AN13217

GUI Guider Peripheral Interaction

Rev. 0 — 18 June 2021 Application Note

Contents

1 Introduction....................................1
2 GUI Guider overview.......................1
3 Adding custom code........................4
4 Software...6
5 Conclusion....................................10
6 Revision history............................10

1 Introduction

The GUI Guider, by NXP, is a powerful development tool when creating graphical user interface applications. It allows for quick development of graphical displays. Often, however, applications have to do more than just operate a display. The controller has to run a display and take input from a keyboard, or buttons, or read information from sensors, and so on. Controls on the display must interact with these other functions of the controller. This application note is intended to show how to get your GUI Guider application to interact with other peripherals in your application.

2 GUI Guider overview

GUI Guider is a What You See Is What You Get (WYSIWYG) user interface design tool with drag and drop support. This tool is a complementary tool from NXP that supports NXP devices. It allows for adding events, actions, and animations, which are useful when allowing the target MCU to interact with the outside world. See Figure 1 for the main interface of the GUI Guider.

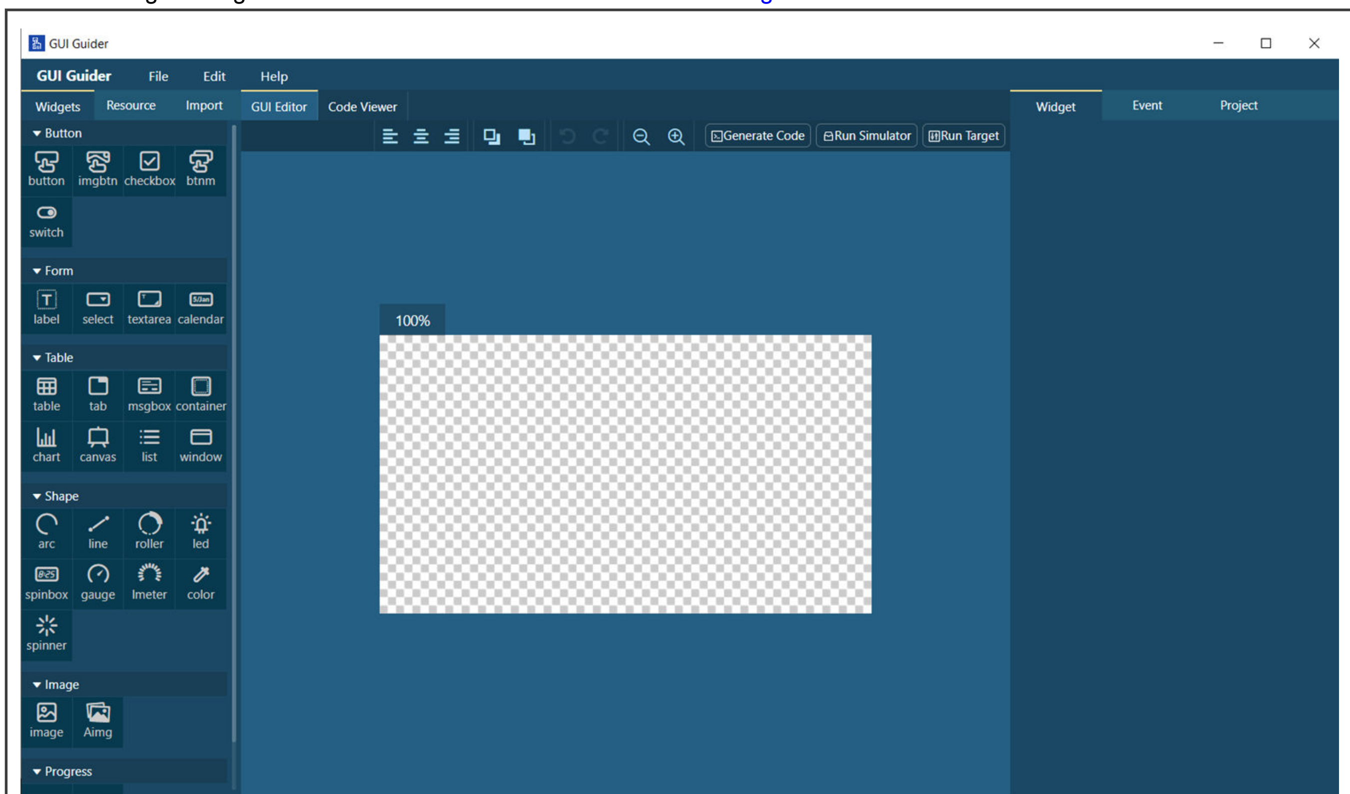


Figure 1. GUI Guider main interface

In the main interface, there is a left panel, middle panel, and right panel. The focus of this application note to give the detail about the right panel, as that is where the events are added.

To demonstrate how GUI Guider can interact with peripherals, we use the buttonCounterDemo application template. After creating a demo based on this template, screen shown in Figure 2 should appear.

