AN14015

Implement Camera Preview with Framework Enabled on SDK

Rev. 2 — 1 February 2024 Application note

Document information

Information	Content
Keywords	Smart HMI, smart TLHMI, framework
Abstract	This application note describes how to enable framework to implement camera video showing on display panel on SLN-TLHMI-IOT board based on SDK.

1 Overview

NXP has launched a solution development kit named SLN-TLHMI-IOT, which focuses on smart HMI applications. It enables smart HMI with ML vision, voice, and graphics UI implemented on one NXP i.MX RT117H MCU. Its software framework supports flexible designs and customization of vision and voice functions. To help the users to use the software platform better, some basic documents are provided, for example, the software development user guide. The guide introduces the basic software design and architecture of the applications covering all components of the solution, including a framework to help the developers more easily and efficiently implement their applications using the SLN-TLHMI-IOT.

For more details about the solution and relevant documents, visit:

NXP EdgeReady Smart HMI Solution based on i.MX RT117H with ML Vision, Voice and Graphical UI | NXP Semiconductors

However, it is still not so easy for the developers to implement their applications referring to these basic guides. Based on the SDK, the solution software is constructed on a design called framework. That is, the framework is the kernel of the software. Introducing a simple typical example with a framework is the best entry to help study the development of the framework.

This application note describes how to enable a framework to implement camera video showing on the display panel of the SLN-TLHMI-IOT board based on SDK. By implementing the simple camera preview example of the MCUXpresso IDE, this document introduces how to:

- Create a C++ project with the SDK resources on MCUXpresso IDE.
- Build board hardware support for SLN-TLHMI-IOT board.
- Enable the framework on SDK.
- Implement an application on the framework.

This document helps the developers be able to:

- Understand the framework and the solution software more deeply.
- Develop their application from 0 to 1.
- Implement their application with more own features based on the example project.

For the software package of this application note, refer to https://mcuxpresso.nxp.com/appcodehub.

1.1 Framework overview

The solution software is primarily designed around the use of a "framework" architecture that is composed of several different parts:

- Device managers – The core part
- Hardware Abstraction Layer (HAL) devices
- Messages/Events

The mechanism of the framework is as shown in Figure 1 and it is explained below.

Device managers are responsible for "managing" devices used by the system. Each device type (input, output, and so on) has its own type-specific device manager. With a device manager starting after the devices being registered to it, it waits and checks a message to transfer data to the devices and other managers after initializing and starting the registered devices.

The HAL devices are written "on top of" lower-level driver code, helping to increase code understandability by abstracting many of the underlying details.

Events are a means by which information is communicated between different devices via their managers. When an event is triggered, the device that first received the event communicates that event to its manager. Then, in turn, it notifies other managers designated to receive the event.

