AN15175

i.MX RT700 Custom MIPI Display Driver Integration

Rev. 1.0 — 28 October 2026 Application note

Document information

Information	Content
Keywords	AN15175, i.MX RT700, MIPI
Abstract	This document describes the recommended display bring-up sequence required to obtain valid image output.

1 Introduction

The i.MX RT700 features up to five computing cores designed to power smart AI-enabled edge devices such as wearables, consumer medical, smart home, and Human Machine Interface (HMI) devices. Its compute subsystem includes a primary Arm Cortex-M33 running at 325 MHz. Includes 2.5D GPU with vector graphics acceleration and frame buffer compression and the NXP's eIQ Neutron NPU accelerating AI workloads by up to 172x and integrates up to 7.5 MB of onboard Static Random Access Memory (SRAM).

The device includes advanced display capabilities through its integrated display controller and Mobile Industry Processor Interface (MIPI) Display Serial Interface (DSI) interface, enabling efficient connectivity to a wide range of modern display panels.

This application note is intended to guide developers through the high-level process of initializing and enabling custom MIPI displays on the i.MX RT700. The document describes the recommended display bring-up sequence required to obtain valid image output, highlighting both the standard controller-driven data path and optional panel-specific initialization mechanisms. As a practical reference, the described flow is demonstrated using a Waveshare MIPI display, serving as an example of how these concepts are applied to a real-world display module.

Specifically, this application note covers:

- A high-level overview of the i.MX RT700 display subsystem architecture, including the display controller, MIPI DSI host, and PHY.
- The recommended initialization flows to bring up an MIPI display and render an image.
- Frame buffer configuration and how display data is autonomously streamed over the MIPI interface.
- Practical considerations and guidelines for integrating custom MIPI displays, illustrated using a Waveshare display module on the i.MX RT700-based designs.

1.1 Target audience

This guide is intended for embedded software engineers and hardware designers who are integrating custom MIPI DSI displays into the i.MX RT700-based designs. Familiarity with C programming, embedded systems concepts, and the MIPI DSI protocol is assumed. Prior experience with the MCUXpresso SDK is beneficial but not required.

2 Hardware and software requirements

The following hardware and software components are required to reproduce the display initialization process described in this application note.

2.1 Hardware

- **i.MX FRDM RT700 board**
- i.MX RT700 evaluation board
  - Used as the target platform for display controller and MIPI DSI configuration.
  - Product information: https://www.nxp.com
- **Waveshare 7-inch MIPI DSI Display (1024 × 600)**
  - Reference display used throughout this application note.
  - The procedure can be adapted to other MIPI displays with different timing requirements.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

2.2 Software

- **MCUXpresso for Visual Studio Code**
  - Development environment used to build, debug, and deploy the reference software.
  - MCUXpresso for Visual Studio Code
- **MCUXpresso SDK for i.MX RT700**
  - Provides the LCD Interface/Display Controller (LCDIF), MIPI DSI host, DPHY, panel driver, and board support software used in this application note.
  - https://mcuxpresso.nxp.com

2.3 Reference example

The implementation described in this document is based on the SDK (version 26.9.0) display example:

`Driver_examples/lcdif_rgb565_cm33_core0`

Import this example into VS Code using the MCUXpresso extension and select Freestanding application when prompted for the project type.

The application demonstrates LCDIF frame generation and transmission through the MIPI DSI interface to a connected display panel.

Panel-specific implementations can be found in the display component drivers included in the SDK display middleware.

3 i.MX RT700 display architecture

This section describes the i.MX RT700 display architecture.

3.1 Display data path

The i.MX RT700 display pipeline consists of three main hardware blocks operating in sequence:

1. LCDIF (LCD Interface/Display Controller) - Fetches pixel data from frame buffers in PSRAM or SRAM, applies pixel formatting, and generates the Display Pixel Interface (DPI) video timing signals including HSYNC, VSYNC, DE (data enable), and pixel clock.
2. MIPI DSI Host - Accepts the DPI video stream from the LCDIF and packetizes it into MIPI DSI long packets. The host manages the virtual channel arbitration, End of Transmission Packet (EoTp) insertion, and the command channel for sending DCS commands directly to the panel.
3. MIPI DPHY - The physical layer that drives the differential D-PHY lanes at high speed (HS mode) for video data and at low speed (LP mode) for control signaling and escape commands.

3.2 Display hardware blocks

Key hardware blocks involved in display operation:

Table 1. Display hardware blocks

Block	Description
LCDIF	Display controller. Reads frame buffers from memory and generates DPI video timing.
MIPI DSI Host	Serializes DPI pixel data into MIPI DSI packets and provides a command channel for DCS/vendor commands.
MIPI DPHY	Physical layer driver for the MIPI D-PHY differential lanes. Supports HS and LP modes.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 1. Display hardware blocks*...continued*

Block	Description
PSRAM / SRAM	Frame buffer storage. The i.MX RT700 integrates up to 7.5 MB of on-chip SRAM; external PSRAM can also be used.
GPIO	Used to control panel reset, backlight enable, and power enable signals.
I2C	Some panels (for example, Waveshare, Raspberry Pi) use an I2C controller on the panel PCB for power-on sequencing and register access.

3.3 Software architecture

The MCUXpresso SDK organizes the display software into the following layers:

Table 2. Software architecture

Layer	Description
Application	User application code. Calls `DEMO_LCDIF_Init()` and `DEMO_LCDIF_RGB()` to start the display.
Board support (lcdif_support.c/h)	Board-level initialization: clock setup, DSI host config, panel selection, and frame buffer addresses.
Panel driver (fsl_<panel>.c/h)	Panel-specific driver. Implements the `display_operations_t` interface: `init`, `deinit`, `start`, stop .
SDK drivers (fsl_lcdif.c, fsl_mipi_dsi.c)	Low-level peripheral drivers for LCDIF and MIPI DSI host.
SDK video middleware	Common display abstractions: `display_handle_t`, `display_config_t`, `display_operations_t`.

4 Information required from the display datasheet

This section describes the information required from the display data sheet.

4.1 Display timing parameters

Before writing any code, extract the following timing parameters from the display panel data sheet. These values define the synchronization intervals and blanking periods required by the panel.

