Target setup for NXP RT117X

This document provides a detailed guide on setting up and using the NXP RT117X hardware with CGI Studio.
It explains the steps for environment setup, flashing the hardware, building applications, and running tests.

- Hardware setup for NXP RT117X
- Software Setup for Flashing NXPRT117X
- Blend Mode Mapping
- Various Device Specific Content
    - RT117XBitmapConverter

Hardware setup for NXP RT117X

Required Tools and Equipment

The following tools and equipment are needed to set up and run the project:

- NXP RT117X hardware board with display
- CGI Studio (Minimum version 3.15.1)
- CMake GUI
- Visual Studio Code
- Ninja Build Tool
- Required binary files (.bin)
- IAR Embedded Workbench
- Board Support Packages
- Hardware Variants

Hardware Revision	Supported Display	Supported BSP versions	Notes
Rev_a	720 × 1280 only	Up to BSP 2.16	BSP versions higher than 2.16 are not supported
Rev_b	720 × 1280 or 800 × 480 (Raspberry Pi Display)	2.16 and newer (e.g., 25_06_00)	Supports newer BSP versions

Hardware Connection Setup

1. Connect the **power cable** to the NXP RT117X board.
2. Connect the **Micro-USB port** on the board to a **USB-A port** on your laptop or PC.
3. Make sure the Micro-USB is connected to the **USB Debug port**.
4. Power on the board.
5. Check if the **power LEDs** on the board are glowing.
6. Wait for your system to detect the board. You can verify this by checking **Device Manager → Ports (COM & LPT)** — a new **COM port** should appear for the NXP RT117X board

Verification after completing the flashing

1. Confirm that the display output matches the expected layout from Scene Composer.

2. Check that images, colors, and animations render correctly.
3. Validate Bitmap Stride behavior for both aligned and unaligned images.
4. Ensure there are no visual distortions or flickering.

Software Setup for Flashing NXPRT117X

Scene Composer Configuration

1. Open Scene Composer and create a new solution using the Empty Sample Solution template.
2. Select and import multiple images into Scene Composer from the following path: (CGI-ROOT/cgi_studio_content/2D/Paid_Stock_Images)
3. Drag and drop the imported images into the Scene Editor.
4. In the Scene Editor, adjust the images by scaling and rotating them as needed.
5. Add the following elements to the Scene Editor:
    1. One Solid Color Node
    2. One Text Node
    3. One Button
6. Open the Render Target tab (next to the toolbox) and select an available render target.
7. In the Properties panel, set the following values:
    1. Orientation:
        1. 90 ° for 720x1280/Portrait mode displays.
        2. 0° for 800 x 480/ Landscape mode displays.
    2. Width: 720
    3. Height: 1280
8. Click on Display (0) under the Render Target tab.
9. In the Display properties, set the X and Y resolution to match the Render Target resolution.
10. Navigate to File → Solution Options → General Configuration.
11. Finally, generate the Target Asset.

VS Code Configuration

1. Open Visual Studio Code (VS Code) from the CGI root directory.
2. Navigate to the toolchain file as per the hardware revision (rev_a/rev_b) and BSP version used, refer the following table to select the necessary toolchain file:
   Toolchain Table:

Hardware Revision	Toolchain file	Description
Rev_a	IAR-toolchain-RT117X_Native2D.txt	Rev_a without Touch

IAR-toolchain-RT117X_Native2D_Touch.txt	Rev_a with touch enabled display	
Rev_b	IAR-toolchain-RT117X_Native2D_B.txt	Rev_b without Touch support
	IAR-toolchain-RT117X_Native2D_B_Touch.txt	Rev_b with Touch enabled display
	IAR-toolchain-RT117X_Native2D_B_Touch_2.16.txt	Rev_b with Touch enabled display and BSP version 2.16

For Eg: If using a Rev_b hardware version with BSP version 2.16, navigate to the following file.
(CGI-ROOT/cgi_studio_devices/src/RT117X/ToolchainFiles/IAR-toolchain-RT117X_Native2D_B_Touch_2.16.txt)

3. Open the file and make the following edits:
   1. Disable touch (only if the display does not support touch)
      set (CGIDEVICE_TOUCH_ENABLED OFF CACHE BOOL "Enable Touch support").
   2. Update the **BSP path** to match the local directory.
4. Next, open the following file and update the display configurations as required:
   (CGI-ROOT/cgi_studio_devices/src/RT117X/CourierPlatformDefs/Target_FreeRTOS_RT117X/KernelTargets/iar/bsp_cm7_B216_cgi/vglite/cm7/display_support.h)
5. In this file, go to line 28 and update the display panel configuration as shown below:
   1. **Before:**
      #define DEMO_PANEL DEMO_PANEL_RASPI_7INCH //DEMO_PANEL_RK055MHD091
   2. **After:**
      #define DEMO_PANEL DEMO_PANEL_RK055MHD091
      This removes the commented-out section and sets the display to **RK055MHD091**.

Project Configuration using Cmake GUI

1. Open **CMake GUI**.
2. Set the **source code folder** to:
   (CGI-ROOT\cmake\Apps\Courier\CourierSampleAppLight)
3. Set the **build folder** (the location where build files will be generated).
4. Click **Configure**.
5. When prompted, select **Ninja** as the generator.
6. Specify the **toolchain file** for cross-compiling refer the toolchain table, select the appropriate toolchain file as per the hardware version.
7. Click **Finish → Next**.
8. Click **Configure** again until all red entries disappear, then click **Generate**.
9. Close **CMake GUI**.
10. Open terminal in the **build folder** and give a **ninja** build.
11. Run the following command
    ninja

Adding Assets

1. Create an **Assets** folder in:
   CGI-ROOT\cgi_studio_courier_apps\src\CourierSampleAppLight
2. Add the generated asset file to this folder.
3. Rename it to:
   AssetLibCff_RT117X_Target.bin

Flashing and Execution

1. Open the generated project from the following path:
   BUILD-ROOT\iar_project\debug_cm7_B_cgi\CGI_RT117X.ew
2. **IAR Embedded Workbench IDE** will launch automatically.
3. Click **Download and Debug** to flash the binary to the hardware.
4. After flashing, click **GO** in IAR to start execution.
5. Observe the output on the display connected to the NXP RT117X board.