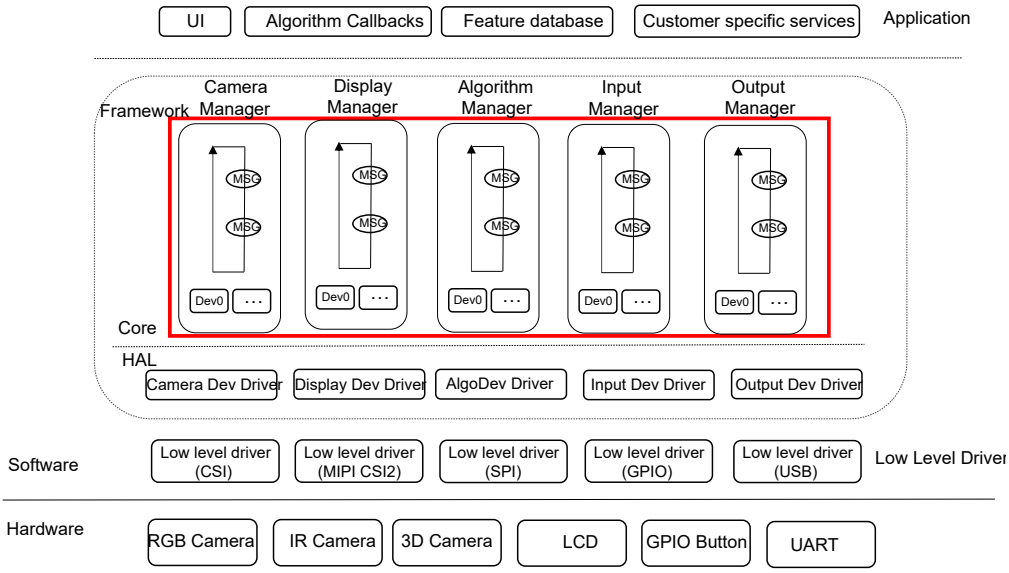

Figure 1. Smart HMI software architecture on framework

The architectural design of the framework is centered on the following three primary goals:

- Ease-of-use
- Flexibility/Portability
- Performance

The framework is designed with the goal of speeding up the time to market for vision and other machine-learning applications. To ensure a speedy time to market, it is critical that the software itself is easy to understand and modify. Keeping this goal in mind, the architecture of the framework is easy to modify without being restrictive, and without coming at the cost of performance.

For more details about the framework, refer to MCU-SMHMI-SDUG.

2 Development environment

First, prepare and set up the hardware and software environment for implementing the camera preview example of the framework.

- **Hardware environment**
  The hardware environment is set up for verifying the example:
  – The smart HMI development kit based on NXP i.MX RT117H (SLN-TLHMI-IOT kit)
  – SEGGER J-Link with a 9-pin Cortex-M adapter and V7.84a or higher
- **Software environment**
  The software environment is set up for developing the example:
  – MCUXpresso IDE V11.7.0
  – RT1170 SDK V2.13.0
  – SLN-TLHMI-IOT software V1.1.1 – smart HMI source codes released on NXP GitHub repository

For details about the acquirement and setup of the software environment, refer to Getting Started with the SLN-TLHMI-IOT | NXP Semiconductors.

3 Implement camera preview on framework

After the development environment is ready, begin the process of implementing the camera preview example (the example for short later) with the framework on SLN-TLHMI-IOT board. It starts from creating a C++ project on MCUXpresso IDE.

3.1 Create a C++ project with the SDK resources on MCUXpresso IDE

A C++ project is required because C++ design is used for some components in the solution, such as face recognition algorithm. There is a wizard to do it in MCUXpresso. Follow it to create the C++ project with the SDK software components for examples after the SDK V2.13.0 is installed in MCUXpresso IDE:

1. Choose to create a C/C++ project to enter the wizard.
2. Select MCU **MIMXRT1170** and then the board **evkmimxrt1170**, as shown in Figure 2.

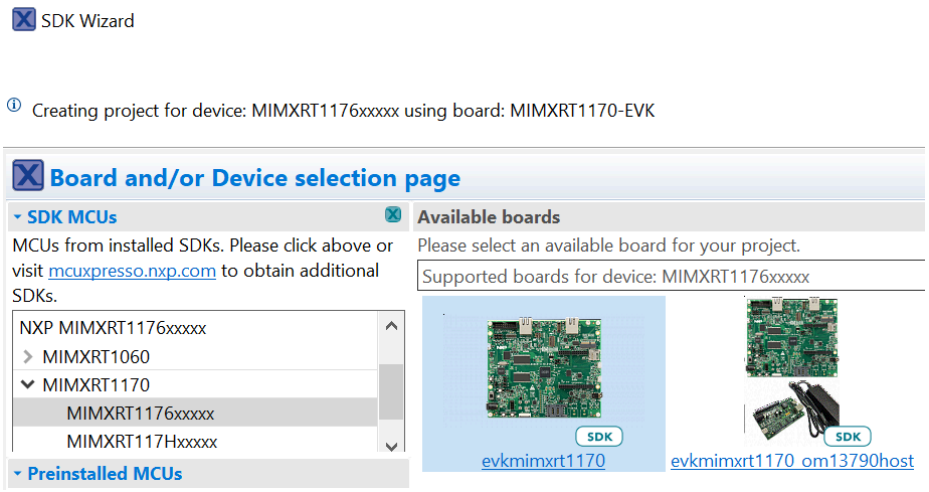

Figure 2. Select a device and board for creating a project

3. Click **Next** to go to the **Configure the project** page (see Figure 3).
   - Specify a name for the project. It is `sln_tlhmi_iot_camera_preview_cm7` for the example.
   - Select the project type as C++ Project.
   - Select the cores as `cm7`.