Table 3. Display timing parameters

Parameter	Description
Active Width	Horizontal resolution in pixels (for example, 1024).
Active Height	Vertical resolution in lines (for example, 600).
Horizontal Sync Width (HSW)	Duration of the HSYNC pulse in pixels.
Horizontal Front Porch (HFP)	Blank pixels after active area before HSYNC.
Horizontal Back Porch (HBP)	Blank pixels after HSYNC before active area.
Vertical Sync Width (VSW)	Duration of the VSYNC pulse in lines.
Vertical Front Porch (VFP)	Blank lines after active area before VSYNC.
Vertical Back Porch (VBP)	Blank lines after VSYNC before active area.
HTOTAL	Active Width + HSW + HFP + HBP

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 3. Display timing parameters*...continued*

Parameter	Description
VTOTAL	Active Height + VSW + VFP + VBP

4.2 MIPI DSI requirements

From the data sheet, identify:

- Number of MIPI DSI data lanes (1 or 2 for most panels; up to 4 for high-resolution panels).
- Maximum supported DPHY bit clock rate (for example, 850 Mbit/s).
- Video mode: Burst mode or Non-Burst with sync pulses.
- Whether the panel requires End of Transmission Packet (EoTp) insertion.
- Whether the panel requires a continuous HS clock or a gated clock.

4.3 Pixel format and color depth

Identify the pixel format supported by the panel. Table 4 lists the common options.

Table 4. Pixel format and color depth

Format	Notes
RGB565	16 bpp, 2 bytes per pixel. Saves memory bandwidth; suitable for smaller displays.
RGB888	24 bpp, 3 bytes per pixel. Full color depth; used by most MIPI panels in video mode.
RGB666	18 bpp. Less common; some panels support it as a power-saving option.

The i.MX RT700 LCDIF DC8000 outputs 24-bit color (RGB888) to the MIPI DSI host, which then packs it into DSI pixel packets of the appropriate format.

4.4 Panel initialization requirements

Determine whether the panel requires any of the following initialization mechanisms:

- Display Command Set (DCS) commands sent over the MIPI DSI command channel.
- Vendor-specific (non-DCS) register writes over MIPI DSI Generic Write.
- I2C register writes to an on-panel controller (for example, Waveshare, Raspberry Pi 7" panels).
- GPIO-controlled reset sequence (assert reset, delay, deassert reset).
- Specific power-up sequencing with delays between steps.

5 High-level display initialization flow

This section describes the high-level process required to initialize a MIPI display and obtain a valid image output on the i.MX RT700-based platforms. The flow is intended to be generic and applicable to most MIPI-DSI displays, while highlighting optional paths commonly required by certain panels.

5.1 Overview

At a high level, displaying an image on a MIPI display involves:

1. Initializing the display controller.
   The first step is to initialize the on-chip display controller, which includes:
   - Enabling the clock sources required by the display subsystem.
   - Resetting the display controller to a known state.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

- Configuring basic parameters such as resolution, pixel format, and timing.

At this stage, the controller is prepared to manage frame buffers and generate output, but no data is sent to the display yet.

2. Configuring the MIPI interface and physical lanes.
   Next, the MIPI interface must be configured, which involves:
   - Initializing the MIPI DSI host.
   - Configuring the number of data lanes.
   - Setting lane speed, voltage levels, and timing parameters.
   - Enabling the MIPI PHY.

   This step ensures that the physical communication channel between the i.MX RT700 and the display panel is correctly established.
3. Providing frame buffer addresses to the display controller
   Once the display controller and MIPI interface are ready:
   - Frame buffers are allocated in memory.
   - The base address (or addresses, if double buffering is used) is provided to the display controller.
   - The controller is configured to automatically fetch pixel data from memory and transmit it over the MIPI lanes.

   From this point onward, the display controller autonomously streams image data to the display without CPU intervention.
4. Optionally sending direct MIPI commands for panel-specific initialization.
   Some MIPI displays require panel initialization commands to be sent directly over the MIPI interface, independently of the display controller frame streaming.
   Examples include:
   - Exiting the sleep mode.
   - Configuring internal panel registers.
   - Selecting display modes or orientations.

   In these cases:
   - Vendor-specific commands are written directly to the MIPI lanes.
   - This step may occur before or after frame buffer configuration, depending on the display requirements

   Note: *This step is optional and dependent on the specific display panel. Some panels require extensive commands sequences, while others handle initialization internally.*
5. Configuring the display and enabling image output
   Finally:
   - The display is taken out of reset (if applicable).
   - The display controller is enabled.
   - The panel is configured to turn on the display output.

   At this stage, the display begins to receive pixel data and renders the image provided through the frame buffer.
   Figure 1 shows the process.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

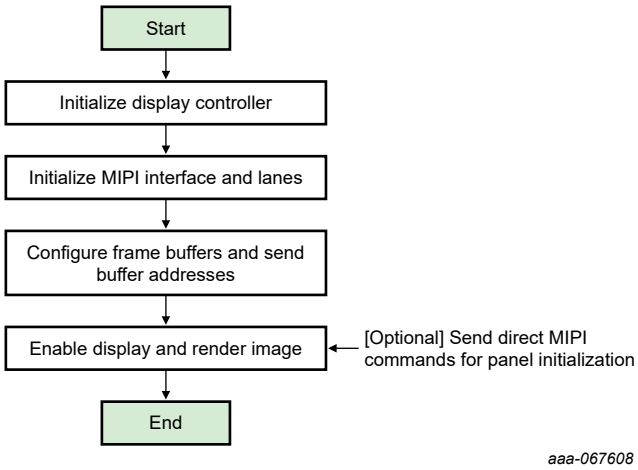


Figure 1. Initialization flowchart

6 Custom display integration workflow

This section describes the custom display integration workflow.

6.1 Overview

Integrating a custom MIPI display on the i.MX RT700 follows a well-defined sequence of steps. To obtain a valid image on the display, each step builds on the previous one and must be completed.

1. Configure panel timing parameters (HSW, HFP, HBP, VSW, VFP, VBP, resolution).
2. Configure display clocks: pixel clock, DPHY escape clock, DPHY HS bit clock.
3. Configure the MIPI DSI host and DPHY.
4. Create a custom panel driver implementing the display_operations_t interface.
5. Implement panel initialization (DCS commands, I2C, GPIO reset as required).
6. Allocate and configure frame buffers.
7. Enable display output: start LCDIF, enable panel, enter frame update loop.