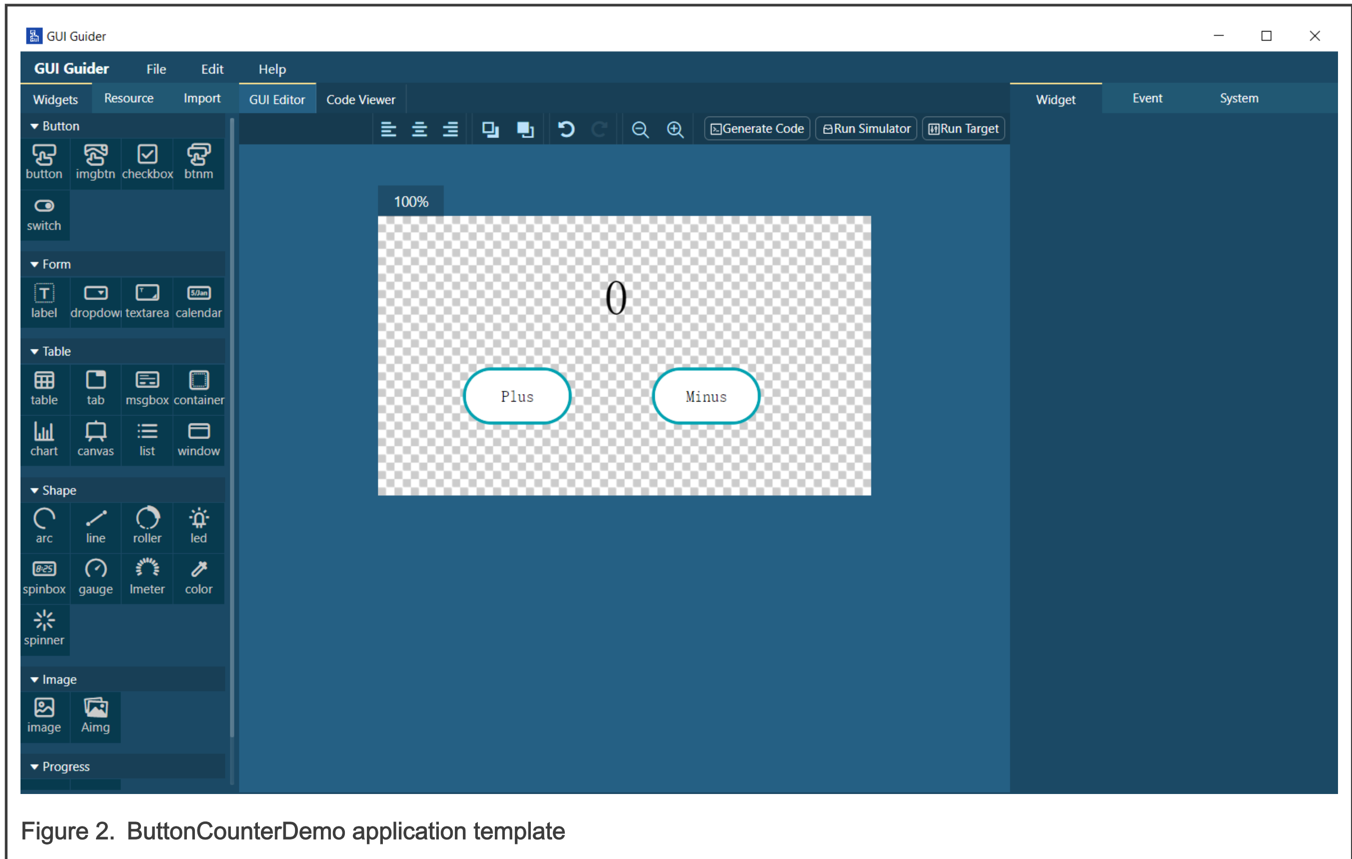


Figure 2. ButtonCounterDemo application template

Now, let us first inspect one of the button widgets. See Figure 3 for the events associated with the minus button.