Blend Mode Mapping

Following are the mapping from OpenGLES blend mode to NXP-RT supported blend modes.

VG Image blend mode mapping (VGLite API 3.x, available only in BSP version 25.06):

1. OPENVG_BLEND_SRC_OVER : sF=SourceAlpha, dF=InverseSourceAlpha, sAF=One, dAF =InverseSourceAlpha
2. OPENVG_BLEND_DST_OVER : sF=InverseDestAlpha, dF=One, sAF=InverseDestAlpha, dAF =One
3. OPENVG_BLEND_SRC_IN : sF=DestAlpha, dF=Zero, sAF=DestAlpha, dAF=Zero
4. OPENVG_BLEND_DST_IN : sF=Zero, dF=SourceAlpha, sAF=Zero, dAF=SourceAlpha
5. OPENVG_BLEND_MULTIPLY : sF=DestColor, dF=Zero, sAF=DestAlpha, dAF=Zero
6. OPENVG_BLEND_SCREEN : sF=One, dF=InverseSourceColor, sAF=One, dAF =InverseSourceColor
7. OPENVG_BLEND_ADDITIVE : sF=One, dF=One, sAF=One, dAF=One

VG Image blend mode mapping (VGLite API 2.x, available only in BSP version 2.16):

1. VG_LITE_BLEND_NONE : sAF=One, dAF=Zero
2. VG_LITE_BLEND_DST_IN : sAF=Zero, dAF=SourceAlpha
3. VG_LITE_BLEND_SRC_OVER : sAF=One, dAF=InverseSourceAlpha
4. VG_LITE_BLEND_SRC_IN : sAF=DestAlpha, dAF=Zero
5. VG_LITE_BLEND_DST_OVER : sAF=InverseDestAlpha, dAF=One
6. VG_LITE_BLEND_MULTIPLY : sF=DestColor, dF=Zero, sAF=One, dAF=InverseSourceAlpha
7. VG_LITE_BLEND_SUBTRACT : sF=InverseSourceColor, dF=One, sAF=One, dAF=One
8. VG_LITE_BLEND_SCREEN : sF=One, dF=InverseSourceColor, sAF=One, dAF =InverseSourceColor
9. VG_LITE_BLEND_ADDITIVE : sF=One, dF=One, sAF=One, dAF=One

PXP Image blend mode mapping:

1. kPXP_PorterDuffSrc : sAF=One, dAF=Zero
2. kPXP_PorterDuffAtop : sAF=DestAlpha, dAF=InverseSourceAlpha
3. kPXP_PorterDuffOver : sAF=One, dAF=InverseSourceAlpha \n
4. kPXP_PorterDuffIn : sAF=DestAlpha, dAF=Zero \n
5. kPXP_PorterDuffOut : sAF=InverseDestAlpha, dAF=Zero \n
6. kPXP_PorterDuffDst : sAF=Zero, dAF=One \n

7. kPXP_PorterDuffDstAtop : sAF=InverseDestAlpha, dAF=SourceAlpha \n
8. kPXP_PorterDuffDstOver : sAF=InverseDestAlpha, dAF=One \n
9. kPXP_PorterDuffDstIn : sAF=Zero, dAF=SourceAlpha \n
10. kPXP_PorterDuffDstOut : sAF=Zero, dAF=InverseSourceAlpha \n
11. kPXP_PorterDuffXor : sAF=InverseDestAlpha, dAF=InverseSourceAlpha \n
12. kPXP_PorterDuffClear : sAF=Zero, dAF=Zero \n
13. kPXP_PorterDuffMax : sAF=One, dAF=One \n\

Various Device Specific Content

RT117XBitmapConverter

Description

This document provides a complete list of bitmap formats supported by the **RT117X target**.

These formats define how pixel color and alpha data are stored in memory. The RT117X extended bitmap formats support alpha-only, RGB, and ARGB image formats, enabling a balance between memory usage, performance, and visual quality.

These bitmap converters are supported only on the target. Windows simulation uses the default bitmap converter.

Alpha-Only Formats

Formats that store only transparency information, typically used for masks and blending.

Serial No.	Format	Description
1	RT117XExtendedBitmapFormatA8	8-bit alpha-only format.

RGB Color Formats

Formats that store color information without an alpha channel, optimized for memory efficiency.

Serial No.	Format	Description
1	RT117XExtendedBitmapFormatRGB565	16-bit RGB format (5-6-5).
2	RT117XExtendedBitmapFormatXRGB8888	32-bit color with unused alpha channel (RGB only).

ARGB Color Formats

Formats that store color information along with an alpha channel for transparency.

Serial No.	Format	Description
1	RT117XExtendedBitmapFormatARGB8888	32-bit color with 8 bits per channel including alpha.

2	RT117XExtendedBitmapFormatARGB1555	16-bit color with 1-bit alpha and 5 bits per RGB channel.
3	RT117XExtendedBitmapFormatARGB4444	16-bit color with 4 bits per channel including alpha.