6.2 Integration flowchart

The following describes the top-level initialization flow executed at startup:

```
BOARD_InitHardware()
  └─ BOARD_InitLcdifClock()          // Enable LCDIF power domains and pixel
 clock

DEMO_LCDIF_Init()
  ├─ LCDIF_Init()                    // Reset and enable LCDIF
  ├─ LCDIF_DpiModeSetConfig()        // Set resolution and DPI timing
  ├─ LCDIF_SetFrameBufferStride()    // Set bytes per line
  └─ BOARD_InitDisplayInterface()
       ├─ BOARD_InitMipiDsiClock()   // DPHY escape and HS bit clocks
       ├─ BOARD_SetMipiDsiConfig()   // DSI host, DPHY, DPI interface
       └─ BOARD_InitLcdPanel()       // Panel driver init (I2C, DSI commands,
 GPIO)

DEMO_LCDIF_RGB()
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
   ├─ Fill frame buffer
   ├─ LCDIF_SetFrameBufferAddr()
   ├─ LCDIF_SetFrameBufferConfig()
   ├─ LCDIF_Start()
   └─ Frame update loop (double buffering + VSYNC sync)
```

6.2.1 Step 1: Configure panel timings

The SDK already contains timing configurations for several supported display panels. These configurations can be found in the board display support files and panel-specific driver implementations.

6.2.1.1 Identify required timing parameters

Locate the display timing table in the panel data sheet. It is usually labeled **AC Characteristics**, **Display Timing**, or **Video Mode Timing**. Extract the values for HSW, HFP, HBP, VSW, VFP, VBP, active width, and active height.

These values define the horizontal and vertical synchronization intervals required by the display and are normally obtained directly from the display data sheet.

When integrating a new display, developers should start by locating an existing panel with similar resolution and refresh requirements and then replace the timing values with those specified by the new panel manufacturer.

6.2.1.2 Add timing configuration

Add a new panel definition block in *lcdif/lcdif_support.h*. Define a unique panel ID constant and add a corresponding timing block:

```
// lcdif/lcdif_support.h
#define MIPI_PANEL_MY_CUSTOM   6   /* Custom panel: 800 x 480 */

#elif (USE_MIPI_PANEL == MIPI_PANEL_MY_CUSTOM)
#define DEMO_PANEL_HEIGHT  600
#define DEMO_PANEL_WIDTH   1024
#define DEMO_HSW           20
#define DEMO_HFP           140
#define DEMO_HBP           160
#define DEMO_VSW           3
#define DEMO_VFP           12
#define DEMO_VBP           23
```

Set `USE_MIPI_PANEL` to the new constant (either in *lcdif_support.h* or as a CMake/compiler define):

```
#define USE_MIPI_PANEL  MIPI_PANEL_MY_CUSTOM
```

For the Waveshare display used in this application note, the timing values were derived from the display specifications and added as a new panel configuration.

6.2.1.3 Validate timing values

Verify that the total horizontal and vertical counts are consistent with the expected pixel clock and frame rate:

HTOTAL = Active_Width + HSW + HFP + HBP

VTOTAL = Active_Height + VSW + VFP + VBP

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Pixel_Clock = HTOTAL * VTOTAL * Frame_Rate

Example (Waveshare 1024x600 @ ~51 fps):

HTOTAL = 1024 + 20 + 140 + 160 = 1344

VTOTAL = 600 + 3 + 12 + 23 = 638

Pixel_Clock = 1344 * 638 * 51 ~ 43.7 MHz (configured as 44 MHz)

Before the LCDIF can generate valid video timing, the display pixel clock must be configured.

The pixel clock frequency is directly determined by the display resolution, synchronization timings, and desired refresh rate.

6.2.2 Step 2: Configure display clocks

This step describes how to configure display clocks.

6.2.2.1 Pixel clock fundamentals

The LCDIF pixel clock must be set to match the display timing requirements. The pixel clock is sourced from the main PLL PFD2 (nominally 528 MHz) and divided down using `kCLOCK_DivLcdifClk`.

6.2.2.2 LCDIF clock configuration

The LCDIF clock is configured in `BOARD_InitLcdifClock()` located in *lcdif/lcdif_support.c*. In this design, the LCDIF module uses the 528 MHz `MAIN_PLL_PFD2` clock source. To generate the pixel clock required by the Waveshare display, set the LCDIF clock divider to **12**, resulting in an approximately 44 MHz output clock (528 MHz/12 = 44 MHz). Adjust this divider as needed to match the pixel clock requirements of the target display panel.

```
void BOARD_InitLcdifClock(void)
{
    POWER_DisablePD(kPDRUNCFG_SHUT_MEDIA_MAINCLK);
    POWER_DisablePD(kPDRUNCFG_APD_LCDIF);
    POWER_DisablePD(kPDRUNCFG_PPD_LCDIF);
    POWER_ApplyPD();

    CLOCK_AttachClk(kMAIN_PLL_PFD2_to_LCDIF);  // 528 MHz source
    CLOCK_SetClkDiv(kCLOCK_DivLcdifClk, 12);    // 528/12 = 44 MHz (Waveshare)

    mipiDsiDpiClkFreq_Hz = CLOCK_GetLcdifClkFreq();
    CLOCK_EnableClock(kCLOCK_Lcdif);
    RESET_ClearPeripheralReset(kLCDIF_RST_SHIFT_RSTn);
}
```

Table 5 lists the divider values for supported panels. The divider value determines the pixel clock generated from the video PLL and therefore has a direct impact on the display refresh rate. The following table shows example divider settings for display panels commonly used with NXP boards, along with their resulting pixel clocks and approximate frame rates.

Table 5. Divider values for supported panels

Panel	Divider	Pixel clock	Approx. frame rate
MIPI_PANEL_RK055AHD091/RK055MHD091	14	528/14 = 37.7 MHz	~36 fps
MIPI_PANEL_RASPI_7INCH	27	528/27 = 19.6 MHz	~42 fps

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 5. Divider values for supported panels*...continued*

Panel	Divider	Pixel clock	Approx. frame rate
MIPI_PANEL_WAVESHARE (1024x600)	12	528/12 = 44 MHz	~51 fps
MIPI_PANEL_RK055IQH091	15	528/15 = 35.2 MHz	~60 fps

6.2.2.3 Frame rate calculation

To verify that the selected pixel clock produces the expected display refresh rate, the frame rate can be calculated from the total horizontal and vertical timing parameters:

Frame_Rate = Pixel_Clock/(HTOTAL x VTOTAL) = 44,000,000/(1344 x 638) ~ 51.3 fps

Where:

- Pixel_Clock = 44 MHz
- HTOTAL = total horizontal pixels per line, including blanking intervals
- VTOTAL = total vertical lines per frame, including blanking intervals

The calculated frame rate is approximately 51.3 fps, which matches the expected refresh rate for the Waveshare display configuration used in this application note.