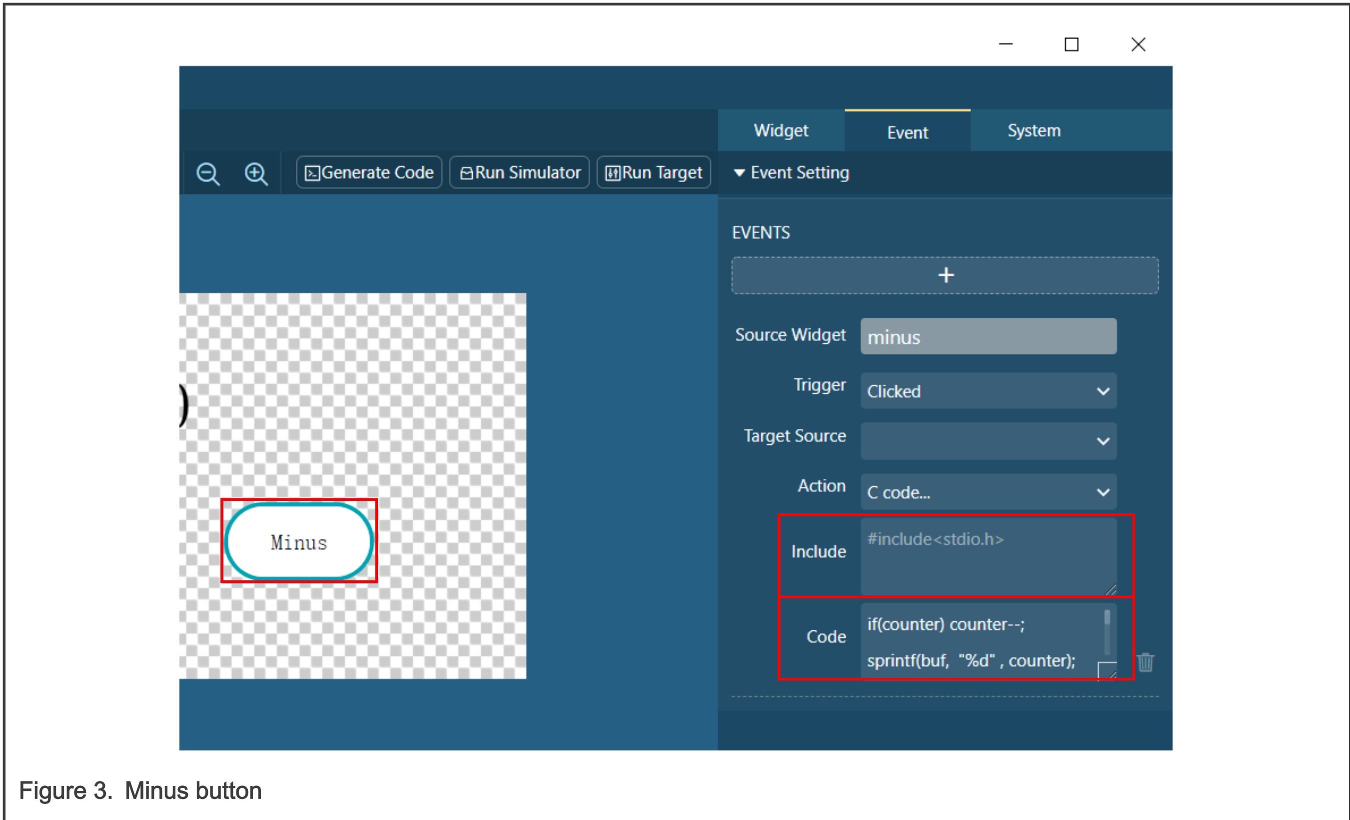


Figure 3. Minus button

In this application there is, by default one action associated with this widget. The action that is performed is the `C code...` action. This is an action that allows custom code to be executed when the widget is triggered. The code to be executed can be directly written into the code dialog box, see Figure 3. Any include files this code may need can be added in the Include dialog box, see Figure 3.

NOTE

More than one C code action per widget / trigger combination is permitted. However, they are going to combine into the same function when the code is generated by GUI Guider.

2.1 File structure

Once the events are configured as desired and code is generated, it is helpful to understand where that code is going to be placed, see Figure 4.

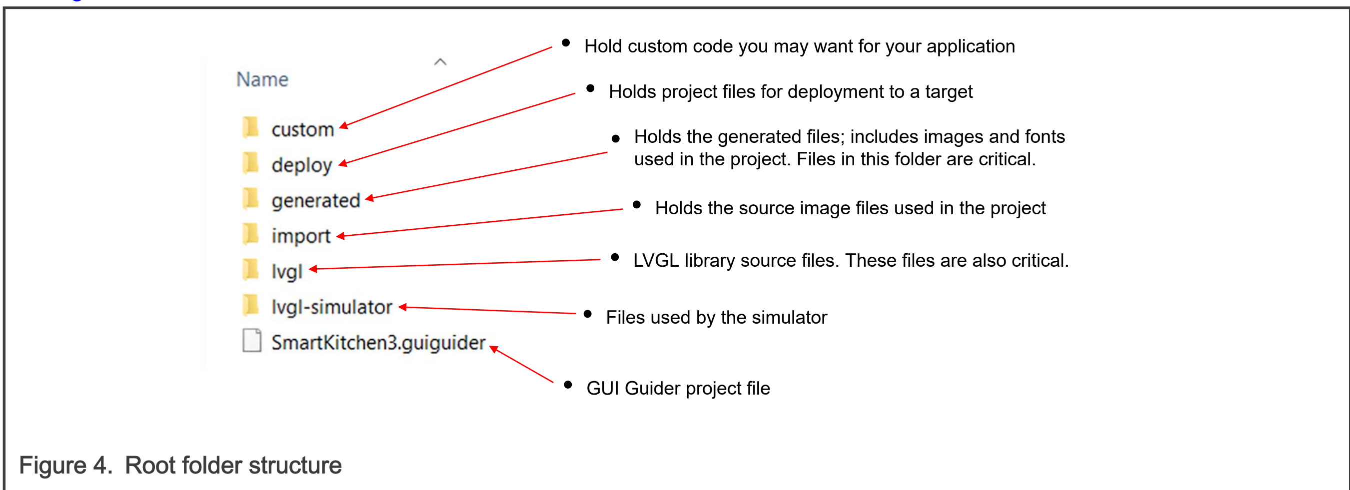


Figure 4. Root folder structure

All event handlers reside in the `events_init.c` file, which resides in the generated folder. Do not manually modify these files, as GUI Guider overwrite any manually written changes when generating code. See Figure 5 for folder structure and description of files within the generated folder.