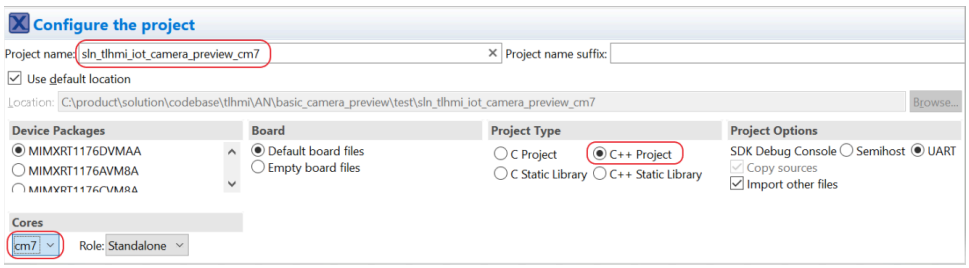

Figure 3. Configure the project

4. Select SDK software components for the project. According to the solution and the example, select or add the below components:
   - Selected FreeRTOS kernel for operating system.
   - Add the drivers: `cache`, `csi`, `i2c`, `i2c-freertos`, `lcdifv2`, `mipi_csi2rx`, `mipi_dsi`, `pit`, `pxp`, `soc_src` for the system, camera, LCD, and display drives.
     Note: *There are already some default drivers in the wizard. Keep them.*

- Select the components of the abstraction layer for the camera and display controls in the example. The components are summarized in Figure 4.

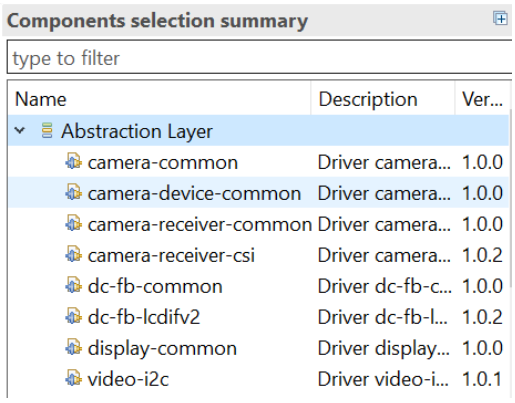

Figure 4. Components for Abstraction Layer

- Add the board component, `display-hx8394`, for display panel initialization.
- Add the software component, `display-mipi-dsi-cmd`, for display panel interface control adapter.

5. Click **Next** to go to the last page – Advanced project settings. Use the default settings to check **Redirect printf/scanf to UART** for log print later, though it is not required for this example.
6. Click **Finish** to create the basic C++ project with the SDK software components for the camera preview example.

Note: *The SDK software components could still be added from the SDK package if some are missed during creating the project.*

3.2 Build board hardware support for SLN-TLHMI-IOT board

When the project is created, only the basic system hardware, such as clock, and debug UART pins are built in the project. Besides it, as introduced above, the demo board evkmimxrt1170 is chosen as the board support when creating the project. However, the SLN-TLHMI-IOT board is used for this example and smart HMI solution. There are some different devices and pins used for both boards. Therefore, it is required to add and update some board hardware support for the example.

The related hardware devices on the SLN-TLHMI-IOT board are:

- Camera sensor - GC2145 with MIPI CSI-2 interface
- Display panel - RK055MHD091 (hx8394) with MIPI-DSI interface
- SDRAM - W9825G6KH
- Flash - W25Q256JV

Add board hardware support for the SLN-TLHMI-IOT board as follows:

1. The board level of drivers is implemented under the *board* folder involved with camera, display, SDRM, and Flash as below:
   - Add the *board_define.h* file used in the smart HMI solution in the *board* folder for enabling and configuring HAL devices:
     a. Clone the file *board_define.h* under *\coffee_machine\cm4\board* to the board folder of the project. Using the file under *cm4* instead of *cm7* is because the camera and display devices are enabled on the M4 core.
     b. Comment the below component definitions in the file since they are not used:

```
#define ENABLE_DISPLAY_DEV_LVGLCoffeeMachine
#define ENABLE_OUTPUT_DEV_RgbLed
#define ENABLE_INPUT_DEV_PushButtons
```

```
#define ENABLE_INPUT_DEV_ShellUsb
#define ENABLE_OUTPUT_DEV_UiCoffeeMachine
#define ENABLE_LPM_DEV_Standby
```

c. Add `#define ENABLE_DISPLAY_DEV_Lcdifv2Rk055mh` to enable display panel.
d. Change `#define WIFI_ENABLED 1` to 0 to disable the Wi-Fi support.

- Modify **pin_mux.c** under the *board* folder for pin settings of the camera and display:
  - Include *board_define.h* and *board.h* files.
  - To initialize the camera and display panel pins, copy the `BOARD_InitMipiCameraPins()` and `BOARD_InitMipiPanelPins()` functions from *\coffee_machine\cm4\board\pin_mux.c* to the *pin_mux.c* of the project.
  - Add the calls of `BOARD_InitMipiCameraPins()` and `BOARD_InitMipiPanelPins()` in the `BOARD_InitBootPins()` function and modify `BOARD_InitPins()` to support LPUART12 on the SLN-TLHMI-IOT board instead of LPUART1 on the *evkmimxrt1170* board.
- Modify *pin_mux.h* to declare the `BOARD_InitMipiCameraPins()` and `BOARD_InitMipiPanelPins()` APIs.
- Modify *board.c* to add camera device control including enabling FreeRTOS I^2C support and update MPU configurations for the SDRAM and flash memory. There is already a macro definition, `SDK_I2C_BASED_COMPONENT_USED`, in the generated *board.c* file for I^2C enablement. Referring to smart HMI solution software, another macro definition, `SDK_I2C_FREERTOS`, is used for I^2C enablement on FreeRTOS:
  Note: *Below codes for the camera control including the FreeRTOS I^2C support are copied from the board.c in the smart HMI solution.*
  - To enable the FreeRTOS I^2C support for camera, add the settings `SDK_I2C_BASED_COMPONENT_USED=1` and `SDK_I2C_FREERTOS=1` on **Project** > **Properties** > **C/C++ + Build** > **settings** > **Tool Settings** > **MCU C compiler** > **Preprocessor**.
  - Include the header file, *fsl_lpi2c_freertos.h* with the `SDK_I2C_FREERTOS` definition.
  - Add the variables related to FreeRTOS I^2C:

```
s_lpi2cIrqs[] and s_masterRTOSHandle[sizeof(s_lpi2cIrqs) /
 sizeof(s_lpi2cIrqs[0])]
```

  - Modify the functions `BOARD_LPI2C_Init()`, `BOARD_LPI2C_Send()`, and `BOARD_LPI2C_Receive()` to enable FreeRTOS I^2C support.
  - Add the implementations of the functions related to MIPI camera device control: `BOARD_MIPICameraI2CInit()`, `BOARD_MIPICameraI2CReceive()`, `BOARD_MIPICameraI2CSend()`, `BOARD_MIPICameraPullResetPin()`, `BOARD_MIPICameraPullPowerDownPin()`.
  - Modify the function `BOARD_ConfigMPU()` under the `__CORTEX_M == 7` definition of Flash and SDRAM memory MPU configurations:
    - Change the size of SDRAM MPU from `ARM_MPU_REGION_SIZE_64MB` to `ARM_MPU_REGION_SIZE_32MB`.
    - Change the size of Flash MPU from `ARM_MPU_REGION_SIZE_16MB` to `ARM_MPU_REGION_SIZE_64MB`.
  - To enable SDRAM MPU support, set **USE_SDRAM** in **Project** > **Properties** > **C/C++ Build** > **settings** > **Tool Settings** > **MCU C compiler** > **Preprocessor**.
- Modify **board.h** for the definitions and configurations related to hardware devices of the SLN-TLHMI-IOT board.
  - Include the file *board_define.h*.
  - Redefine the board name to yours. Here, the name is **SLN-TLHMI-IOT** and the value of `DEBUG_CONSOLE_UART_INDEX` to 12 as LPUART12 is used on the SLN-TLHMI-IOT board.
  - Set `BOARD_FLASH_SIZE` to `0x4000000` (64 MB).

