AN14270

Adding Voice Support to GUI Guider for i.MX 93

Rev. 1.0 — 16 May 2024 Application note

Document information

Information	Content
Keywords	AN14270, VIT, speech recognition, inter-process communication (IPC), message queue, GUI Guider
Abstract	This application note explores the possibility of integrating voice by creating a bridge between a speech recognition technology, such as VIT, and the interface creator GUI Guider.

1 Introduction

The user interface has limited the use of the tool GUI Guider. Getting an interaction only through a mouse or touchscreen can be enough for some use cases. However, sometimes the use case requires to go beyond its limitations. This document explores the possibility of integrating voice by creating a bridge between a speech recognition technology, such as VIT, and the interface creator GUI Guider. It uses a universal way to link all the voice recognition commands and a wakeword to any interaction created by GUI Guider.

2 Overview

To set the communication between GUI Guider and VIT technology commands, refer to Section 8. The communication is build using a code created as a handler, which listens and enables it to simulate events in the GUI to create the interaction.

2.1 GUI Guider

GUI Guider is a user interface development tool from NXP that provides a fast option to create a high-quality display using the LVGL graphics library. It uses a different variety of widgets, animations, and styles, with different trigger configurations and customization with the possibility of not coding. For more information on GUI Guider, refer to *GUI Guider v1.6.1 User Guide* (document GUIGUIDERUG).

2.2 Voice intelligent technology

Voice Intelligent Technology (VIT) is a tool created by NXP to define wakewords and commands using free online tools, library, and voice control software package. MCUXpresso can use it for micro-controllers or Linux BSP can use it for micro-processors.

2.3 Message queue

Message queue (MQUEUE) is a manager that implements the format POSIX 1003.1b message queues. It is used as inter-process communication (IPC) to create the bridge between GUI Guider and VIT. It exchanges data in the form of messages, sending it through VIT and performing the management with the script `command_handler`.

3 Hardware, software, and host requirements

Table 1 provides details of the hardware, software, and host required to use VIT and GUI Guider.

Table 1. Hardware, software, and host used

Category	Description
Hardware	• i.MX 93 EVK • Power supply: USB Type-C 45 W power-delivery supply (5 V/3 A) • USB Type-C male to USB Type-A male cable: assembly, USB 3.0 compliant • LVDSL adapter and HDMI cable or DY1212W-4856 LVCD LCD panel • Internal i.MX 93 microphone
Software	• Linux BSP version: L6.1.55_2.2.0 • GUI Guider v1.6.1 version onward • Toolchain 6.1-Langdale
Host	• X86_64 Linux Ubuntu 20.04.6 LTS

4 Pre-requirements

This section describes the installation of different tools required.

4.1 Flashing Linux version

Before following the below steps, change the boot configuration to the download mode and connect a USB through the host. For more information, refer to *i.MX Linux User's Guide* (document IMXLUG).

To flash the EVK, perform the following steps:

1. Download the recent NXP Linux BSP image release for i.MX 93 (L6.1.55_2.2.0 or latest).
2. To flash the EVK, download the recent UUU: https://github.com/nxp-imx/mfgtools/releases.
3. Connect the EVK with the host using EVK port USB1.
4. Using the `imx-image-full`, place both programs in the same file and flash the EVK using the following command:

```
$ ./uuu.exe -b emmc_all <Kernel>.sd-flash_evk imx-image-full-imx93evk.wic
```

Alternatively, use only the image to flash the EVK:

```
$ ./uuu.exe -b emmc_all imx-image-full-imx93evk.wic
```

Note: *Ensure to check the boot pins.*

4.2 Toolchain with Yocto project

Yocto project is an open source collaboration that helps to create custom Linux-based systems. Yocto creates the image used by i.MX.

Ensure that the host machine has an application development toolkit (ADT) or toolchain to have the same environment as the EVK. Ensure it is able to compile applications for the target board. To get the correct toolchain, refer to "section 4.5.12" in *i.MX Linux Users Guide* (document IMXLUG) and "section 4" in *i.MX Yocto Project Users Guide* (document IMXLXYOCTOUG).

To get the toolchain on the host machine from the Yocto environment, perform the following steps:

1. Create a `bin` folder in the home directory:

```
$ mkdir ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

2. Ensure that the `~/bin` folder is in the `PATH` variable.

```
$ export PATH=~/bin:$PATH
```

3. Clone the recipes to use in the repository:

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://github.com/nxp-imx/imx-manifest -b imx-linux-mickledore -m
 imx-6.1.55-2.2.0.xml
$ repo sync
```

4. To build, configure as follows:

```
$ DISTRO=fsl-imx-fb MACHINE=imx93evk source imx-setup-release.sh -b deploy
```

5. To generate the toolchain, set up a standalone environment without the Yocto Project as follows:

```
$ DISTRO=fsl-imx-fb MACHINE=imx93evk bitbake core-image-minimal -c populate_sdk
```

5 GUI Guider

This section explains about GUI Guider and how to use the basics to create a project based on this tool. It also explains about the different characteristics to use and take advantage of those characteristics.

5.1 Gui Guider widgets and events

When the user creates a project in GUI Guider, the use of different widgets is assigned as an object generated automatically. This object has different properties; one of them is the **Events**. Depending on the widget, the events can have different triggers, and what happens depend on the target. For example, Figure 1 shows what happens if a button targets the screen to have only the action "Load screen".

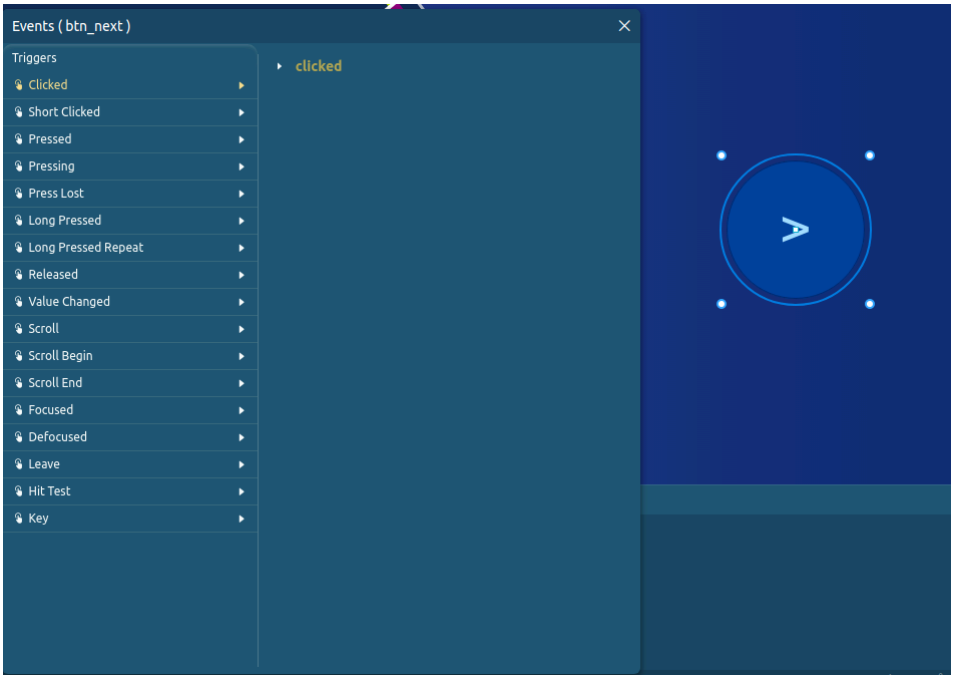

Figure 1. Events assigned to btn_next

These objects can be found in the path `<Project path>/generated/gui-guider.h`. The script `command_handler` takes advantage of the events used by the widgets simulating the trigger.

For more information on widgets and events, refer *GUI Guider v1.6.1 User Guide* (document GUIGUIDERUG).

5.2 Quick start

To start working, install the GUI Guider.