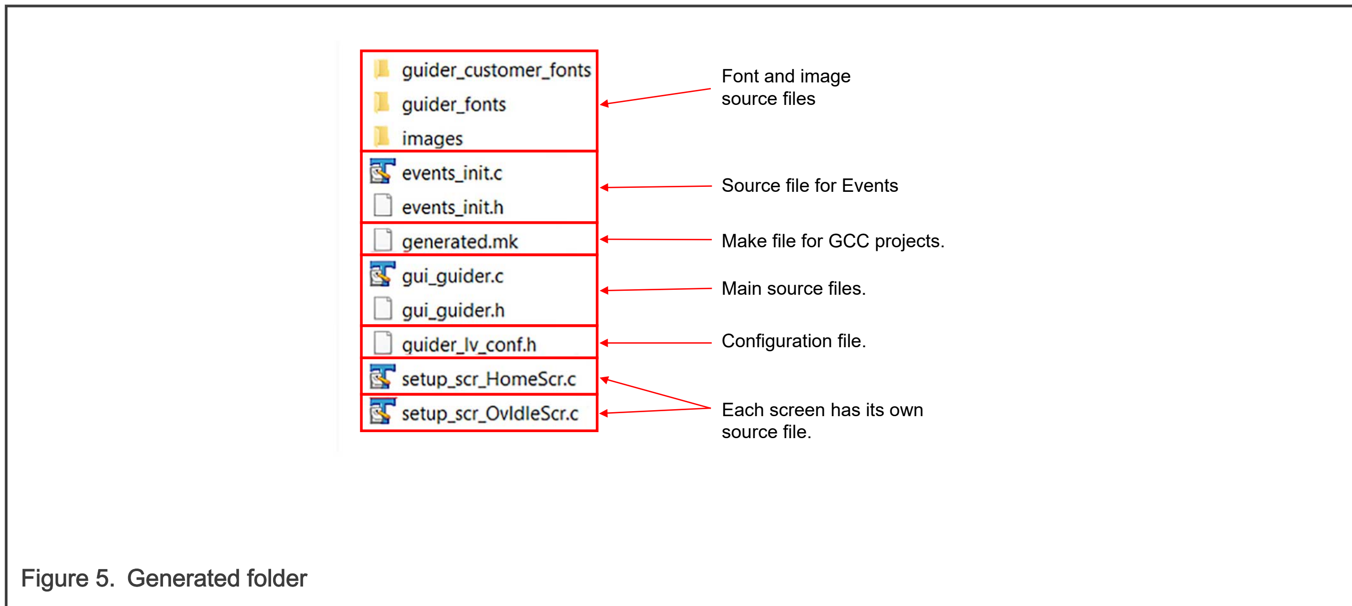


Figure 5. Generated folder

When the project is first created, there is also a custom folder that is created which contains `custom.c/h.` code written in these files, which is not going to be overwritten when generating code via GUI Guider.

3 Adding custom code

3.1 GUI Guider triggering peripherals

If you want a GUI Guider action to trigger a peripheral (for example, a button press on the display is to toggle an LED on the board), custom code in GUI Guider is the way to do this. In this simple example, we use the LED macros that are already defined in `board.h`, see Figure 6.

NOTE

These macros are only defined in board.h, so we must include board.h in the event.