6.2.2.4 Clock validation

At runtime, the pixel clock frequency is read back via `CLOCK_GetLcdifClkFreq()` and stored in `mipiDsiDpiClkFreq_Hz`. This value is later used to compute the minimum required DPHY bit clock. To verify that the LCDIF clock has been configured as expected, add a debug PRINTF statement after the LCDIF clock initialization is completed, for example, in `BOARD_InitLcdifClock()` immediately after the call to `CLOCK_GetLcdifFreq()`. This allows the actual clock frequency generated at runtime to be compared against the intended configuration.

```
PRINTF("DSI: %d", mipiDsiDpiClkFreq_Hz);
```

For the Waveshare panel configuration described in this application note, the printed value must be approximately 44 MHz, confirming that the 528 MHz clock source and divider value of 12 are producing the expected LCDIF pixel clock.

6.2.3 Step 3: Configure MIPI DSI host and DPHY

This step describes how to configure MIPI DSI host and DPHY.

6.2.3.1 DSI host overview

The MIPI DSI host on the i.MX RT700 is initialized through three sub-steps performed by `BOARD_InitDisplayInterface()` in `lcdif_support.c`:

```
status_t BOARD_InitDisplayInterface(void)
{
    BOARD_InitMipiDsiClock();    // 1. DPHY clocks
    BOARD_SetMipiDsiConfig();    // 2. DSI host + DPHY + DPI interface
    return BOARD_InitLcdPanel(); // 3. Panel initialization
}
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

6.2.3.2 DPHY bit clock calculation

The minimum DPHY HS bit clock is derived from the pixel clock, bits per pixel, and lane count:

Min_DPHY_Bit_Clock = (Pixel_Clock x Bits_Per_Pixel)/Num_Lanes = (44 MHz x 24)/2 = 528 Mbps

The minimum required DPHY HS bit clock calculated in the previous section is 528 Mbps. In practice, the DPHY clock must be configured above this minimum value to provide timing margin. The SDK therefore applies a 12.5 % overhead margin (`DEMO_MIPI_DPHY_BIT_CLK_ENLARGE`), equivalent to multiplying the minimum value by 9/8:

```
Target = 528 Mbps x 9/8 = 594 Mbps
```

The DPHY bit clock is generated from the Audio PLL PFD2 output. In the SDK configuration, the Audio PLL operates at 532.48 MHz, while the PFD multiplier is fixed at 18. The PFD divider must be programmed with an integer value, so the required divider is calculated as:

```
div = (532.48 MHz x 18)/594 MHz = 16.13
```

Since only integer divider values are supported, the result is truncated to 16.

The resulting DPHY HS bit clock is therefore:

```
Actual DPHY Bit Clock = 532.48 x 18/16 = 599.04 Mbps
```

6.2.3.3 Escape clock configuration

The DPHY escape clock is sourced from Main PLL PFD1 (396 MHz):

```
CLOCK_AttachClk(kMAIN_PLL_PFD1_to_MIPI_DPHYESC_CLK);
CLOCK_SetClkDiv(kCLOCK_DivDphyEscRxClk, 6);  // RxClkEsc = 396/6 = 66 MHz
 (spec: >= 60 MHz)
CLOCK_SetClkDiv(kCLOCK_DivDphyEscTxClk, 4);  // TxClkEsc = 66/4  = 16.5 MHz
 (spec: 12-20 MHz)
```

6.2.3.4 Video mode selection

The Waveshare 7" and Raspberry Pi 7" panels require non-burst with sync pulses mode. Most other panels use the burst mode for higher bandwidth efficiency. This conditional must be added in `BOARD_SetMipiDsiConfig()` for the definition of `.videoMode` attribute of the `dpiConfig`:

```
#if ((USE_MIPI_PANEL == MIPI_PANEL_MY_CUSTOM))
    .videoMode = kDSI_DpiNonBurstWithSyncPulse,
#else
    .videoMode = kDSI_DpiBurst,
#endif
```

6.2.3.5 DSI configuration example

The `BOARD_SetMipiDsiConfig()` function configures the DSI host, DPHY, and DPI interface for the Waveshare 7" panel:

```
static void BOARD_SetMipiDsiConfig(void)
{
    dsi_config_t      dsiConfig;
    dsi_dphy_config_t dphyConfig;

    const dsi_dpi_config_t dpiConfig = {
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

```
        .pixelPayloadSize = DEMO_PANEL_WIDTH,          // 1024
        .dpiColorCoding   = kDSI_Dpi24Bit,
        .pixelPacket      = kDSI_PixelPacket24Bit,
        .videoMode        = kDSI_DpiNonBurstWithSyncPulse,
        .bllpMode         = kDSI_DpiBllpLowPower,
        .polarityFlags    = kDSI_DpiVsyncActiveLow | kDSI_DpiHsyncActiveLow,
        .hfp = DEMO_HFP, .hbp = DEMO_HBP, .hsw = DEMO_HSW,
        .vfp = DEMO_VFP, .vbp = DEMO_VBP,
        .panelHeight      = DEMO_PANEL_HEIGHT,         // 600
        .virtualChannel   = 0,
    };

    DSI_GetDefaultConfig(&dsiConfig);
    dsiConfig.numLanes       = DEMO_MIPI_DSI_LANE_NUM;   // 2
    dsiConfig.autoInsertEoTp = true;

    DSI_GetDphyDefaultConfig(&dphyConfig,
        mipiDsiDphyBitClkFreq_Hz,    // 599.04 Mbps
        mipiDsiTxEscClkFreq_Hz);     // 16.5 MHz

    DSI_Init(DEMO_MIPI_DSI, &dsiConfig);
    DSI_InitDphy(DEMO_MIPI_DSI, &dphyConfig, 0);
    DSI_SetDpiConfig(DEMO_MIPI_DSI, &dpiConfig,
        DEMO_MIPI_DSI_LANE_NUM,
        mipiDsiDpiClkFreq_Hz,
        mipiDsiDphyBitClkFreq_Hz);
}
```

The key configuration decisions made in this function are:

- Number of data lanes: Set to **DEMO_MIPI_DSI_LANE_NUM** (two lanes for the Waveshare panel).
- EoTp insertion: Enabled automatically for all panels except the Raspberry Pi variant, which requires it to be disabled.
- Video mode: The Waveshare panel uses the `kDSI_DpiNonBurstWithSyncPulse` mode, where the DSI host replicates the HSYNC and VSYNC pulses precisely. Other panels use burst mode for higher bandwidth efficiency.
- BLLP mode: Blank Line Low-Power mode (`kDSI_DpiBllpLowPower`) is used during blanking periods to reduce power consumption by allowing the clock and data lanes to transition to LP state.
- DPHY timing parameters: `DSI_GetDphyDefaultConfig()` automatically derives all HS-to-LP and LP-to-HS transition timings, HS prepare, HS zero, HS trail, and clock lane timings from `mipiDsiDphyBitClkFreq_Hz` (599.04 Mbit/s) and `mipiDsiTxEscClkFreq_Hz` (16.5 MHz), ensuring compliance with the MIPI DPHY specification.