  - Add MIPI Camera configurations about I^2C, reset pin, and power down pin used in the *board.h* of smart HMI solution software, for example:

```
#define BOARD_MIPI_CAMERA_I2C_BASE  LPI2C6
```

  - Modify the settings of MIPI display panel pins and back light pins referring to the *board.h* of smart HMI solution software, for example:

```
#define BOARD_MIPI_PANEL_RST_PIN 22
```

- Modify the file **dcd.c** to set up SDRAM. Replace the array **dcd_data[ ]** with the one in **dcd.c** of smart HMI software.
- About the display panel control at the board level, it is implemented in the *display_support.c* and *display_support.h* files of SDK. To support it in the example project,
  - copy both files from *boards\evkmimxrt1170\display_examples\fbdev_freertos\cm7* of the SDK package to the *board* folder.
  - Since the display panel RK055MHD091 is enabled in *display_support.h*, update nothing but enable the back light of the display panel by setting **0** to the back light pin in the function: `GPIO_PinWrite` (`BOARD_MIPI_PANEL_BL_GPIO`, `BOARD_MIPI_PANEL_BL_PIN`, 0);

2. Add the initialization and settings of the camera sensor – GC2145 that is supported in smart HMI solution:
   - Copy **sln_gc2145.c** and **sln_gc2145.h** from *coffee_machine\cm4\video* in the smart HMI solution to the *video* folder.
   - Add the macro definitions related to the settings of the camera sensor, such as `kCAMERA_DeviceMonoMode`, in the `enum _camera_device_cmd` in *fsl_camera_device.h*. Copy them from the same location of the smart HMI solution.
3. Modify the initialization and settings of the Flash in the structure `qspiflash_config` in *xip* ***evkmimxrt1170_flexspi_nor_config.c***:
   - Set the member `sflashA1Size` to `BOARD_FLASH_SIZE` (redefined in *board.h*). Therefore, add `#include "board.h"` for using the definition.
   - Replace the content of the member lookup table with the one of the smart HMI solutions.
   - Change the value of the member blockSize from 64 u * 1024 u to 256 u * 1024 u.

3.3 Enable framework on SDK

The current software consists of SDK software components. The framework can be added smoothly to it as an SDK software component.

Generally, it is required to implement the related HAL device drivers to enable the framework for an application. Benefiting from the standard design of the HAL drivers and the plenty of different types of HAL device resources supported in the framework, the development of a new HAL driver is not difficult. This is one of the significant performances of the framework.

1. Add the framework component:
   - Copy the *framework* folder in the smart HMI to the root directory of the example folder *sln_tlhmi_iot_camera_preview_cm7*.
   - To add the new folder **framework** for building in the project, check out **Exclude resource from build** in **C/C++ Build** > **Settings** after opening the **Properties** for the **framework** folder on MCUXpresso.
   - Add the below include paths about framework for the camera preview example in **Project** > **Properties** > **C/C++ Build** > **settings** > **Tool Settings** > **MCU C compiler** > **Includes** and **MCU C++ compiler** > **Includes**:

```
"${workspace_loc:/${ProjName}/framework/inc}"
"${workspace_loc:/${ProjName}/framework/hal_api}"
```

```
"${workspace_loc:/${ProjName}/framework/hal}"
```

- No source codes of the framework core are provided in the smart HMI software V1.1.1 from GitHub. Therefore, no *core* folder can be seen in the *framework* group but the library - *libframework_cm7.a* is present under *\coffee_machine\cm7\libs\framework_cm7\Release*.
  Add the library as below:
  – Copy the library to *\libs\framework_cm7\Release* in the example after creating the folders on the path.
  – To build the new *libs* folder in the project, check out **Exclude resource from build** in **C/C++ Build** > **Settings** after opening the **Properties** for the **libs** group on MCUXpresso.
  – Add the library and library search path in **Project** > **Properties** > **C/C++ Build** > **settings** > **Tool Settings** > **MCU C++ Linker** > **Libraries**, as shown in Figure 5.