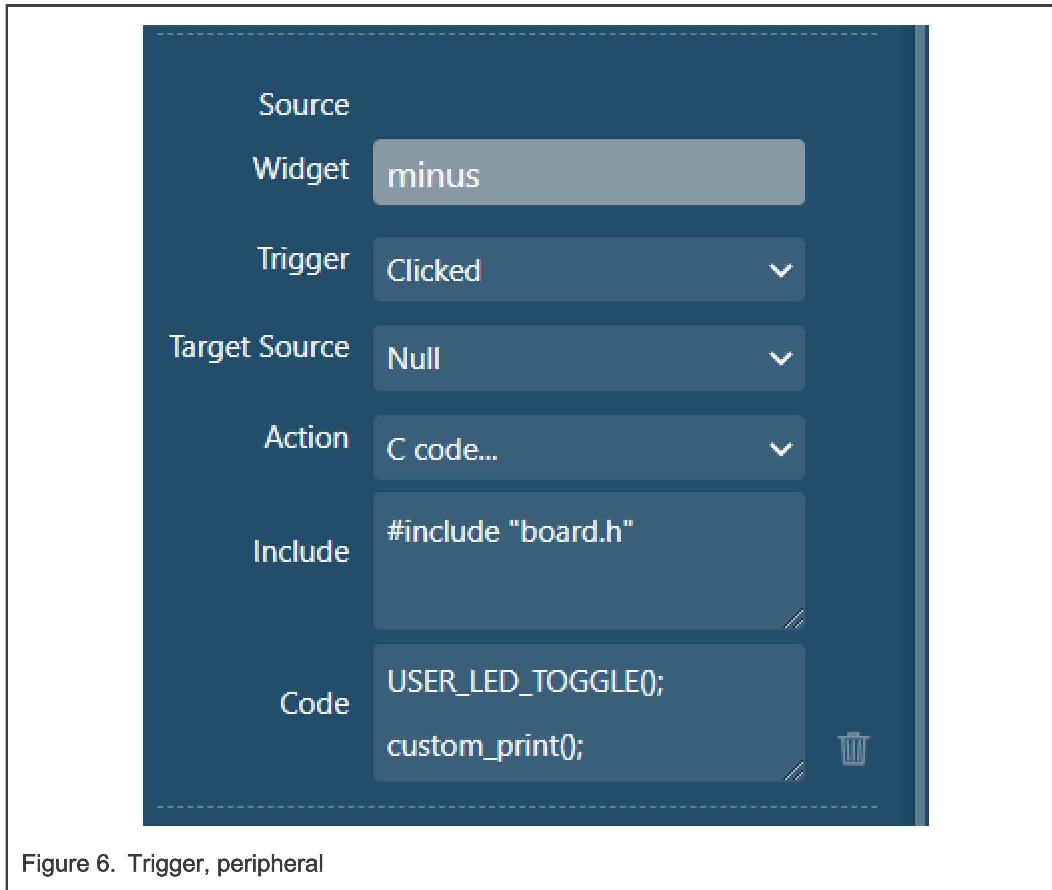


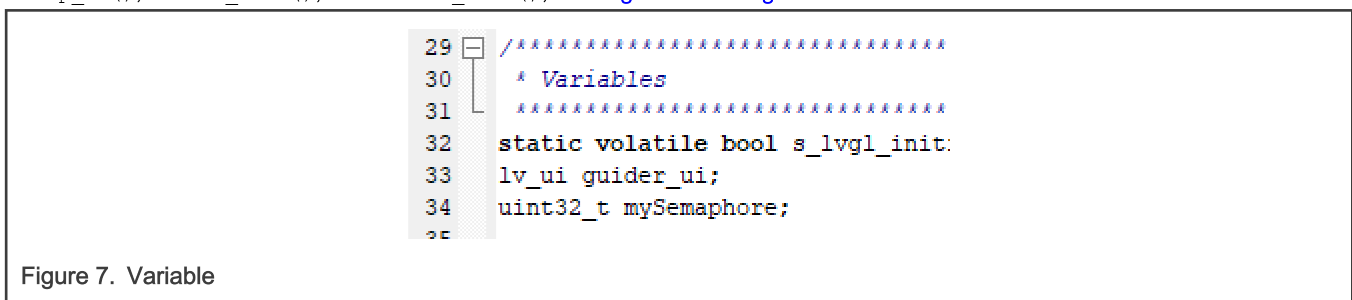
Figure 6. Trigger, peripheral

Now, this is a simple example, not a number of code. Let us take the case where we may need to add many code or something that takes so many lines of code, so it becomes unmanageable in GUI Guider. For this case, the function can be placed in another file, like `custom.c`, and the function called from there.

3.2 Peripherals triggering GUI Guider actions

Take the case where your code would need to interact with GUI Guider without GUI Guider initiating the interaction. This does not require an event to be set up in your GUI Guider application. You simply need to know how to refer and interact with the GUI Guider widgets.

For this purpose, GUI Guider requires a variable of type `lv_ui` to be instantiated and passed to the setup functions as in `setup_ui()`, `events_init()`, and `custom_init()`, see Figure 7 and Figure 8.



```
29 /********************************
30  * Variables
31  ********************************
32 static volatile bool s_lvgl_init;
33 lv_ui guider_ui;
34 uint32_t mySemaphore;
```

Figure 7. Variable

```
79
80      setup_ui(&guider_ui);
81      events_init(&guider_ui);
82      custom_init(&guider_ui);
83
```

Figure 8. Setup functions

This allows user code to interact with the widgets once it has been set up. The `lv_ui` handle contains all of the widgets contained in the application, see Figure 9.

```
typedef struct
{
        lv_obj_t *screen;
        lv_obj_t *screen_img3;
        lv_obj_t *screen_counter;
        lv_obj_t *screen_plus;
        lv_obj_t *screen_plus_label;
        lv_obj_t *screen_minus;
        lv_obj_t *screen_minus_label;
        lv_obj_t *tabPage;
        lv_obj_t *tabPage_tabview0;
        lv_obj_t *tabPage_btn1;
        lv_obj_t *tabPage_btn1_label;
}lv_ui;
```

Figure 9. Widgets

Any LVGL function that can be applied to the object type can be used on the elements of the `lv_ui` structure. The `setup_ui()` function then creates, configure, and assigns the individual elements to their specific widget type. User code must keep track of the widget types of the elements to ensure that elements are only passed to functions meant for that widget type.

4 Software

The example software used for this application note is based on the buttonCounterDemo application template provided by the GUI Guider. This template implements two buttons, a button named plus and other button named minus, which increment or decrement a counter in the center of the screen. The plus button increments the count while, and the minus button decrements the counter. To demonstrate the two situations discussed in this application note, a custom function implements when the minus button on the screen is pressed and an external pushbutton increment the count.

4.1 Code for GUI Guider to trigger peripherals

The method for getting GUI Guider to trigger peripherals starts in the GUI Guider IDE. See Figure 10 for modified event in the GUI Guider.