For the DBI (command-mode) path, the DPI interface configuration is skipped and the pixel format is instead set directly on the DSI command interface via `DSI_SetDbiPixelFormat()`.

6.2.3.6 Summary of all configured clocks

Table 6 summarizes all configured clocks.

Table 6. Summary of all configured clocks

Clock domain	Source	Divider(s)	Result
LCDIF pixel clock	Main PLL PFD2 (528 MHz)	/12	44 MHz
DPHY RxClkEsc	Main PLL PFD1 (396 MHz)	/6	66 MHz
DPHY TxClkEsc	Main PLL PFD1 (396 MHz)	/6/4	16.5 MHz

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 6. Summary of all configured clocks*...continued*

Clock domain	Source	Divider(s)	Result
DPHY HS bit clock	Audio PLL PFD2 (frac=16)	/1	599.04 Mbit/s

6.2.4 Step 4: Create a custom panel driver

This step describes how to create a custom panel driver.

6.2.4.1 Driver structure

A panel driver consists of two files: *fsl_<panel>.h and fsl_<panel>.c*. The driver must implement four functions matching the `display_operations_t` interface:

```
// fsl_mypanel.h
#ifndef _FSL_MYPANEL_H_
#define _FSL_MYPANEL_H_

#include "fsl_display.h"
#include "fsl_mipi_dsi.h"

typedef struct {
    mipi_dsi_device_t *dsiDevice;
    status_t (*writeRegister)(uint8_t reg, uint8_t val);  // I2C write (if
 needed)
    status_t (*readStatus)(uint8_t reg, uint8_t *val);    // I2C read  (if
 needed)
} mypanel_resource_t;

extern const display_operations_t mypanel_ops;

status_t MYPANEL_Init(display_handle_t *handle, const display_config_t *config);
status_t MYPANEL_Deinit(display_handle_t *handle);
status_t MYPANEL_Start(display_handle_t *handle);
status_t MYPANEL_Stop(display_handle_t *handle);

#endif
```

6.2.4.2 Driver registration

Register the driver by defining the `display_operations_t` structure and adding the panel to the selection logic in *lcdif_support.c*:

```
// fsl_mypanel.c
const display_operations_t mypanel_ops = {
    .init   = MYPANEL_Init,
    .deinit = MYPANEL_Deinit,
    .start  = MYPANEL_Start,
    .stop   = MYPANEL_Stop,
};
// lcdif_support.c - add to BOARD_InitLcdPanel():
#elif (USE_MIPI_PANEL == MIPI_PANEL_MY_CUSTOM)
    status = MYPANEL_Init(&mypanelHandle, &displayConfig);
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

6.2.4.3 Board integration

Add the panel driver source files to the build system. In the MCUXpresso SDK CMake project, add the files to *CMakeLists.txt*. In this example, the driver files are stored in a folder named `waveshare_panel` located under the project directory. If you place the driver files in a different location or use a different folder name, update the paths accordingly.

```
target_sources(${MCUX_SDK_PROJECT_NAME} PRIVATE
    ${CMAKE_CURRENT_LIST_DIR}/waveshare_panel/fsl_mypanel.c
)
target_include_directories(${MCUX_SDK_PROJECT_NAME} PRIVATE
    ${CMAKE_CURRENT_LIST_DIR}/waveshare_panel
)
```

6.2.4.4 Register the display driver

After creating the display driver, register the driver handle and resource structures in *lcdif_support.c*, so that the LCDIF framework can instantiate and control the panel.

```
#elif (USE_MIPI_PANEL == MIPI_PANEL_MY_CUSTOM)
static mipi_dsi_device_t dsiDevice = {
    .virtualChannel = 0,
    .xferFunc       = PANEL_DSI_Transfer,
};
static const mycustom_resource_t mycustomResource = {
    .dsiDevice    = &dsiDevice,
    .readStatus   = &BOARD_ReadPanelStatus,
    .writeRegister = &BOARD_WritePanelRegister,
};
static display_handle_t mycustomHandle = {
    .resource = &mycustomResource,
    .ops      = &mycustom_ops,
};
```

The `mipi_dsi_device_t` instance provides the DSI transport layer for panel communication. The resource structure supplies the panel-specific callbacks, while the display handle binds the resources and operations tables used by the display framework.

The custom panel driver header must also be included in *lcdif_support.c*, so that the display framework can access the driver definitions.

```
#elif (USE_MIPI_PANEL == MIPI_PANEL_MY_CUSTOM)
#include "fsl_mypanel.h"
```

Replace `MIPI_PANEL_MY_CUSTOM` and `fsl_mypanel.h` with the panel-specific identifiers used by the target display driver.

6.2.4.5 Board hardware requirements

Some MIPI DSI displays use the Raspberry Pi display connector or MIPI connector available on the i.MX RT700 EVK. When integrating these displays, additional board rework may be required to route the MIPI DSI signals and power rails to the connector.

Consult the i.MX RT700 EVK hardware documentation and board schematics for the required modifications before attempting to connect the display.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

6.2.5 Step 5: Implement panel initialization

This step describes how to implement panel initialization.

6.2.5.1 DCS commands

Standard MIPI Display Command Set (DCS) commands are sent using the SDK helper functions provided in *fsl_mipi_dsi_cmd.h*. Common DCS commands include:

Table 7. DCS commands

Command	Description
0x11	Sleep Out - exits the sleep mode. Typically it requires a 150 ms delay after sending.
0x29	Display On - enables the display output. Typically it requires a 20 ms delay.
0x28	Display Off.
0x10	Enter the sleep mode.
0x3A	Set the pixel format (COLMOD).
0x36	MADCTL - Memory Access Control (orientation, flip).
0x35	Tearing Effect Line On/Off.

6.2.5.2 Vendor-specific commands

The Waveshare display module requires a vendor-specific initialization sequence before normal operation. Unlike MIPI DCS commands, these settings are programmed through the board-specific `writeRegister()` callback, which writes values to the display control registers exposed by the module.

The initialization sequence is implemented as a table of register/value pairs:

```
static const wsdisp_cmd_t s_wsdispCmds[] = {
    {0xAD, 0x00},
    {0xC0, 0x01},
    {0xC2, 0x01},
    {0xAC, 0x01},
    {0xAD, 0x01},
    {0xAB, 0xC0},
    {0xAA, 0x01},
};
```

During initialization, the driver iterates through the command table and programs each register using the board-specific register write function:

```
for (uint8_t i = 0; i < ARRAY_SIZE(s_wsdispCmds); i++)
{
    status = resource->writeRegister(
        s_wsdispCmds[i].reg,
        s_wsdispCmds[i].val);
    if (status != kStatus_Success)
    {
        return kStatus_Fail;
    }
    WSDISP_DelayMs(100);
}
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

The sequence enables the display power circuitry, configures the display operating mode, sets the backlight brightness level, and finally enables the display output. Additional delays are inserted before and after the sequence to allow the display hardware to stabilize before video transmission begins.