On host installation, perform the following steps:

1. Download the most recent version of GUI Guider (1.7.1 or latest).
2. Follow the steps to download.

Here, the user can choose to create a project with official examples or the local projects.

To create a GUI project, perform the following steps:

1. Open GUI Guider 1.7.1.
2. Create a project.

3. Select the LVGL version.

Figure 2. Select the LVGL version

4. For i.MX 93, select the i.MX processor.

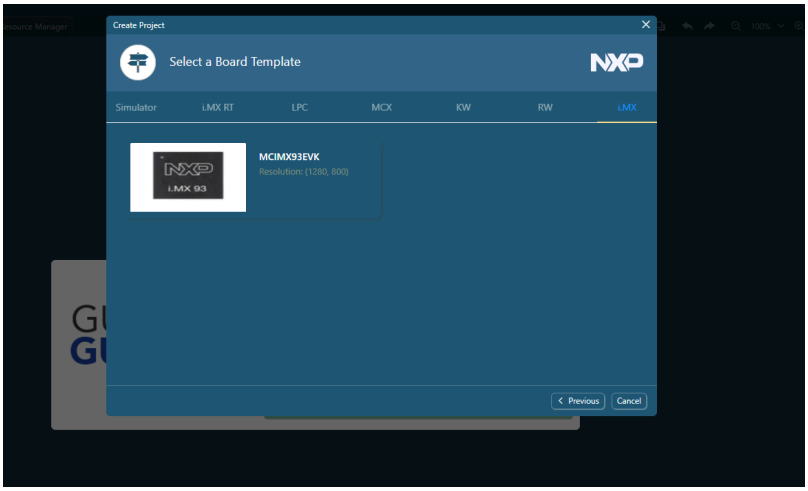

Figure 3. i.MX boards

5. Select a template. For this document, choose the "ScreenTransition" template.

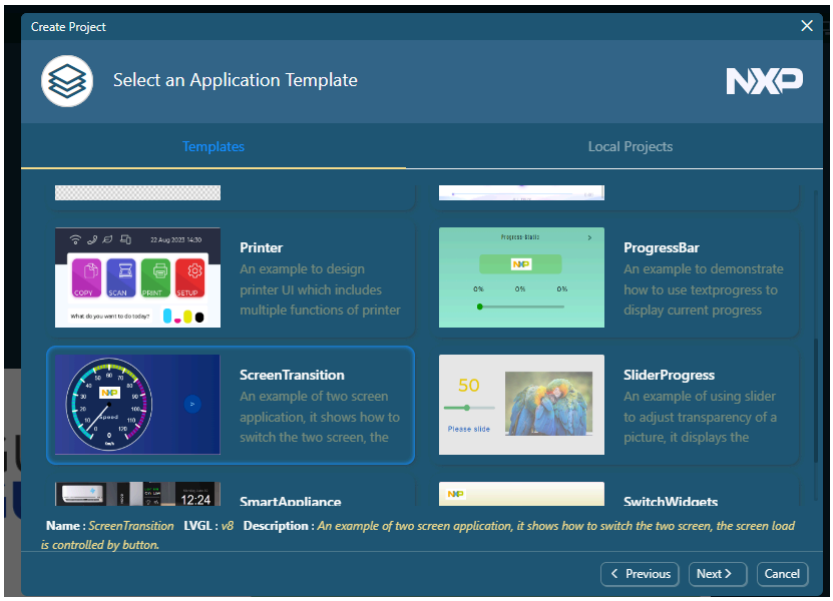

Figure 4. i.MX 93 templates

6. Choose a **Project Name** and to create a project, click **Create**.

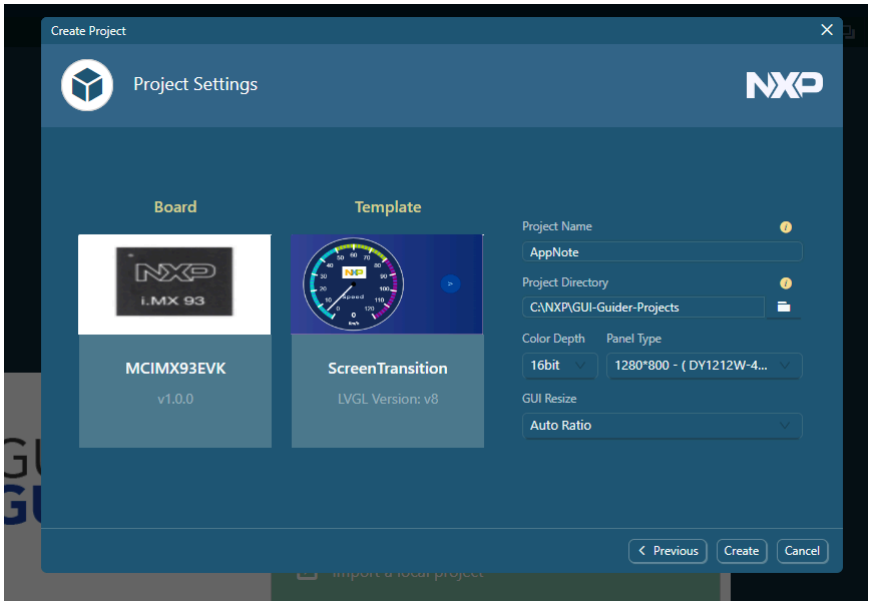

Figure 5. Project settings

7. The main window must appear, as shown in Figure 6.

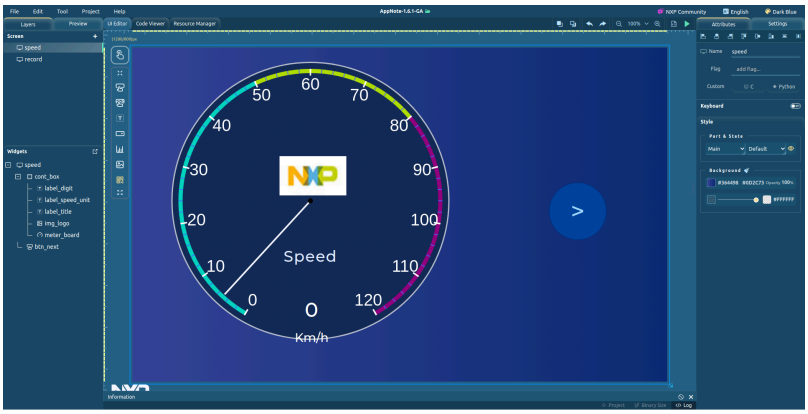

Figure 6. GUI Guider GUI

5.3 Creating widgets, events, and triggers

To create widgets, events, and triggers, perform the following steps:

1. On the left-side of the GUI Guider, click the button, highlighted in red, two times.

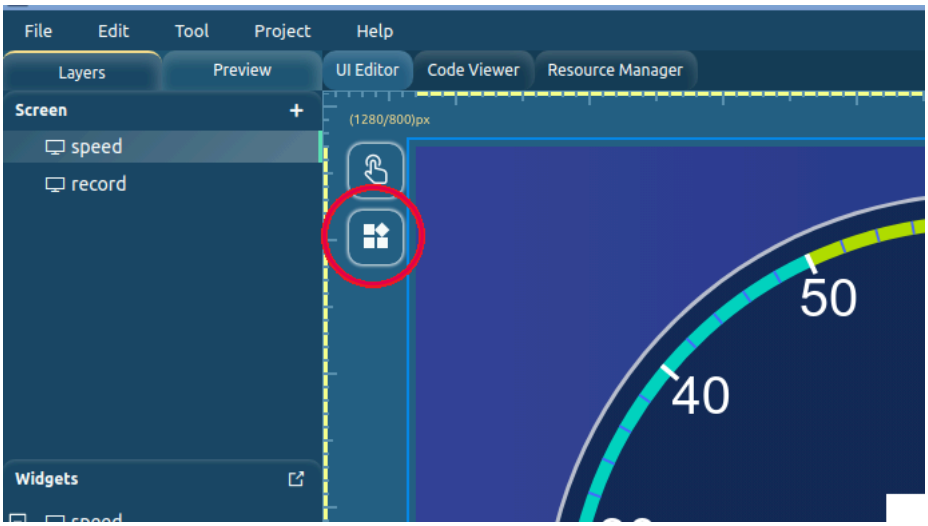

Figure 7. Widgets

2. As a result, the button expands to show all the available widgets.

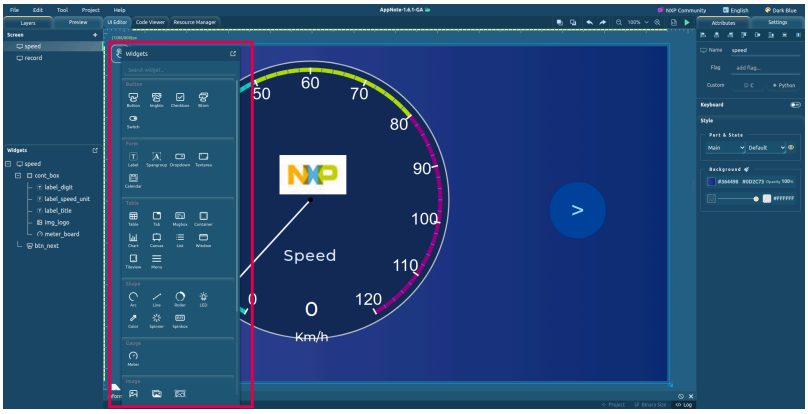

Figure 8. Widget menu

There can be various widgets with different properties. This application note focuses on the widget type button. However, there can be other types of widgets with their limitations. For more information, refer to "Widget details" in *GUI Guider v1.6.1 User Guide* (document GUIGUIDERUG).

3. Add the **Button** widget by dragging it to the UI from the widgets tab.

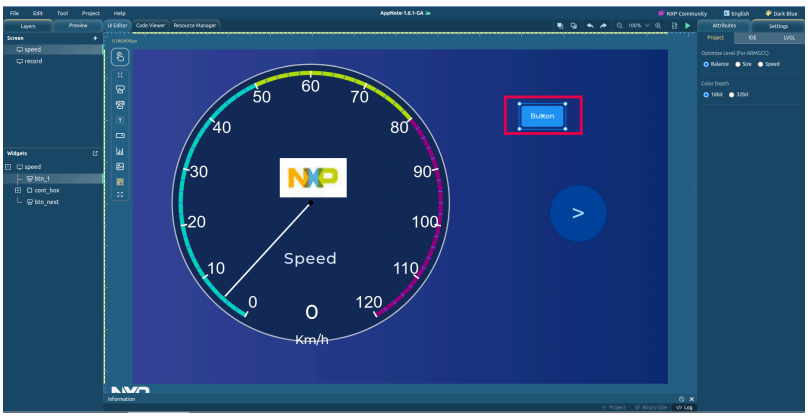

Figure 9. Widget created

4. Right-click on the **Button** for the properties and click **Add event**.

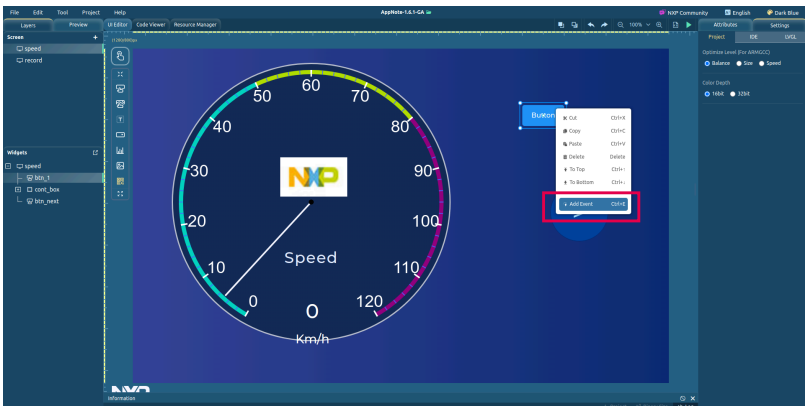

Figure 10. Adding events

5. A window pops up showing all the events the widget can trigger.

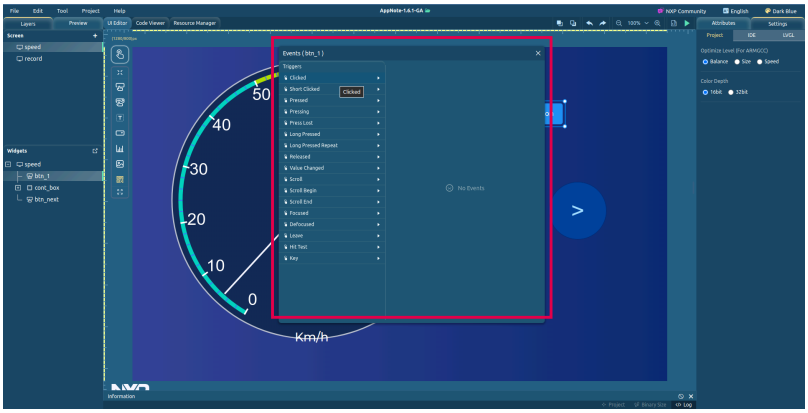

Figure 11. Event triggers

6. Next, the window show all the events the trigger can fire. These events can be applied to screens, other widgets, or creating custom events.

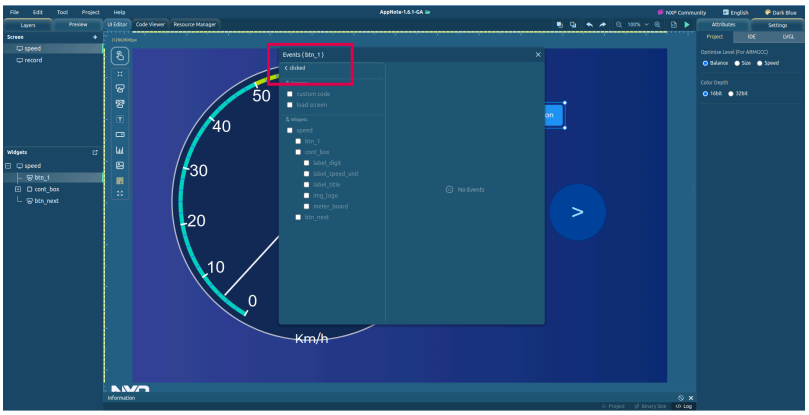

Figure 12. Clicked trigger

7. For this example, a new screen is loaded. Click the **load screen** and select the screens to be loaded.

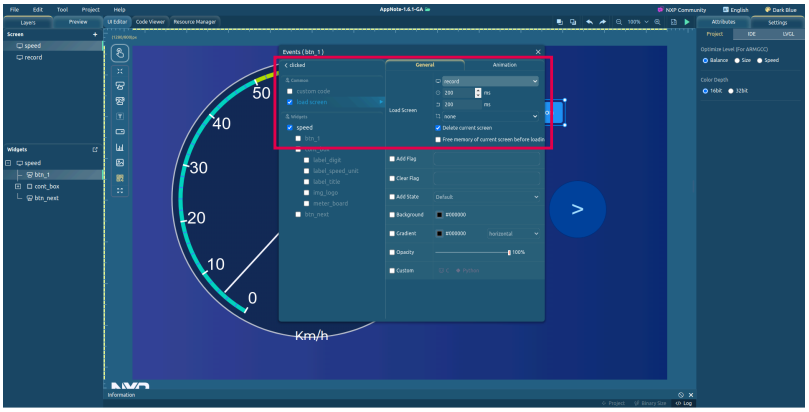

Figure 13. Adding events action

8. To test the application, use the simulator integrated with GUI Guider. It is used to select the next button and the type of simulation to use. For this case, use a simulator in **C**.