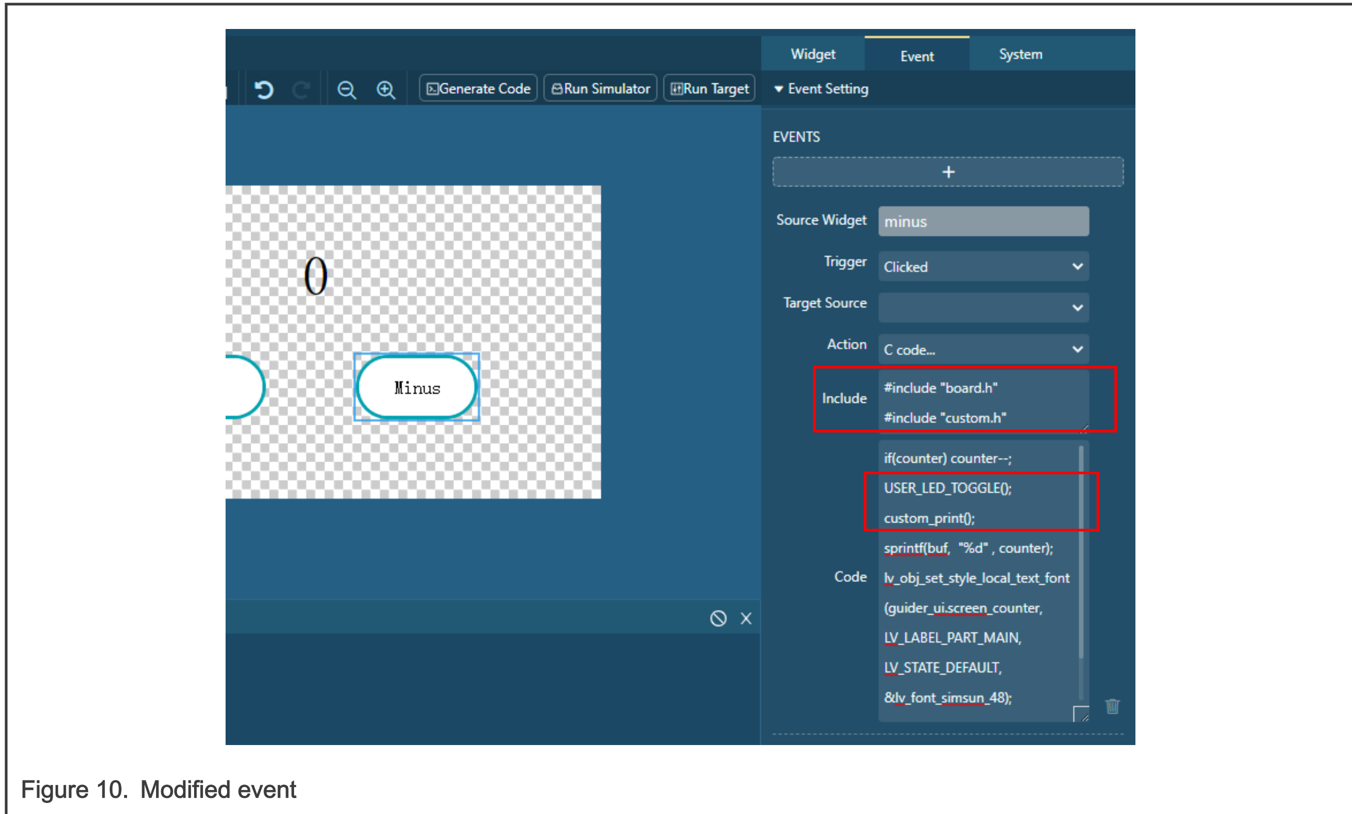


Figure 10. Modified event

With these additions, the code executes the `USER_LED_TOGGLE()` function, which toggles the LED on the board, and a `custom_print()` function, which has to be defined in the code. Also notice that `board.h` has been included. This is necessary because the `USER_LED_TOGGLE()` function is defined in `board.h`. The `custom_print()` function is defined elsewhere, so we should not include for this function.

Now let us examine the generated code. All the code is executed when the minus button is pressed, contained within the `screen_minusevent_handler()`. This function is located in `events_init.c`, see Figure 11.

```
35 static void screen_minusevent_handler(lv_obj_t * obj, lv_event_t event)
36 {
37     switch (event)
38     {
39     case LV_EVENT_CLICKED:
40     {
41         if(counter) counter--;
42         USER_LED_TOGGLE();
43         custom_print();
44         sprintf(buf,  "%d" , counter);
45         lv_obj_set_style_local_text_font(guider_ui.screen_counter, LV_LABEL_PART_MAIN, LV_STATE_DEFAULT, &lv_font_simsun_48);
46         lv_label_set_text(guider_ui.screen_counter, buf);
47     }
48         break;
49     default:
50         break;
51     }
52 }
```

Figure 11. screen_minusevent_handler()

NOTE

The code for the `LV_EVENT_CLICKED` case is exactly how it was written in the GUI Guider IDE event. The include that were added, have also been added to the top of the `events_init.c` file, see Figure 12.