After the vendor-specific initialization is complete, standard MIPI DCS commands can be used to control the panel state, such as enabling or disabling the display through `MIPI_DSI_DCS_SetDisplayOn()`.

6.2.5.3 GPIO control

For panels with GPIO-controlled reset and power lines, the init sequence follows this pattern:

```
// Assert reset
GPIO_PinWrite(BOARD_MIPI_RST_GPIO, BOARD_MIPI_RST_PIN, 0);
SDK_DelayAtLeastUs(10000, SystemCoreClock);   // 10 ms

// Deassert reset
GPIO_PinWrite(BOARD_MIPI_RST_GPIO, BOARD_MIPI_RST_PIN, 1);
SDK_DelayAtLeastUs(120000, SystemCoreClock);  // 120 ms - wait for panel stable

// Send DCS init commands...

// Enable backlight and power after successful init
GPIO_PinWrite(BOARD_MIPI_BL_GPIO, BOARD_MIPI_BL_PIN, 1);
GPIO_PinWrite(BOARD_MIPI_POWER_GPIO, BOARD_MIPI_POWER_PIN, 1);
```

6.2.5.4 I2C-controlled panels

Some display panels integrate a secondary MCU or power management IC accessible via I2C. The Waveshare 7" panels use this approach. The I2C interface is initialized before the panel driver is called:

```
// In BOARD_InitLcdPanel():
BOARD_MIPIPanelTouch_I2C_Init();  // Initialize I2C bus for panel control

// I2C write helper (implemented in board support files):
static status_t BOARD_WritePanelRegister(uint8_t regAddr, uint8_t value)
{
    return BOARD_I2C_Send(BOARD_MIPI_PANEL_TOUCH_I2C_BASEADDR,
                WSDISP_CTRL_ADDR,  // I2C device address (0x45)
                regAddr, 1,
                &value, 1);
}
```

6.2.5.5 Power-up sequence

The recommended power-up sequence for displays that use an external controller accessible through I2C is:

1. Initialize the I2C interface used by the display controller.
2. Apply power to the display module and wait for the controller to boot.
3. Wait approximately 500 ms after power-up.
4. Send the vendor-specific initialization sequence by writing the required register/value pairs over I2C.
5. Insert the required delay between consecutive register writes (100 ms in the reference implementation).
6. Wait approximately 500 ms after the final configuration command to allow the display to stabilize.
7. Enable video output and start the MIPI DSI video stream.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

8. Turn on the display using the standard MIPI DCS command:

```
MIPI_DSI_DCS_SetDisplayOn(dsiDevice, true);
```

The reference driver implements Step 3 to Step 6 inside `WSDISP_Init()`, where the initialization table is sent through the `writeRegister()` callback before the display is enabled. This sequence ensures that the display controller is fully configured before video data is transmitted over the MIPI DSI interface.

6.2.6 Step 6: Configure frame buffers

This step describes how to configure frame buffers.

6.2.6.1 Memory requirements

Frame buffer size depends on resolution and pixel format:

`Frame_Buffer_Size = Width x Height x Bytes_Per_Pixel`

Waveshare 1024x600 RGB565: = 1024 x 600 x 2 = 1,228,800 bytes = 0x12C000 bytes (~1.2 MB per buffer)

6.2.6.2 Single buffering

Single buffering uses one frame buffer. The CPU writes pixel data directly to the buffer that the LCDIF is currently reading. This may cause visible tearing if the frame is updated while the display is being refreshed, but reduces memory usage.

6.2.6.3 Double buffering

Double buffering uses two frame buffers: one is displayed by the LCDIF while the CPU renders the next frame into the other. The buffers are swapped at VSYNC to eliminate tearing.

```
// lcdif_support.h (Waveshare 1024x600)
#define DEMO_FB0_ADDR  0x60000000U  // Buffer 0 base
#define DEMO_FB1_ADDR  0x60200000U  // Buffer 1 base (0x12C000 apart)

// lcdif_rgb565.c
static uint32_t s_frameBufferAddr[2] = {DEMO_FB0_ADDR, DEMO_FB1_ADDR};
```

6.2.6.4 LCDIF frame buffer configuration

lcdif_fb_config_t fbConfig;

```
LCDIF_FrameBufferGetDefaultConfig(&fbConfig);

fbConfig.enable         = true;
fbConfig.inOrder        = kLCDIF_PixelInputOrderARGB;
fbConfig.rotateFlipMode = kLCDIF_Rotate0;
fbConfig.format         = kLCDIF_PixelFormatRGB565;
fbConfig.alpha.enable   = false;
fbConfig.width          = DEMO_IMG_WIDTH;
fbConfig.height         = DEMO_IMG_HEIGHT;

LCDIF_SetFrameBufferAddr(DEMO_LCDIF, 0, s_frameBufferAddr[0]);
LCDIF_SetFrameBufferConfig(DEMO_LCDIF, 0, &fbConfig);
```


All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Note: *For the DC8000 LCDIF variant (i.MX RT700), the image stride must be 64-byte aligned. The macro* `DEMO_BUFFER_STRIDE_BYTE` *in lcdif_support.h performs this alignment:*

```
#define DEMO_BUFFER_STRIDE_BYTE  \
  (((((DEMO_IMG_WIDTH * DEMO_BYTE_PER_PIXEL) - 1U) / 64U) + 1U) * 64U)
```

6.2.7 Step 7: Enable display output

This step describes how to enable the display output.