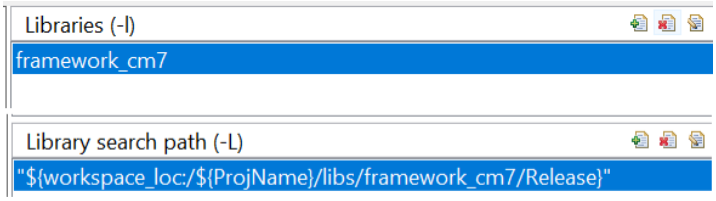

Figure 5. Settings for the framework core library

Note: *If there are source codes of the framework core, this step to add a library is not required.*

2. In the example, the HAL drivers for the camera and display are required to drive the camera video displaying on the display panel. The HAL driver of the camera GC2145 has been supported in the *hal_camera_mipi_gc2145.c* file under the *hal\camera* of the current framework. However, there is no HAL driver for the display panel RK055MHD091 but for RK055AHD091. Actually, the HAL driver for RK055MHD091 is same to RK055AHD091. So, the development is easy:
   - Clone the *hal_display_lcdifv2_rk055ahd091.c* file for the HAL driver of RK055AHD091 and rename it to *hal_display_lcdifv2_rk055mhd091.c* under *framework\hal\display*.
   - Replace all strings `055ah` strings with `055mh` in the new file.
   - Change `DISPLAY_DEV_Lcdifv2Rk055mh_LEFT` to `DISPLAY_DEV_Lcdifv2Rk055mh_START_X` and `DISPLAY_DEV_Lcdifv2Rk055mh_TOP` to `DISPLAY_DEV_Lcdifv2Rk055mh_START_Y` in `DISPLAY_InitDisplay()`. function. The macro definitions are used for defining the start coordinate of a camera preview on the display panel.
   - Define the related configurations of the display panel used in `DISPLAY_InitDisplay()` function and `s_DisplayDev_Lcdif` structure into the *board_define.h* file referring to the definitions of `ENABLE_DISPLAY_DEV_LVGLCoffeeMachine` in the file.

   Note:
   There is a small extra job about the framework for this example application. That is, move `#include "fwk_platform.h"` *outside the definition* `#ifdef LOG_ENABLE` *in fwk_log.h and add* `#include "fwk_log.h"` *in fwk_common.h.*
   The reason is: The header file fwk_paltform.h under the `#ifdef LOG_ENABLE` *is not included because the log is not enabled in the example application. It causes the building error without including the header file. However, there is no problem for the smart HMI solution since the log is enabled in it.*

3.4 Implement an application on framework

With the framework enabled, the example application can be implemented on it. Besides, the example application (also the smart HMI solution application) works on FreeRTOS and the FreeRTOS is combined with the framework. Therefore, to implement the example application, it is also required to do some modifications about FreeRTOS at the application level.

1. Modify the main file, *source**sln_tlhmi_iot_camera_preview_cm7.cpp*, for implementing the example application on framework:
   - Under the *comment /* TODO:* insert other include files here. */, include the header files related to framework and hardware, for example:

     ```
     #include "fwk_display_manager.h"
     ```

   - Under the comment */* TODO:* insert other definitions and declarations here. `*/`, add the device declarations of the camera, display, and `pxp` (used for display), for example, HAL_CAMERA_DEV_DECLARE(MipiGc2145) and the definitions of the task priorities of the camera and display, for example:

     ```
     #define CAMERA_MANAGER_TASK_PRIORITY 2, with C++ format.
     ```

   - Add the implementation of the functions `APP_InitFramework()` for the initialization of camera and display managers, `APP_RegisterHalDevices()` for camera, display, and `pxp` devices registration and `APP_StartFramework()` for starting the camera and display managers.
   - Add the implementation of the function `APP_BoardInit()` for the initialization of board hardware:
     – Move the board initialization calls from `main()` to the function.
     – Add the `SRC_AssertSliceSoftwareReset()` function to reset the display and `Time_Init(1)` to initialize the timer for supporting FreeRTOS and framework run-time counter. To enable it:
       – Include the header file, *sln_time.h*, in *sln_tlhmi_iot_camera_preview_cm7.cpp*.
       – Copy *sln_time.c* and *sln_time.h* from *\coffee_machine\cm7\utilities* of smart HMI solution to the *utilities* folder of the example software.
   - Modify the function `main()` of which the handling is a standard process – call `APP_BoardInit()`, `APP_InitFramework()`, `APP_RegisterHalDevices()`, and `APP_StartFramework()` in sequence. Finally, start the task scheduler by calling the API `vTaskStartScheduler()` of FreeRTOS.
2. To configure the FreeRTOS for following the framework and the solution application:
   - Update the below definitions in *FreeRTOSConfig.h* under *source*\ *referring* to the one in *[smart HMI]*\ *coffee_machine**cm7**config_file*\ (and *cm4**source*\):
     – Change `configMAX_PRIORITIES` from 5 to 8.
     – Set `configUSE_TIME_SLICING` to 1.
     – Add `#define configFRTOS_MEMORY_SCHEME` 4.
     – Set `configSUPPORT_STATIC_ALLOCATION` to 1.
     – Increase `configTOTAL_HEAP_SIZE` to ((`size_t`)(1024 * 1024 * 2)).
     – Set `configCHECK_FOR_STACK_OVERFLOW` to 1.
     – Set `configUSE_MALLOC_FAILED_HOOK` to 1.
     – Set `configGENERATE_RUN_TIME_STATS` to 1.
     – Set `configUSE_STATS_FORMATTING_FUNCTIONS` to 1.
     – Increase `configTIMER_TASK_STACK_DEPTH` to (1024).
     – Set `INCLUDE_uxTaskGetStackHighWaterMark` to 1.
   - With some features related to memory and time enabled in *FreeRTOSConfig.h*, add the corresponding APIs support:
     – With `configGENERATE_RUN_TIME_STATS` enabled, enable the FreeRTOS APIs of `portCONFIGURE_TIMER_FOR_RUN_TIME_STATS()` and `portGET_RUN_TIME_COUNTER_VALUE()` with the related functions in *sln_time.c* in *FreeRTOSConfig.h* for supporting runtime statistic.
     – Copy *os_hooks.c* from *[smart HMI]\coffee_machine\cm7\source* to the *source* folder of the example software for some enabled memory features and FreeRTOS hook functions support.

Note: ***Remark****: Remove the* `pvPortCalloc()` *function in os_hooks.c as it has been implemented in the v10.5.0 of FreeRTOS kernel used in the example software while not in v10.4.3 for the smart HMI v1.1.1.*

3. Configure the example project of the memory assignment and memory section settings for the application based on the board hardware.
   - Update the memory assignment on **Project** > **Properties** > **C/C++ Build** > **MCU settings**, as shown in Figure 6.

Memory details (MIMXRT1176xxxxx)*

Default LinkServer Flash Driver

Type	Name	Alias	Location	Size
Flash	BOARD_FLASH	Flash	0x30000000	0x800000
RAM	BOARD_SDRAM	RAM	0x80000000	0x1000000
RAM	NCACHE_REGION	RAM2	0x81000000	0x400000
RAM	SRAM_DTC_cm7	RAM3	0x20000000	0x80000
RAM	SRAM_ITC_cm7	RAM4	0x0	0x40000
RAM	SRAM_OC1	RAM5	0x20240000	0x80000
RAM	SRAM_OC2	RAM6	0x202c0000	0x80000
RAM	SRAM_OC_ECC1	RAM7	0x20340000	0x10000
RAM	SRAM_OC_ECC2	RAM8	0x20350000	0x10000
RAM	SRAM_OC_cm7	RAM9	0x20360000	0x20000

Figure 6. Memory assignment for the camera preview example

   - Configure the extra linker script input sections as some memory sections are assigned for the specific codes and data in the application. May refer to the coffee machine app in the smart HMI. The configurations in **Project** > **Properties** > **C/C++ Build** > **Settings** > **Tool Settings** > **MCU C++ Linker** > **Managed Linker Script** for the example project is as shown in Figure 7.