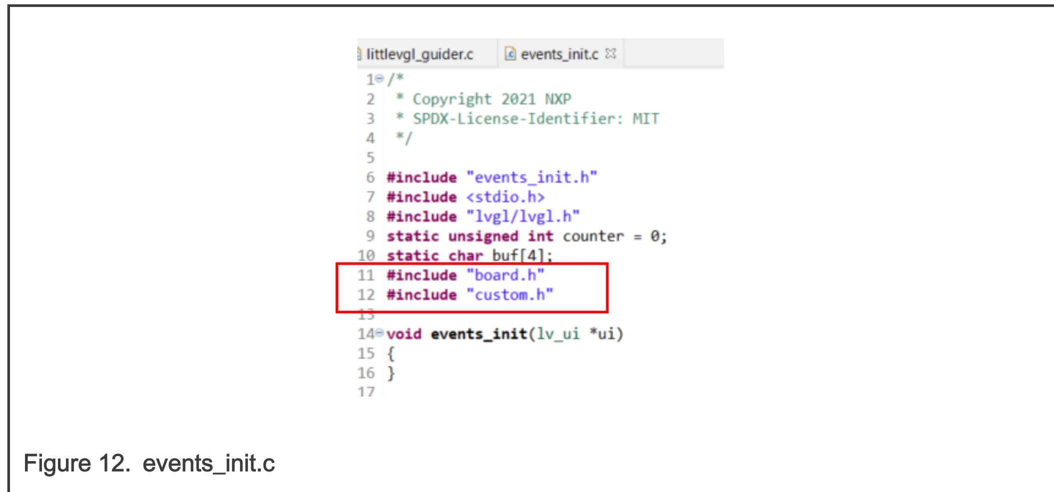


Figure 12. events_init.c

The custom code function is added to the `custom.c` file. No custom code should be added to any of the other GUI Guider files as these files are overwritten every time whenever code is regenerated, see Figure 13 for custom code.

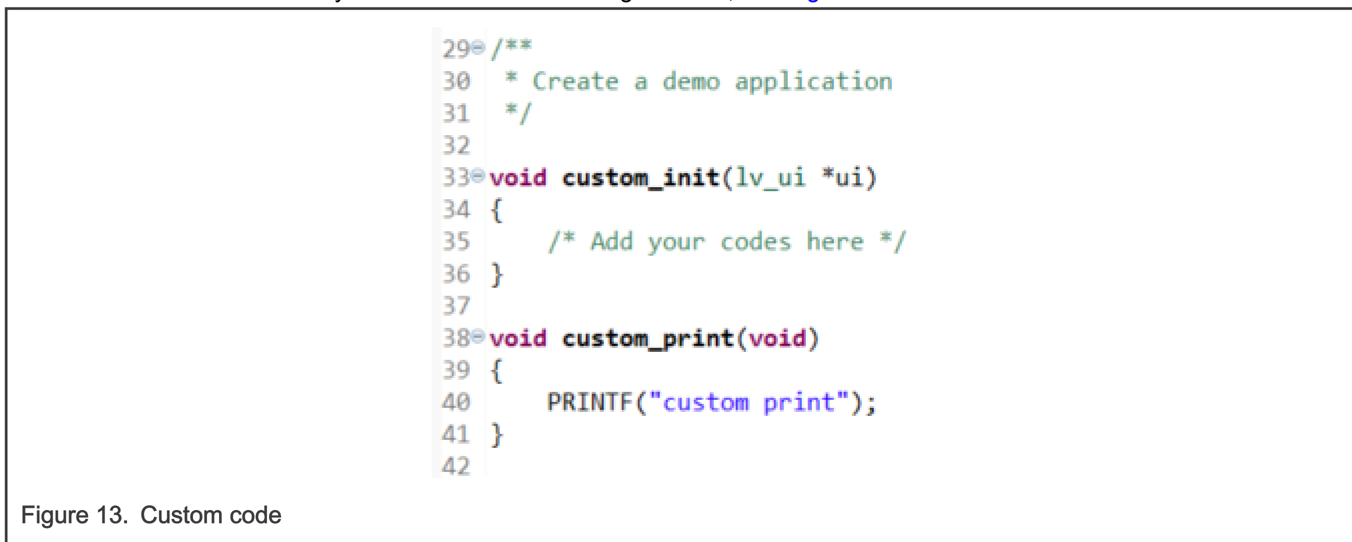


Figure 13. Custom code

4.2 Code for peripherals triggering GUI Guider actions

Enabling peripherals to trigger GUI Guider actions is slightly more involved than the other way around. The main reason for peripherals triggering is because LVGL is not thread safe. So, if an operating system is being used, which is the case in this example, then you cannot interact with the screen at just any point in the application. To cope with this requirement, first an LVGL task has to be created, see Figure 14.