6.2.7.1 Start LCDIF

```
LCDIF_Start(DEMO_LCDIF); // Begin pixel fetching and DPI timing generation
```

6.2.7.2 Enable panel

After `LCDIF_Start()`, the panel backlight and power GPIOs are asserted to make the rendered image visible:

```
GPIO_PinWrite(BOARD_MIPI_BL_GPIO, BOARD_MIPI_BL_PIN, 1);      // Backlight on
GPIO_PinWrite(BOARD_MIPI_POWER_GPIO, BOARD_MIPI_POWER_PIN, 1); // Power on
```

6.2.7.3 Verify video output

After `LCDIF_Start()`, the LCDIF begins outputting video timing. The display shows the content of the initial frame buffer. If the screen remains black, refer to Section 7 for debugging steps.

6.2.7.4 Frame update mechanism

The frame update loop toggles between two frame buffers, synchronized to the VSYNC interrupt:

```
while (1)
{
    frameBufferIndex ^= 1U;  // Toggle to next buffer

    // Wait for frame done interrupt (set in LCDIF IRQ handler)
    while (!s_frameDone) {}

    LCDIF_SetUpdateReady(DEMO_LCDIF);      // Notify LCDIF new config is ready
    s_frameDone = false;

    DEMO_FillFrameBuffer(s_frameBufferAddr[frameBufferIndex]);  // Render next
 frame
    LCDIF_SetFrameBufferAddr(DEMO_LCDIF, 0,
 s_frameBufferAddr[frameBufferIndex]);
    // New address is latched at next VSYNC
}
```

7 Debugging and validation

This section describes how to debug and validate.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

7.1 Clock verification

Use the SDK debug console to print clock frequencies after initialization and before `DEMO_LCDIF_RGB()`:

```
PRINTF("Pixel clock:    %u Hz\r\n", CLOCK_GetLcdifClkFreq());
PRINTF("DPHY bit clock: %u Hz\r\n", CLOCK_GetMipiDphyClkFreq());
PRINTF("TxClkEsc:       %u Hz\r\n", CLOCK_GetMipiDphyEscTxClkFreq());
```

7.2 DSI verification

Confirm that `BOARD_InitDisplayInterface()` returns `kStatus_Success`.

Check that the DSI host status registers show no BTA timeout or PHY errors.

Verify that `mipiDsiDphyBitClkFreq_Hz` is within the supported bit rate range of the panel.

Ensure that EoTp is enabled/disabled as required by the panel specification.

7.3 Panel verification

Confirm that `BOARD_InitLcdPanel()` (or the panel Init function) returns `kStatus_Success`.

For I2C panels: verify that the I2C bus is initialized before the panel driver is called.

Check that the I2C device address matches the panel hardware address.

Add PRINTF statements inside the init command loop to identify which command fails.

7.4 Frame buffer verification

Confirm that frame buffer addresses are in valid, non-cached, and accessible memory regions.

Verify that `DEMO_FillFrameBuffer()` writes non-zero pixel data (add a PRINTF to confirm).

Check that `LCDIF_SetFrameBufferAddr()` is called before `LCDIF_Start()`.

Ensure that the frame buffer stride is correctly computed and set with `LCDIF_SetFrameBufferStride()`.

7.5 Signal validation checklist

If an oscilloscope or logic analyzer is available, verify these signals:

Table 8. Signal validation checklist

Signal	Expected behavior
MIPI DSI D0+/D0-	Differential data lane 0. Show HS burst activity during active video lines.
MIPI DSI CLK+/CLK-	Differential clock lane. Toggle at DPHY_BIT_CLK/2 during HS periods.
HSYNC / VSYNC	Active-low sync pulses. Frequency must match the configured frame rate.
I2C SDA/SCL	I2C transactions during panel initialization. Verify the correct device address and register values.
BL_EN/POWER_EN	GPIO signals. Go high after successful panel initialization.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

8 Common integration issues

This section describes the common integration issues.

8.1 Black screen

Table 9 lists possible causes and solutions for black screen.

Table 9. Possible causes and solutions for black screen

Cause	Solution
Backlight not enabled	Verify that `GPIO_PinWrite` (`BOARD_MIPI_BL_GPIO`, ..., 1) is called after successful init.
Panel init failed	Check the return value of panel Init function. Enable the PRINTF in the init loop.
Frame buffer not filled	Call `DEMO_FillFrameBuffer()` before `LCDIF_Start()`.
Wrong frame buffer address	Verify that `DEMO_FB0_ADDR` points to accessible memory with correct alignment.
Clock not configured	Confirm that `BOARD_InitLcdifClock()` and `BOARD_InitMipiDsiClock()` are called before `LCDIF_Start()`.

8.2 Flickering image

Table 10 lists possible causes and solutions for flickering image.

Table 10. Possible causes and solutions for flickering image

Cause	Solution
Frame buffer updated while displayed	Enable double buffering. Update the inactive buffer only.
`LCDIF_SetUpdateReady()` not called	Call `LCDIF_SetUpdateReady()` before updating frame buffer address.
Pixel clock too slow	Recalculate the pixel clock. Increase the divider or adjust it to target ~60 fps.

8.3 Corrupted colors

Table 11 lists possible causes and solutions for corrupted colors.

Table 11. Possible causes and solutions for corrupted colors

Cause	Solution
Wrong pixel format	Ensure fbConfig.format and dpiConfig.dpiColorCoding match the panel requirements.
Incorrect byte order	Check fbConfig.inOrder (kLCDIF_PixelInputOrderARGB vs. RGBA, BGRA, etc.).
Frame buffer stride wrong	Verify DEMO_IMG_BYTES_PER_LINE calculation and alignment.

8.4 DSI communication failures

Table 12 lists possible causes and solutions for DSI communication failures.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Table 12. Possible causes and solutions for DSI communication failures

Cause	Solution
DPHY bit clock too low	Recalculate: ensure bit clock > (pixel_clk * bpp / lanes) with 12.5 % margin.
EoTp mismatch	Some panels require EoTp disabled. Set `dsiConfig.autoInsertEoTp` = false.
Wrong video mode	Try `kDSI_DpiNonBurstWithSyncPulse` if the burst mode does not work.
Lane count mismatch	Verify that `DEMO_MIPI_DSI_LANE_NUM` matches panel specification.

8.5 Timing-related problems

Table 13 lists possible causes and solutions for timing-related problems.

Table 13. Possible causes and solutions for timing-related problems

Cause	Solution
Incorrect HFP/HBP/HSW values	Double-check values against the data sheet. Wrong blanking causes sync loss.
Refresh rate out of spec	Recalculate pixel clock to achieve a target frame rate within panel tolerance.
PSRAM cache conflict	If the frame buffer is in PSRAM with cache enabled, disable cache (see *lcdif_support.c* comment).