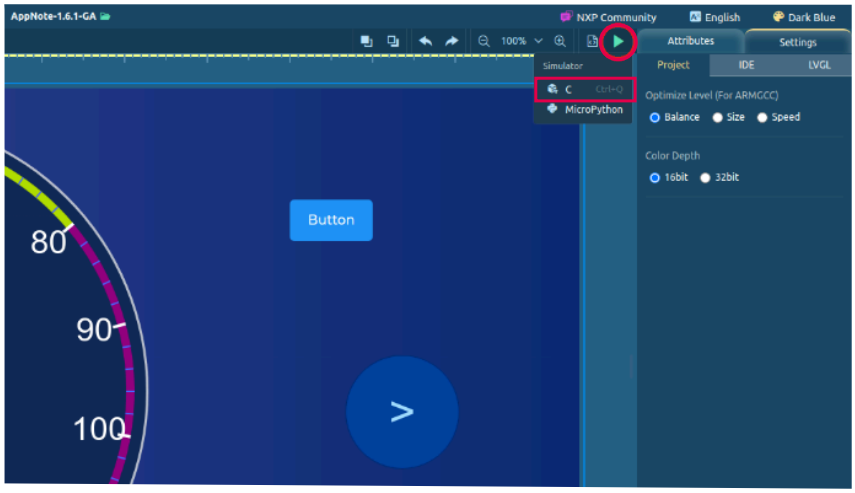

Figure 14. Simulator tab

9. To load the new screen, click **Button**.

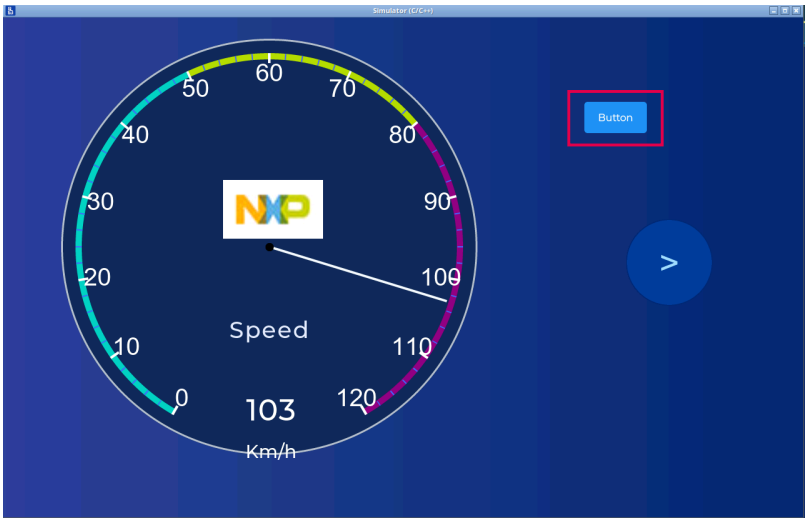

Figure 15. Simulator window

5.4 Building for i.MX 93

To build i.MX 93, perform the following steps:

1. Ensure that the toolchain used by GUI Guider has been installed correctly. To cross-verify, check the path `~/<GUI-Guider project path>/ports/linux/build.sh`:

```
  1 #!/bin/sh
  2
  3 toolchain=$1
  4 if [ -z "$toolchain" ];then
  5     toolchain=/opt/fsl-imx-xwayland/6.1-langdale/sysroots/x86_64-pokysdk-linux/usr/share/
cmake/armv8a-poky-linux-toolchain.cmake
  6 fi
```

2. From the previous example, to create the application and run it on i.MX 93, select **Project > Build > Yocto** from the top bar.

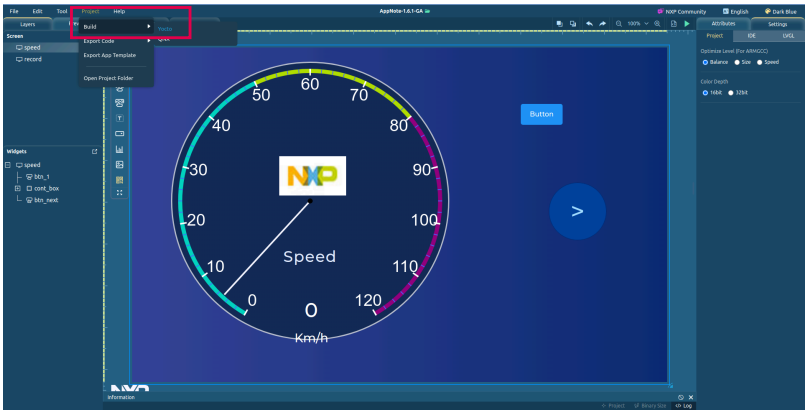

Figure 16. Project building

3. To check the status of Project, Binary size, and Log, select the **Information** tab at the bottom of the application. Check the log by expanding the **Information** tab.

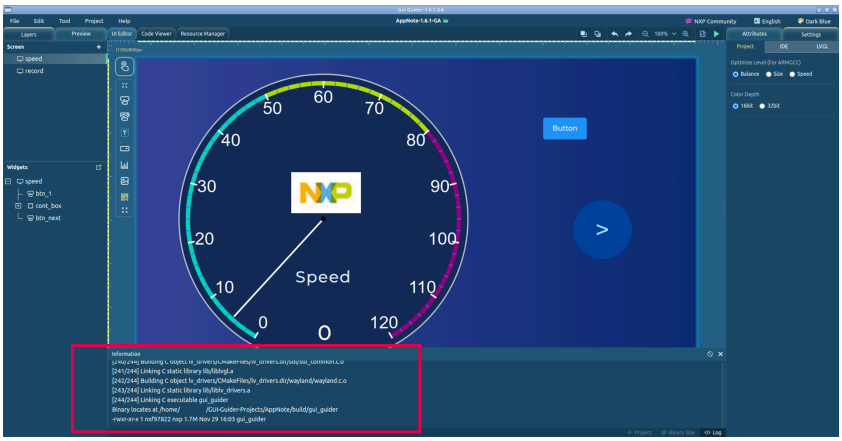

Figure 17. Information tab

4. The log provides building information including the location of the binary file. For this case, the binary is in the path `/<GUI-Guider project path>/build/gui_guider`.

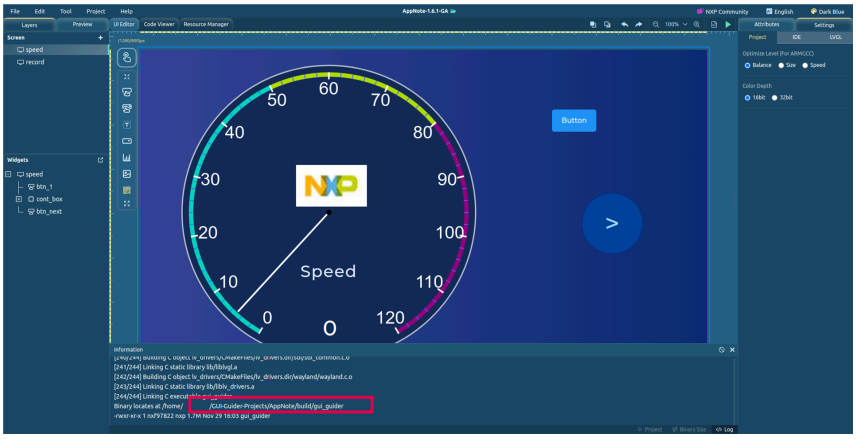

Figure 18. Log information

5. Locate the host terminal and send it to the EVK using the following command:

```
$ scp <binary location> root@<evk ip>:/home/root
```

Note: *To use the above approach, it is necessary that both the machines, host, and target are on the same network and the board IP is known.*

6. Execute the binary file on the EVK using the following command:

```
root@imx93evk:~# ./gui_guide
```

For example, using an LVDS screen, which shows the project built by GUI Guider, as shown in Figure 19.

Figure 19. Binary example using LVDS

6 VIT

This section explains how to use VIT standalone and generate the model to link it with the GUI Guider. It explains how to use the host to generate a model with the desired characteristics. For more information, refer to VOICE-INTELLIGENT-TECHNOLOGY.