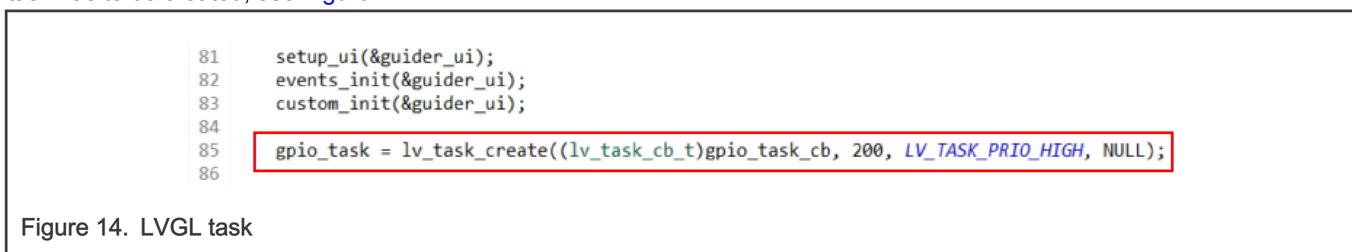


Figure 14. LVGL task

This task checks for a semaphore to be set. If that semaphore is set, the semaphore will be cleared and then the event will be sent to inform LVGL that the plus sign has been clicked, see Figure 15.

```
48 void gpio_task_cb(lv_task_t task)
49 {
50     if(mySemaphore) {
51         mySemaphore = 0;
52         lv_event_send(guider_ui.screen_plus, LV_EVENT_CLICKED, NULL);
53     }
54 }
```

Figure 15. Semaphore

This semaphore will be set in the GPIO interrupt. But before this can happen, the GPIO must be set up for this operation. This is done as one of the first operations in the AppTask, see Figure 16.

```
57 static void AppTask(void *param)
58 {
59     gpio_pin_config_t sw_config = {
60         kGPIO_DigitalInput,
61         0,
62         kGPIO_IntRisingEdge,
63     };
64     lv_task_t * gpio_task;
65
66     lv_port_pre_init();
67     lv_init();
68     lv_port_disp_init();
69     lv_port_indev_init();
70
71     mySemaphore = 0;
72
73     EnableIRQ(BOARD_USER_BUTTON_IRQ);
74     GPIO_PinInit(BOARD_USER_BUTTON_GPIO, BOARD_USER_BUTTON_GPIO_PIN, &sw_config);
75
76     /* Enable GPIO pin interrupt */
77     GPIO_PortEnableInterrupts(BOARD_USER_BUTTON_GPIO, 1U << BOARD_USER_BUTTON_GPIO_PIN);
78
79     s_lvgl_initialized = true;
```

Figure 16. AppTask

Now, the GPIO interrupt handler is written, see Figure 17.

```
39 void BOARD_USER_BUTTON_IRQ_HANDLER(void)
40 {
41     /* clear the interrupt status */
42     GPIO_PortClearInterruptFlags(BOARD_USER_BUTTON_GPIO, 1U << BOARD_USER_BUTTON_GPIO_PIN);
43     mySemaphore = 1;
44     /* Insert LVGL code. */
45     SDK_ISR_EXIT_BARRIER;
46 }
47
```

Figure 17. GPIO interrupt handler

The task of the GPIO interrupt handler is to simply service the GPIO interrupt (clear the flags) and set the semaphore. This way the `gpio_task_cb()` function knows that the GPIO has been pressed and the `LV_EVENT_CLICKED` signal can be sent.

Finally, the pin MUX of the pin must be set correctly. This is configured in the `BOARD_InitPins()` function in the `pin_mux.c` file, see Figure 18. The `IOMUXC_SetPinMux` function should be used to set the appropriate pin and the clock to the `IOMUXC SNVS` logic should also be enabled, see Figure 19.

```
101 void BOARD_InitPins(void) {
102   CLOCK_EnableClock(kCLOCK_Iomuxc);           /* iomuxc clock (iomuxc_clk_enable): 0x03u */
103   CLOCK_EnableClock(kCLOCK_IomuxcSnvs);       /* iomuxc_snvs clock (iomuxc_snvs_clk_enable): 0x03u */
104
105   IOMUXC_SetPinMux(
106       IOMUXC_GPIO_AD_B0_02_GPIO1_IO02,        /* GPIO_AD_B0_02 is configured as GPIO1_IO02 */
107       0U);                                    /* Software Input On Field: Input Path is determined by
```

Figure 18. BOARD_InitPins()

```
177   IOMUXC_SetPinMux(
178           IOMUXC_GPIO_AD_B0_09_GPIO1_IO09,
179           0U);
180   IOMUXC_SetPinMux(
181         IOMUXC_SNVS_WAKEUP_GPIO5_IO00,          /* WAKEUP is configured as GPIO5_IO00 */
182         0U);
183   IOMUXC_SetPinConfig(
184       IOMUXC_GPIO_AD_B0_02_GPIO1_IO02,          /* GPIO_AD_B0_02 PAD functional properties : */
185       0x10B0u);                                 /* Slew Rate Field: Slow Slew Rate
```

Figure 19. IOMUXC_SetPinMux

5 Conclusion

In this application note, it has been shown how to insert custom code into your GUI Guider project to interact with on-chip peripherals, as well as have on-chip peripherals interact with GUI Guider widgets in your project. Both actions have a slightly different implementation and are useful actions for your application. It has also been shown how to add custom code such that your custom code is not destroyed upon subsequent code generations.

6 Revision history

Table 1 summarizes the changes done to this document since the initial release.

Table 1. Revision history

Revision number	Date	Substantive changes
0	18 June 2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

© NXP B.V. 2021. All rights reserved.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 18 June 2021

Document identifier: AN13217