

Camera Overlay on RaspberryPI5

**Provided by:**

**Matin Karimi**

**Contact info:**

[matinkarimi234@gmail.com](mailto:aliszedbazi2000@gmail.com)

**Farand Systems Company**

Contents

[ Introduction 2](#_Toc189748592)

[ Hardware Topology 2](#_Toc189748593)

[ Hardware Components 2](#_Toc189748594)

[ Firmware Configuration 2](#_Toc189748595)

[ Python Modules 2](#_Toc189748596)

[ Additional Settings 2](#_Toc189748597)

[ Conclusion 2](#_Toc189748598)

# Introduction

This report documents the journey of integrating a low-latency video stream with an overlay on a Raspberry Pi using the DRM (Direct Rendering Manager) framework. It outlines the initial challenges faced, the steps taken that led to failures, and the final successful implementation using Picamera2 in Python.

# Initial Approach: Using DRM and Libcamera

The first attempt involved using the legacy MMAL/DispmanX stack with raspivid for a low-latency camera preview while overlaying graphics via DispmanX. The idea was to leverage DRM and libcamera together to achieve hardware-accelerated rendering with minimal latency.

**Steps Attempted:**

1. **Configured the Raspberry Pi for the legacy camera stack**:
   * Added start\_x=1 and gpu\_mem=128 to /boot/config.txt.
   * Enabled the legacy camera driver.
2. **Used raspivid for real-time video capture**:
   * Achieved a low-latency preview.
3. **Attempted to overlay graphics using DRM and Libcamera**:
   * Created a DRM preview window.
   * Rendered an overlay using a separate plane.

**Problems Encountered:**

* DRM and libcamera did not function together seamlessly.
* Using DRM with a separate graphics overlay interfered with the libcamera video pipeline.
* The inability to properly integrate an overlay while maintaining low latency led to a dead-end.

# Transition to Picamera2

After encountering roadblocks with DRM and libcamera integration, an alternative approach using Picamera2 in Python was explored.

**Steps Taken with Picamera2:**

1. **Configured Picamera2 for optimized preview**:
   * Used Picamera2() with create\_preview\_configuration() to get the default optimized settings.
   * Set the preview mode to Preview.DRM for direct rendering, bypassing extra processing layers.
2. **Added an overlay with a moving cross**:
   * Used a NumPy array to create an RGBA overlay.
   * Dynamically updated the overlay position using keyboard input (curses module).
3. **Successfully achieved low-latency preview with an interactive overlay**:
   * The overlay was hardware-composited and displayed in real-time.
   * Performance was significantly improved compared to previous attempts.
4. **Extended functionality to include video recording**:
   * Introduced functionality to start and stop recording using keyboard input.
   * Initially, the recorded video did not include the overlay since DRM overlays are not part of the camera capture stream.
   * Explored DRM framebuffer capture solutions such as kmsgrab to record the preview with the overlay.

**Python Implementation**

Below is the Python code used to achieve the low-latency video stream with an interactive overlay:

#!/usr/bin/python3

import time

import numpy as np

import curses

from picamera2 import Picamera2, Preview

from picamera2.encoders import H264Encoder

def main(stdscr):

    # Initialize Picamera2 using the default (optimized) preview configuration.

    picam2 = Picamera2()

    config = picam2.create\_preview\_configuration()  # using the optimal configuration

    picam2.configure(config)

    encoder = H264Encoder(10000000)

    picam2.start\_preview(Preview.DRM)

    picam2.start()

    time.sleep(1)  # Allow the camera to settle

    # Get the preview resolution from the configuration.

    preview\_width, preview\_height = config["main"]["size"]

    # Create an RGBA overlay array matching the preview resolution.

    overlay = np.zeros((preview\_height, preview\_width, 4), dtype=np.uint8)

    # Cross parameters

    thickness = 5               # Thickness of cross lines in pixels

    cross\_color = (255, 255, 255, 128)  # White with 50% transparency

    # Initial cross position (centered)

    cross\_x = preview\_width // 2

    cross\_y = preview\_height // 2

    def draw\_cross(ov, cx, cy, thick, color):