9 Conclusion

This application note has presented a structured and practical approach for enabling and integrating custom MIPI displays on the i.MX RT700 platform. By walking through both the conceptual flow and the implementation steps, it has demonstrated how a display can be successfully brought up from initialization to image rendering.

Through this document, the following key points were covered:

- A clear understanding of the i.MX RT700 display subsystem, including the roles of the display controller, MIPI DSI host, MIPI PHY, and frame buffer memory.
- A high-level display initialization flow applicable to most MIPI-based panels, serving as a reusable reference for custom display integrations.
- The standard controller-driven data path, where frame buffers are automatically fetched and streamed to the display over the MIPI interface.
- The importance of panel-specific initialization and the use of optional direct MIPI commands required by certain displays.
- A practical implementation example using a Waveshare MIPI display, illustrating how the theoretical flow maps to a real hardware setup.
- Guidance on designing flexible display driver implementations that can be adapted to different MIPI panels.

Overall, this application note equips developers with the essential knowledge and practical guidance needed to develop, customize, and debug MIPI display drivers on the i.MX RT700-based designs. The concepts and flow described here can be directly reused or extended to support additional MIPI displays with minimal integration effort.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

10 References

For more information on the topics covered in this document, refer to the following documents:

- *i.MX RT700 Crossover Microcontroller Reference Manual* (document IMXRT700RM)

11 Note about the source code in the document

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

Copyright 2026 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.

12 Revision history

Table 14 summarizes the revisions to this document.

Table 14. Revision history

Document ID	Release date	Description
AN15175 v.1.0	28 October 2026	Initial public release.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

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.

Evaluation products — This evaluation product is intended solely for technically qualified professionals, specifically for use in research and development environments to facilitate evaluation purposes. It is not a finished product, nor is it intended to be a part of a finished product. Any software or software tools provided with an evaluation product are subject to the applicable license terms that accompany such software or software tools.

This evaluation product is provided on an "as is" and "with all faults" basis for evaluation purposes only and is not to be used for product qualification or production. If you choose to use these evaluation products, you do so at your risk and hereby agree to release, defend and indemnify NXP (and all of its affiliates) for any claims or damages resulting from your use. NXP, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this evaluation product remains with user.

In no event shall NXP, its affiliates or their suppliers be liable to user for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the evaluation product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that user might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP, its affiliates and their suppliers and user's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by user based on reasonable reliance up to the greater of the amount actually paid by user for the evaluation product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose and shall not apply in case of willful misconduct.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

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.

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 and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) 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.

eIQ — is a trademark of NXP B.V.

I2C-bus — logo is a trademark of NXP B.V.

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Tables

Tab. 1. Display hardware blocks 3
Tab. 2. Software architecture 4
Tab. 3. Display timing parameters 4
Tab. 4. Pixel format and color depth 5
Tab. 5. Divider values for supported panels 9
Tab. 6. Summary of all configured clocks 12
Tab. 7. DCS commands 15
Tab. 8. Signal validation checklist 19
Tab. 9. Possible causes and solutions for black screen 20
Tab. 10. Possible causes and solutions for flickering image 20
Tab. 11. Possible causes and solutions for corrupted colors 20
Tab. 12. Possible causes and solutions for DSI communication failures 21
Tab. 13. Possible causes and solutions for timing-related problems 21
Tab. 14. Revision history 22

Figures

Fig. 1. Initialization flowchart 7

All information provided in this document is subject to legal disclaimers.

© 2026 NXP B.V. All rights reserved.

Contents

1 Introduction 2
1.1 Target audience 2
2 Hardware and software requirements 2
2.1 Hardware 2
2.2 Software 3
2.3 Reference example 3
3 i.MX RT700 display architecture 3
3.1 Display data path 3
3.2 Display hardware blocks 3
3.3 Software architecture 4
4 Information required from the display datasheet 4
4.1 Display timing parameters 4
4.2 MIPI DSI requirements 5
4.3 Pixel format and color depth 5
4.4 Panel initialization requirements 5
5 High-level display initialization flow 5
5.1 Overview 5
6 Custom display integration workflow 7
6.1 Overview 7
6.2 Integration flowchart 7
6.2.1 Step 1: Configure panel timings 8
6.2.1.1 Identify required timing parameters 8
6.2.1.2 Add timing configuration 8
6.2.1.3 Validate timing values 8
6.2.2 Step 2: Configure display clocks 9
6.2.2.1 Pixel clock fundamentals 9
6.2.2.2 LCDIF clock configuration 9
6.2.2.3 Frame rate calculation 10
6.2.2.4 Clock validation 10
6.2.3 Step 3: Configure MIPI DSI host and DPHY 10
6.2.3.1 DSI host overview 10
6.2.3.2 DPHY bit clock calculation 11
6.2.3.3 Escape clock configuration 11
6.2.3.4 Video mode selection 11
6.2.3.5 DSI configuration example 11
6.2.3.6 Summary of all configured clocks 12
6.2.4 Step 4: Create a custom panel driver 13
6.2.4.1 Driver structure 13
6.2.4.2 Driver registration 13
6.2.4.3 Board integration 14
6.2.4.4 Register the display driver 14
6.2.4.5 Board hardware requirements 14
6.2.5 Step 5: Implement panel initialization 15
6.2.5.1 DCS commands 15
6.2.5.2 Vendor-specific commands 15
6.2.5.3 GPIO control 16
6.2.5.4 I2C-controlled panels 16
6.2.5.5 Power-up sequence 16
6.2.6 Step 6: Configure frame buffers 17
6.2.6.1 Memory requirements 17
6.2.6.2 Single buffering 17
6.2.6.3 Double buffering 17
6.2.6.4 LCDIF frame buffer configuration 17
6.2.7 Step 7: Enable display output 18
6.2.7.1 Start LCDIF 18
6.2.7.2 Enable panel 18
6.2.7.3 Verify video output 18
6.2.7.4 Frame update mechanism 18
7 Debugging and validation 18
7.1 Clock verification 19
7.2 DSI verification 19
7.3 Panel verification 19
7.4 Frame buffer verification 19
7.5 Signal validation checklist 19
8 Common integration issues 20
8.1 Black screen 20
8.2 Flickering image 20
8.3 Corrupted colors 20
8.4 DSI communication failures 20
8.5 Timing-related problems 21
9 Conclusion 21
10 References 22
11 Note about the source code in the document 22
12 Revision history 22
Legal information 23

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2026 NXP B.V. All rights reserved.