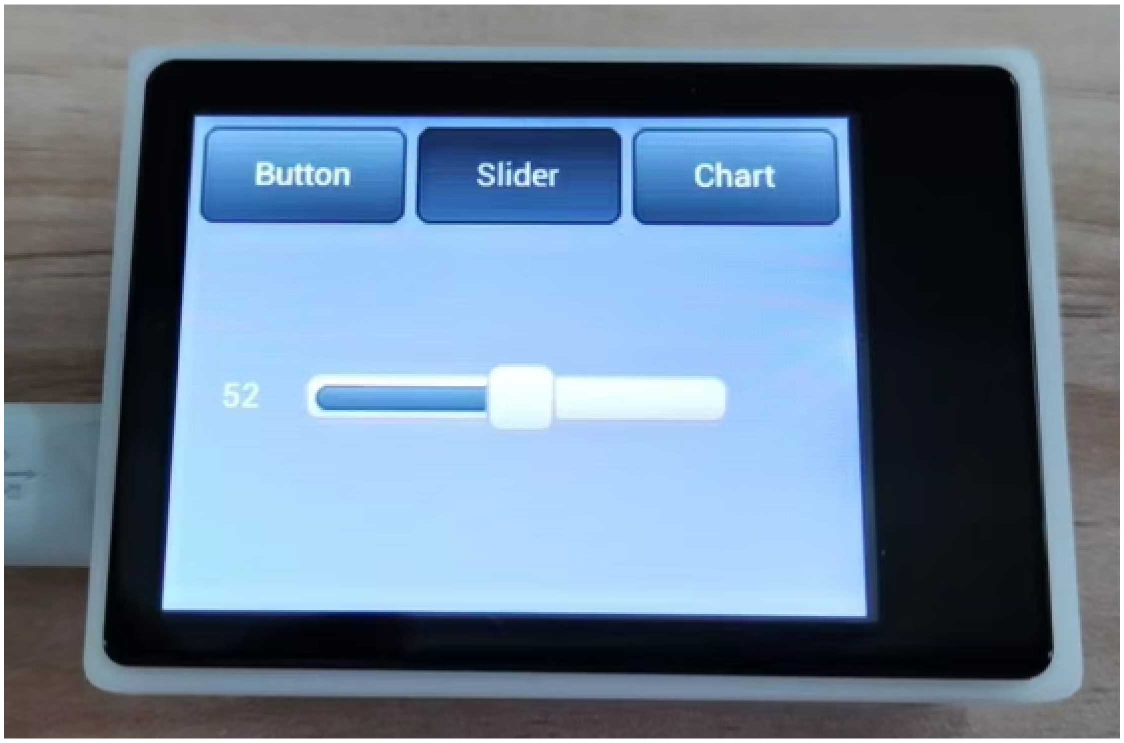
9. Since the images of lvgl need to be updated in real time, it is necessary to refresh the tasks of lvgl every 5 ms.

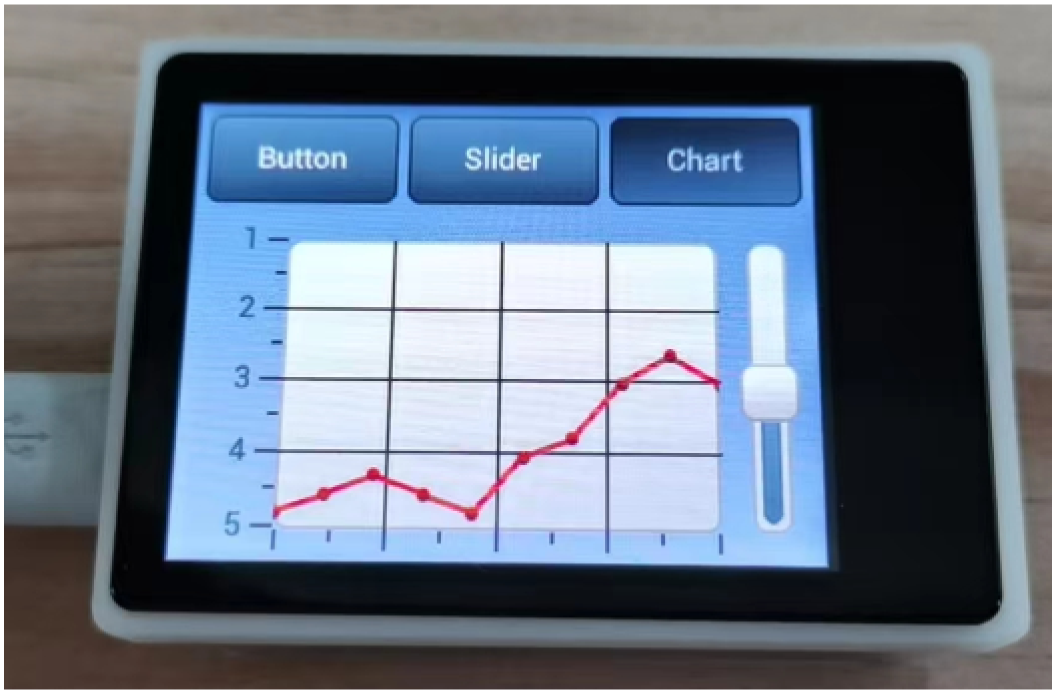
```
tim = time.ticks_ms()
while True:
    if time.ticks_ms()-tim > 5:
        tim = time.ticks_ms()
        lv.task_handler()
        lv.tick_inc(5)
```

## 5.3 experimental results

Connect the K210 module to the computer through the microUSB data cable, CanMV IDE click the connect button, after the connection is completed click the Run button to run the routine code. You can also download the code as main.py and run it in the K210 module.

It can be seen that three buttons are displayed on the top of the LCD display, among which the "Button" represents the switch button function page, the "Slider" button represents the switch to the slider function page, and the "Chart" button represents the switch to the dynamic chart function page. The corresponding page function can be entered by touching the corresponding button, and the function of the three pages is similar to the function of the previous routine.





## 5.4 experiment summary

lvgl can not only be drawn graphical buttons, sliders, etc., these may also be the layout of the collection together, divided into a plurality of interface display, such a production function that is beautiful and practical.