6.1 Create the model

To create the model, perform the following steps:

1. Log in to the VIT website: VIT Model Generation Tool
2. Click the **GENERATE MODEL** tab.

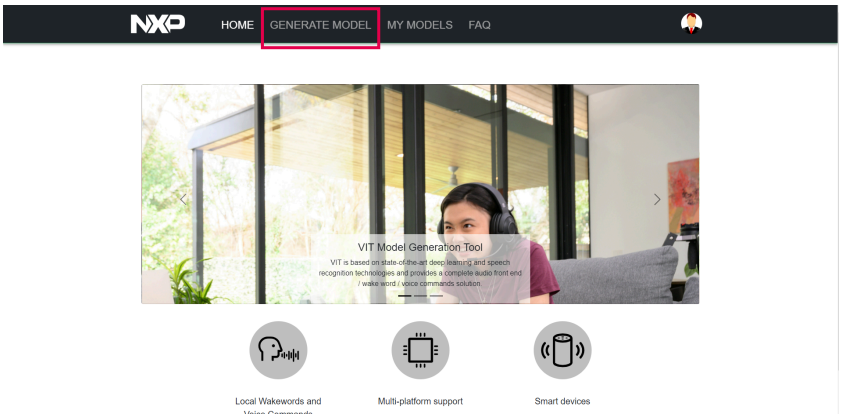

Figure 20. Log in to the VIT website

3. Select **SW platform & version** as "Linux BSP" and "LF6.1.55_2.2.0". Also, select the applicable options for **Device** as "i.MX93" and **Language** as "English".

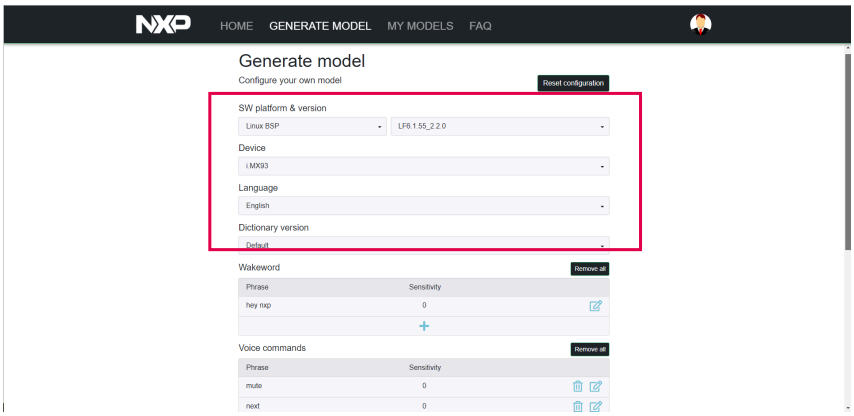

Figure 21. VIT model configuration

4. Add wakewords, which act as a trigger that tells VIT when to start listening for a voice command. When a new wakeword or command is created, it asks to set the value for "Sensitivity". This parameter increases the recognition rate, which means if it is a positive value it is easier to detect but can result in more false detections. Instead of the negative value used to avoid confusion between keywords, maintain the sensitivity value as 0. For example, here, the phrase "hey led" is added.

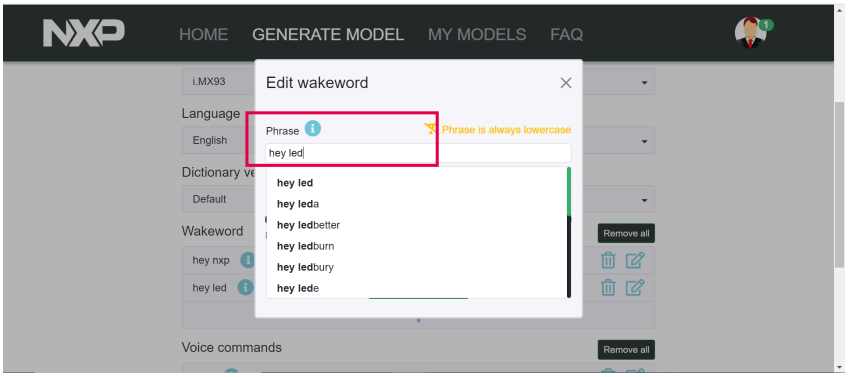

Figure 22. Adding wakewords

5. Add the voice commands to be used and eliminate the ones not used.

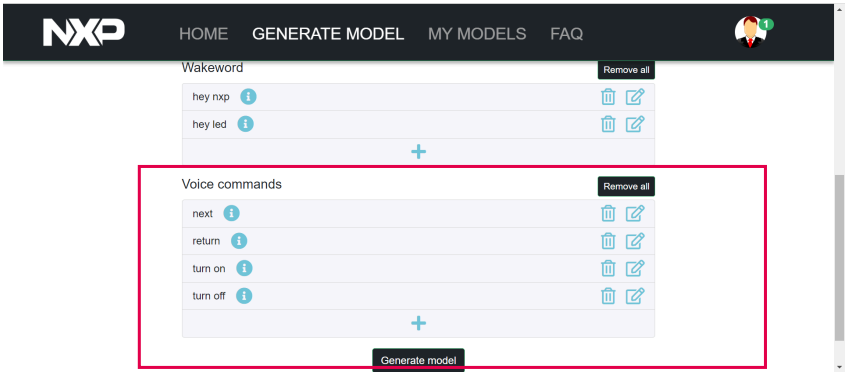

Figure 23. Adding and eliminating voice commands

6. Click the **Generate model** button and wait until the **Download model** button is unlocked.

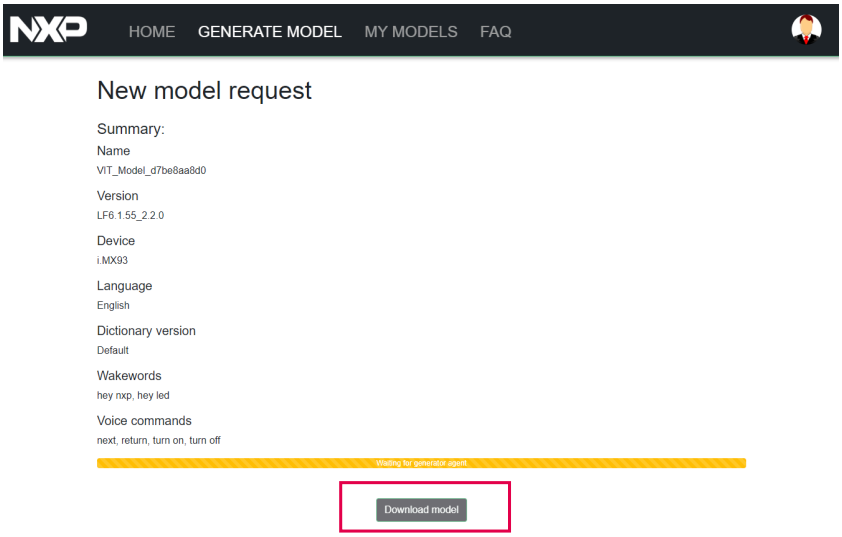

Figure 24. VIT model generate request

7. The model is sent to the **MY MODELS** tab. To download the most recent model, click the download icon.

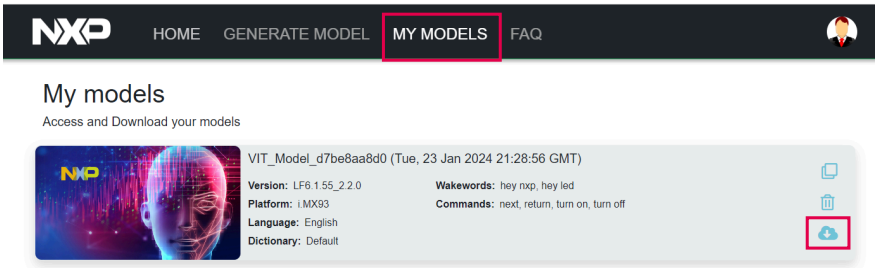

Figure 25. Downloading the recent model

8. Extract the zip folder and save the file `VIT_Model_en` containing the `VIT_package` folder.

6.2 Compiling VIT voice_ui_app as standalone

Voice_ui_app is an example created for the repository `imx-voiceui`. This application uses the model to detect wakewords and commands. A utility used by this document is the "`notify`" argument. This argument when it detects a wakeword or command, opens a Python file `WakeWordNotify` or `WWCommandNotify` with a system argument using the identifier (ID). This ID helps to differentiate between the triggers.