Extra linker script input sections

Input section description	Region	Section Type
*(.NonCacheable.init)	NCACHE_REGION	.data
*(NonCacheable)	NCACHE_REGION	.bss
*(.bss.fwk_section)	SRAM_DTC_cm7	.bss
*(.ocram_non_cacheable_data)	SRAM_DTC_cm7	.bss
*(.CodeQuickAccess)	SRAM_ITC_cm7	.data

Figure 7. Memory sections settings for the camera preview example

Therefore, the camera preview example can run on the SLN-TLHMI-IOT board after building and programming. The basic mechanism of the camera preview on framework is:

After getting the event from the camera low driver of SDK with receiving camera frame, the camera HAL sends a message to inform the camera manager to get the frame data from the camera HAL. The display manager requests camera frames by sending a message to the camera manager, and transfers the frame data to the display HAL. The display HAL calls the display driver of SDK to display the camera frame on the LCD panel.

Note: *For more details about the modifications introduced above, check the example software at https://mcuxpresso.nxp.com/appcodehub.*

3.5 Summary

To understand the development quicker and better, below is a brief summary about the key points:

1. **Create a C++ project**
   It is easy following the wizard. The main job is to select the **SDK software components** including:
   - operating system

- drivers
- abstraction layer
- board components
- software components

2. **Build board hardware**
   This is the biggest workload in the development as it is involved with multiple larger devices: *camera*, *display panel*, *SDRAM*, and *Flash*. However, the workload is not so much with the plenty of resources from SDK.
   The files to be updated are:
   - Mainly under *board*\:
     – *pin_mux.c/.h* – for the pin settings of camera and display
     – *board.c/.h* – for the configurations of camera interface and MPU
     – *dcd.c* – for the initialization of SDRAM
     – *display_support.c/.h* - cloned from SDK software package almost without any modification for the configuration of display panel interface
     – *board_define.h* - cloned from smart HMI and do some configurations for the HAL device related to camera and display
   - Under *xip*\:
     – *evkmimxrt1170_flexspi_nor_config.c* – for XIP flash configurations
   - Under *video*\:
     – *sln_gc2145.c/.h* – cloned from smart HMI without any modification for camera sensor control.
3. **Enable framework**
   This is the key job in the development since the application is built on it. Generally, to enable the framework on SDK, it includes:
   - Add framework to SDK as an SDK software component.
   - Implement the related HAL device drivers by cloning and do some modifications under the standard configuration and APIs.
     In the example app, the display HAL driver is implemented by:
   - Cloning the *hal_display_lcdifv2_rk055ahd091.c*.
   - Changing the string `rk055ahd` to `rk055mhd`.
   - Configuring the related parameters to the display HAL device.
4. **Implement the example application**
   To implement the example application on the framework, the files to be updated are under *source*\:
   - *sln_tlhmi_iot_camera_preview_cm7.cpp* – modify it to implement
     – `APP_InitFramework()`
     – `APP_RegisterHalDevices()`
     – `APP_StartFramework()`
     After they are implemented, update the HAL devices and device managers in them when more or different devices are required with the new application requirements.
   - *FreeRTOSConfig.h*
   - *os_hook.c* (cloned from the smart HMI)

4 Verifications with the example project

Visit https://mcuxpresso.nxp.com/appcodehub and get the example software package for this application note.

Open the example project on MCUXpresso IDE. Build and program the *.axf* file to the address `0x30000000`. The real-time video streams captured by the camera are shown on the display panel on the SLN-TLHMI-IOT board. The camera preview example works successfully.

5 Note about the source code in the document

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

Copyright 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 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.

6 Revision history

Table 1 summarizes the revisions to this document.

Table 1. Revision history

Revision number	Release date	Description
2	1 February 2024	Updated Section 4.
1	14 September 2023	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.

Contents

Date of release: 1 February 2024
Document identifier: AN14015