        """Clear the overlay and draw a cross at (cx, cy)."""

        ov.fill(0)  # Clear overlay to transparent

        # Draw vertical line:

        x\_start = max(cx - thick // 2, 0)

        x\_end   = min(cx + thick // 2 + 1, preview\_width)

        ov[:, x\_start:x\_end] = color

        # Draw horizontal line:

        y\_start = max(cy - thick // 2, 0)

        y\_end   = min(cy + thick // 2 + 1, preview\_height)

        ov[y\_start:y\_end, :] = color

        return ov

    # Draw the initial cross and apply the overlay.

    draw\_cross(overlay, cross\_x, cross\_y, thickness, cross\_color)

    picam2.set\_overlay(overlay)

    # Initialize curses for non-blocking input and arrow key support.

    stdscr.nodelay(True)

    stdscr.keypad(True)

    stdscr.clear()

    stdscr.addstr(0, 0, "Arrow keys: move cross | r: start recording | s: stop recording | q: quit")

    stdscr.refresh()

    recording = False

    while True:

        key = stdscr.getch()

        if key == curses.KEY\_UP:

            cross\_y = max(cross\_y - 10, 0)

        elif key == curses.KEY\_DOWN:

            cross\_y = min(cross\_y + 10, preview\_height - 1)

        elif key == curses.KEY\_LEFT:

            cross\_x = max(cross\_x - 10, 0)

        elif key == curses.KEY\_RIGHT:

            cross\_x = min(cross\_x + 10, preview\_width - 1)

        elif key == ord('r'):

            if not recording:

                recording = True

                picam2.start\_recording(encoder,"video.h264")

                stdscr.addstr(1, 0, "Recording started                     ")

        elif key == ord('s'):

            if recording:

                recording = False

                picam2.stop\_recording()

                stdscr.addstr(1, 0, "Recording stopped, saved as video.h264")

        elif key == ord('q'):

            break

        # Update the overlay with the new cross position.

        draw\_cross(overlay, cross\_x, cross\_y, thickness, cross\_color)

        picam2.set\_overlay(overlay)

        time.sleep(0.05)  # Short delay for responsiveness

    # Clean up before exit.

    picam2.stop\_preview()

    picam2.stop()

if \_\_name\_\_ == '\_\_main\_\_':

    curses.wrapper(main)

# Raspberry Pi Configuration for Low-Latency Streaming

To optimize the Raspberry Pi for this setup, the following system configurations were applied:

1. **Disable the Desktop Environment (for lower latency)**:
2. sudo systemctl disable lightdm
3. sudo systemctl set-default multi-user.target

2. **Optimize /boot/config.txt**:

* Add or modify the following parameters:
* start\_x=1
* gpu\_mem=256
* disable\_camera\_led=1
* dtoverlay=vc4-kms-v3d

3. **Ensure Minimal Background Processes**:

* Use top or htop to monitor and stop unnecessary services.
* Disable unused system daemons:
* sudo systemctl disable bluetooth
* sudo systemctl disable avahi-daemon

# Key Lessons

DRM and libcamera cannot be easily integrated for overlays due to the way they manage buffers and rendering.

Picamera2 offers a more flexible and functional solution for real-time overlays in Python, making it easier to control overlays and recording.

DRM preview mode is optimal for low-latency applications, but additional steps (like capturing the full DRM output) are required to include overlays in recordings.

Framebuffer capture or screen recording techniques (e.g., ffmpeg -f kmsgrab) can be used to record overlays along with the video.

# Conclusion

* Through a series of failures and optimizations, the final working implementation involved using Picamera2 with DRM preview and a dynamic overlay controlled via keyboard inputs. The experience highlights the challenges of integrating DRM and libcamera directly and the benefits of alternative approaches such as Python-based Picamera2 for achieving a functional, low-latency streaming solution with an interactive overlay.