To create the voice_ui_app on the host and help to assign it to the previous model created, perform the following steps:

1. Clone VIT repository including the branch version, using the following command:

```
$ git clone https://github.com/nxp-imx/imx-voiceui -b lf-6.1.55-2.2.0
```

2. Create a backup of the original file, using the following command:

```
$ cd <build-dir>/imx-voiceui
  $ mv <Custom VIT_Model_en.h> ./vit/platforms/iMX9_CortexA55/lib/VIT_Model_en.h
```

3. Set up the toolchain previously installed:

```
$ source /opt/fsl-imx-xwayland/6.1-langdale/environment-setup-armv8a-poky-linux
```

Note: *Use the toolchain created by Yocto.*

4. Build your project, using the following command:

```
$ make all VERSION=04_08_01 CURRENT_GCC_VERSION=10 BUILD_ARCH=CortexA55
```

5. Once the project is built, it generates a directory named `release`. Copy the file `voice_ui_app` in this directory to the EVK:

```
$ scp release/voice_ui_app root@<evk ip>:/home/root
```

6.3 Using the parameter -notify

The script called by voice_ui_app when passing the "`-notify`" flag, must be in the path `/usr/bin/`. Use the attached files to `/usr/bin/` and copy these scripts to the EVK.

```
$ scp WakeWordNotify root@<evk ip>:/usr/bin/
$ scp WWCommandNotify root@<evk ip>:/usr/bin/
```

The files inside, use the wakeword/command ID and send it through the message queue.

After copying these files to the EVK, use the parameter "`-notify`" to imply that the files `WakeWordNotify`, and `WWCommandNotify`, have the necessary permissions. To add it on the EVK, execute the following command:

```
root@imx93evk:~# chmod a+x /usr/bin/WakeWordNotify
root@imx93evk:~# chmod a+x /usr/bin/WWCommandNotify
```

6.4 Audio front-end

The audio front-end (AFE) is used as a feed for VIT voice recognition. It helps to clean noise and echo by using the source and a reference of the speaker. Therefore, the result is a clear single channel microphone audio that can be used for processing. For more information, see VOICESEEKER.

AFE can be found inside the EVK at the path `/unit_tests/nxp-afe`.

To prepare and execute the program, follow the steps in file `TODO.md` in nxp-afe:

1. Ensure that the DTB is `imx93-11x11-evk.dtb`.
2. Install `aloop` module to support AFE:

```
root@imx93evk:~# sudo modprobe snd-aloop
```

3. Create a backup of `asound.conf` and use the corresponding `asound.conf` for the board:

```
root@imx93evk:~# mv /etc/asound.conf /etc/asound-o.conf
```

```
root@imx93evk:~# cp /unit_tests/nxp-afe/asound.conf_imx93 /etc/asound.conf
```

4. Change the `WakeWordEnginge` to use the VIT word engine correctly. This configuration is inside the file `/unit_tests/nxp-afe/Config.ini`.
5. Modify the property `WakeWordEngine = VoiceSpot` that uses VoiceSpot as a default to `WakeWordEngine = VIT`.
6. To test the AFE, execute `voice_ui_app`:

```
root@imx93evk:~# ./voice_ui_app &
```

Note: *For this case, it is not necessary to add the parameter "`-notify`".*

7. Execute the AFE, using the following command:

```
root@imx93evk:~# /unit_tests/nxp-afe/afe libvoiceseekerlight &
```

8. To determine if AFE runs in the background, use the `&` command. To know what other programs are running in the background, use the following command:

```
root@imx93evk:~# ps
```

9. To close the AFE or `voice_ui_app`, use the following command:

```
root@imx93evk:~# pkill afe
root@imx93evk:~# pkill voice_ui_app
```

6.5 Running voice_ui_app without -notify

1. After following the steps in the `TODO.md` file, run the binary `voice_ui_app` from the terminal on the EVK. It displays information about how the VIT is running.

```
root@imx93evk:~# ./voice_ui_app &
rdspVoiceSpot_CreateControl: voicespot_status = 0
rdspVoiceSpot_CreateInstance: voicespot_status = 0
VoiceSpot model: HeyNXP_en-US_1.bin
rdspVoiceSpot_OpenInstance: voicespot_status = 0
rdspVoiceSpot_EnableAdaptiveThreshold: voicespot_status = 0
rdspVoiceSpot_SetParametersFromBlob: voicespot_status = 0
VoiceSpot library version: 0.24.1.1696512275
VoiceSpot model version: 0.13.1

VIT Model info
  VIT Model Release = 0x40900
  Language supported : English
  Number of WakeWords supported : 2
  Number of Commands supported : 4
  WakeWord supported :
   'HEY NXP'
   'HEY LED'
  Voice commands supported :
   'NEXT'
   'RETURN'
   'TURN ON'
   'TURN OFF'
Using VIT for wakeword detection.
```

2. To feed the `voice_ui_app`, execute the AFE using the following command:

```
root@imx93evk:~# /unit_tests/nxp-afe/afe libvoiceseekerlight &
```

3. Say the wakeword and voice command and check if it is working as expected. It shows the wakeword and the voice command in the terminal as follows:

```
- Wakeword detected 1 HEY NXP StartOffset 16640
 - Voice Command detected 3 TURN ON
```

7 GUI Guider VIT application

As explained earlier, the application/script `command_handler` through the VIT notification sends the command ID and wakeword ID to a message queue as IPC. It then captures these IDs to simulate an event in a GUI-Guider application. Figure 26 shows how this communication has been executed.

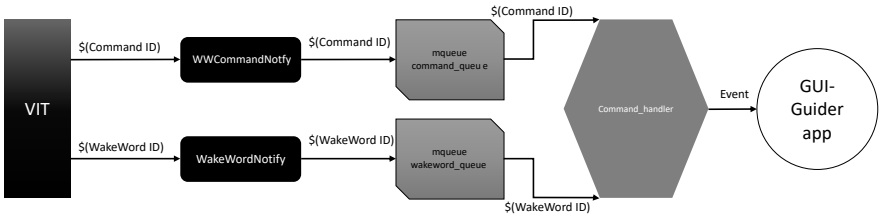

Figure 26. VIT to GUI Guider communication

Note: *Ensure to configure the handler to work correctly with the custom model created. These modifications must be applied on the host.*

7.1 Use command_handler to simulate events

To use the `command_handler` to simulate events, perform the following steps:

1. Add the files `command_handler.h` and `command_handler.c` to the GUI Guider project in the directory `/<GUI-Guider project path>/custom/`.
2. To match the current model used, modify the `command_handler.h` by changing the `voice_cmd_t` and `voice_ww_t`.
   Note: *Ensure that the same order is used in the model.*

```
typedef enum{
      HEY_NXP = 1,
      HEY_LED = 2,
}voice_ww_t;

typedef enum{
      UNKNOWN,
      NEXT,
      RETURN,
      TURN_ON,
      TURN_OFF,
}voice_cmd_t;
```

3. Modify the quantity of wakewords and commands in the file `/<GUI-Guider project path>/custom/command_handler.h`:

```
#define VIT_WW_NUMBER       2
#define VIT_CMD_NUMBER      5
```

4. Initialize the command interface in the file /<GUI-Guider project path>/custom/custom.c. GUI Guider generates this file automatically.

```
#include "command_handler.h"
```

5. Function defined as void custom_init(lv_ui *ui) is available in the file /<GUI-Guider project path>/custom/custom.c. This function can be modified to add a code and the initializer command start_command_handler() as follows:

```
void custom_init(lv_ui *ui)
  {
      /* Add your codes here */
      start_command_handler();
  }
```

Where:
The start_command_handler() is used for creating a thread running as a handler, taking messages sent by VIT, and executing commands assigned by command_handler_link().

6. To link the VIT wakewords and command with the object and event, use the following command:

```
void command_handler_link(voice_ww_t WW_Id, voice_cmd_t CMD, lv_obj_t** obj, lv_event_code_t
 event);
```

Where:

- The command_handler_link() is used to save an event to simulate for VIT execution.
- The inputs, voice_ww_t and voice_cmd_t, are created in step 2 relate directly with the VIT model.
- The third argument, lv_obj_t**, relates to GUI Guider object creation. First, locate the object to be linked. The name conforms with the next structure <screen located>_<name of the object>. To find where it is defined, check the file generated by GUI Guider at generated/gui_guider.h. Here, you can find the next structure with all the possible objects to link.

```
typedef struct
  {

      lv_obj_t *speed;
      bool speed_del;
      lv_obj_t *speed_btn_next;
      lv_obj_t *speed_btn_next_label;
      lv_obj_t *speed_cont_box;
      lv_obj_t *speed_meter_board;
      lv_meter_indicator_t *speed_meter_board_scale_1_ndline_0;
      lv_obj_t *speed_img_logo;
      lv_obj_t *speed_label_title;
      lv_obj_t *speed_label_speed_unit;
      lv_obj_t *speed_label_digit;
      lv_obj_t *speed_btn_1;
      lv_obj_t *speed_btn_1_label;
      lv_obj_t *speed_led_1;
      lv_obj_t *record;
      bool record_del;
      lv_obj_t *record_chart_board;
      lv_obj_t *record_label_title;
      lv_obj_t *record_btn_before;
      lv_obj_t *record_btn_before_label;
      lv_obj_t *record_img_logo;
  }lv_ui;
```

The function custom_init(lv_ui *ui) is used to initialize at the start of GUI Guider execution. This structure can be used to relate it with an object, knowing how to use it correctly. The pointer of the given structure is *ui, and the pointer to search is lv_obj_t**. Therefore, it is necessary to use this structure with the next format:

```
&ui->speed_btn_1
```

- The fourth argument, `lv_event_code_t event`, relates to the event that is going to be triggered. It usually has a structure like this: `LV_EVENT_<EVENT ASSIGNED>`. It determines what to do with the triggered event through the code viewer in the file `events_init.c`.
  For example, the btn_1 created in the screen **speed** have these events generated by GUI Guider.

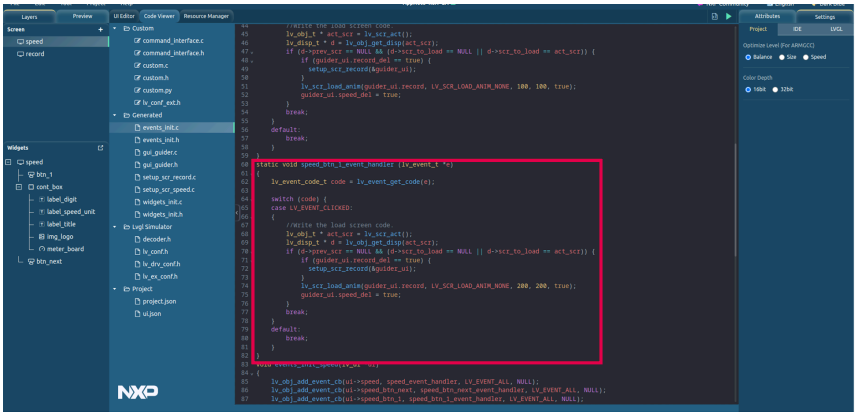

Figure 27. btn_1 event handler

7.2 Example

This section demonstrates an example of this implementation to add voice support to the GUI Guider, toggling the LED widget and changing between GUI screens.

1. Using the GUI template created with the button, add the widgets. For example, add an LED widget.

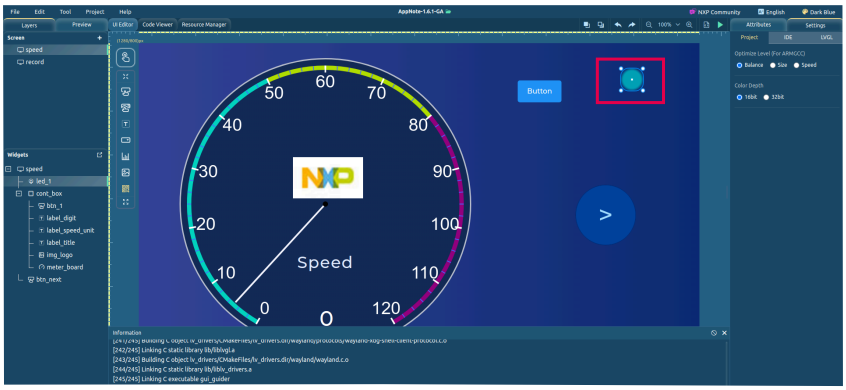

Figure 28. Added the LED widget

2. Add the event **pressed** to the btn_1 and to change the background add the configuration of the event. For this case, the background must be selected as black to "turn off" the LED widget. Therefore, the event used is **pressed > led_1 > Background black (#000000)**.

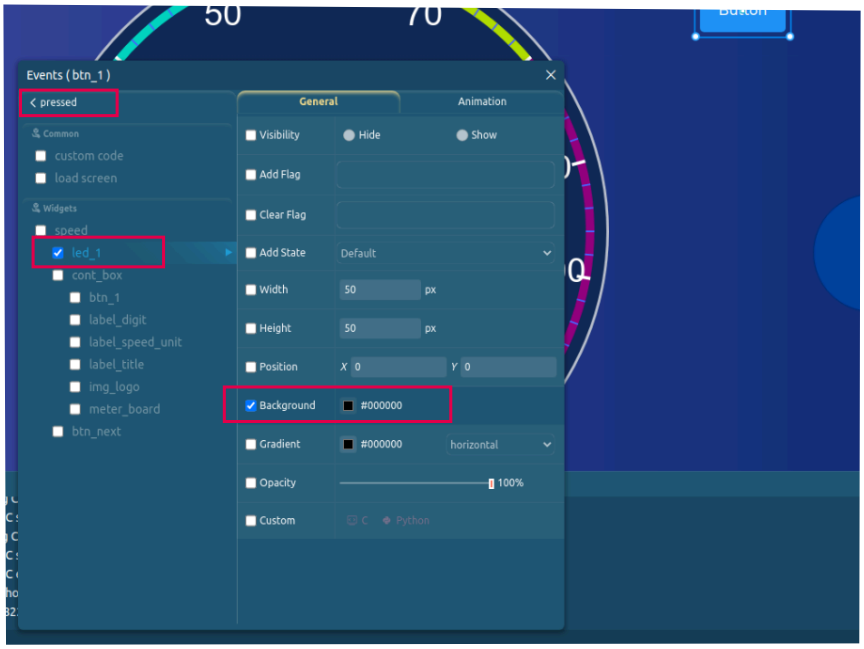

Figure 29. Changing the LED background through events to turn off

3. Using the same button, configure an event to assign it to "turn on". For this case, add the event **released** to the btn_1 and add red to the background. Therefore, the event used is **released > led_1 > Background red (#ff0000)**.

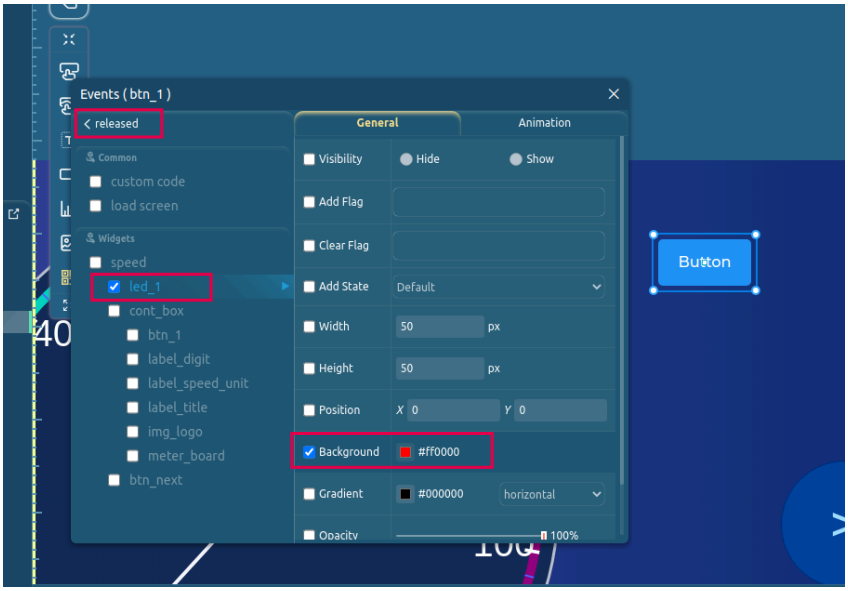

Figure 30. Changing the LED background through events to turn on

4. Once the GUI is created, add `command_handler.c` and `command_handler.h` to the custom/folder.
5. To create the link between events and VIT, add the following lines in `custom_init()` inside the file in `custom/custom.c`. To change between screens, add two more events by linking btn_1 to change to screen 2.

```
void custom_init(lv_ui *ui)
  {
      /* Add your codes here */
      start_command_handler();
      command_handler_link(HEY_LED, TURN_OFF, &ui->speed_btn_1, LV_EVENT_PRESSED);
```

```
        command_handler_link(HEY_LED, TURN_ON, &ui->speed_btn_1, LV_EVENT_RELEASED);

        command_handler_link(HEY_NXP, NEXT, &ui->speed_btn_1, LV_EVENT_CLICKED);
        command_handler_link(HEY_NXP, RETURN, &ui->record_btn_before ,LV_EVENT_CLICKED);


}
```

Where:

- The wakeword `HEY_LED` and command `TURN_OFF` combination is assigned to turn off the LED. In other words, change the background to black.
- The wakeword `HEY_LED` and command `TURN_ON` combination is assigned to turn the LED red.
- The wakeword `HEY_NXP` and command `NEXT` combination is assigned to change between screens using the event assigned all to btn_1, and using btn_before in screen 2.
- The wakeword `HEY_NXP` and command `RETURN` combination is assigned to return to screen 1.

6. Select **Project > Build > Yocto** and build the project.

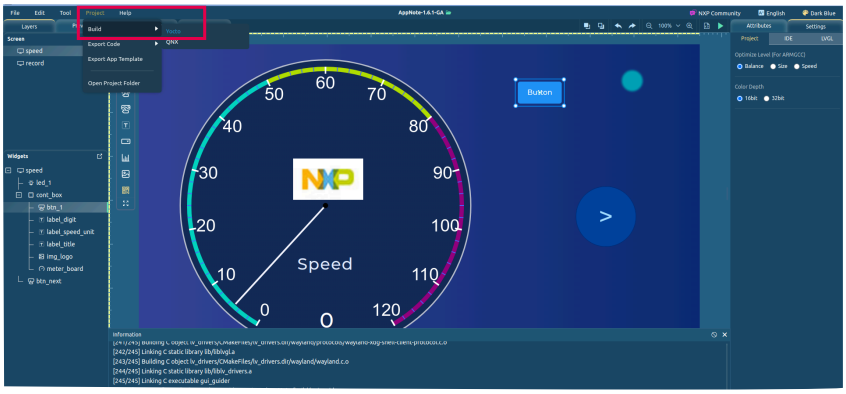

Figure 31. Build the project

7. Sent the new binary to the EVK.

Note: *The information log provides the binary location.*

```
scp <binary location> root@<evk ip>:/home/root
```

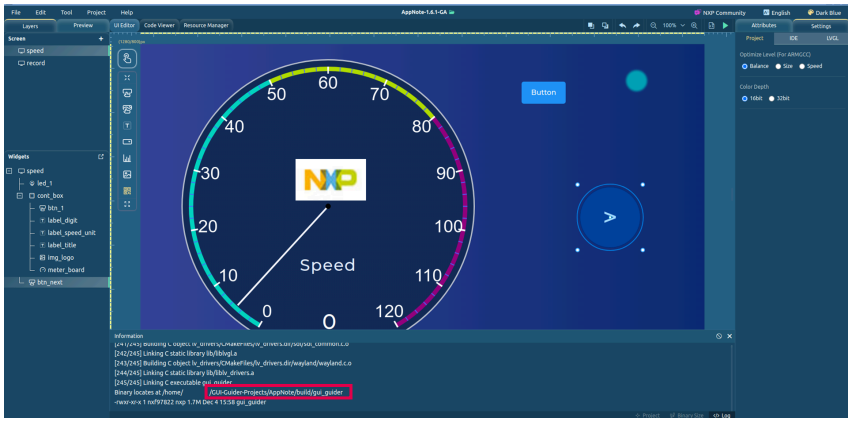

Figure 32. Information log with binary location example

7.3 Testing and configuration

Once the download has finished, perform the following steps on the EVK:

1. Verify that the `snd-aloop` module is already loaded by running `lsmod`. If the module is not found, load it using the following command:

```
root@imx93evk:~#  sudo modprobe snd-aloop
```

2. Run `voice_ui_app` using the following command:

```
root@imx93evk:~#  ./voice_ui_app -notify &
```

   Where:
   - The `-notify` is used to send a notification to `WakeWordNtfy` and `WWCommandNtfy`.
     Note: *Remember to copy `WakeWordNtfy` and `WWCommandNtfy` to `usr/bin`.*
   - The `&` is used to run in the background.
3. Verify that the VIT engine is set on the `Config.ini`.
4. Run AFE with `libvoiceseekerlight` in the background:

```
root@imx93evk:~#  cd /unit_tests/nxp-afe/
root@imx93evk:~#  ./afe libvoiceseekerlight &
```

5. Open the GUI Guider application using the following command:

```
root@imx93evk:~#  ./gui_guider
```

   Until this step, the LVDS screen, or HDMI displays the GUI created.

Figure 33. GUI example with widget LED turned on

6. Try using a previously assigned wakeword and voice command, for example, say "Hey NXP" and "Turn off". After saying the command for power off, depending on the callback assigned, GUI Guider performs an action. For this example, GUI Guider changes the background color for the LED widget.

Figure 34. GUI example with widget LED turned off

8 Related resources

Table 2 lists some additional resources used to supplement this document.

Table 2. Related resources

Resource	Link/how to obtain
i.MX 93 Applications Processor Family – Arm Cortex-A55, ML Acceleration, Power Efficient MPUNXP i.MX 93 A1 (i.MX93)	https://www.nxp.com/products/processors-and-microcontrollers/arm-processors/i-mx-applications-processors/i-mx-9-processors/i-mx-93-applications-processor-family-arm-cortex-a55-ml-acceleration-power-efficient-mpu:i.MX93
Embedded Linux for i.MX Applications Processors (IMXLINUX)	http://www.nxp.com/IMXLINUX
GUI Guider v1.6.1 User Guide (GUIGUIDERUG)	https://www.nxp.com/docs/en/user-guide/GUIGUIDERUG-1.6.1.pdf
VIT i.MX voiceUI repository	https://github.com/nxp-imx/imx-voiceui

9 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2023-2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES

OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

10 Revision history

Table 3 summarizes the revisions to this document.

Table 3. Revision history

Document ID	Release date	Description
AN14270 v.1.0	16 May 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

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.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. 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 B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

i.MX — is a trademark of NXP B.V.

Contents

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

Date of release: 16 May 2024
Document identifier: AN14270