

DESIGN / RECORDED

Little radio. Thoughtful software.

One control loop, five buttons, and a world to explore.

The architecture behind Alfa's offline player.



ADR 0001: An offline Raspberry Pi player with explicit hardware boundaries
Linux application · Raspberry Pi · Python

Status: Recorded from the existing implementation

Scope: The Linux application in `app/`. Pico firmware is a separate future implementation.

Context

The radio uses a small SPI display, an I²S amplifier, and five analog buttons read through an I²C ADC. It must start without a network connection or desktop session. A child can hold or repeatedly press a button. The same application must support a hardware-free preview on a development computer.

Decision

Use a Python command-line application with a single control loop and an asynchronous audio worker. Keep device access behind explicit display, ADC, audio, and service interfaces. The public entry point is `python -m app`.

`app.app` loads `Settings`, selects the command, prepares media, acquires resources through `ExitStack`, and runs the control loop. `PlayerSession` owns navigation and button actions. It combines the current image, optional character animation, and letter drawing state.

Input and state

`ADS1x15` reads keypad voltage. Calibration maps that voltage to a button name. `Debouncer` requires a stable press and a release before another press. `PressGuard` limits repeated actions. SSH keyboard input maps to the same player actions.

The control loop passes accepted actions to `PlayerSession`. Left and right change the selected image. Up and down control volume or optional character animations. Center controls playback or starts the selected letter lesson. The standalone service continues running while idle and ignores normal interactive quit controls.

Media and display

`app.media` prepares images with Pillow and converts recordings with FFmpeg. Content hashes and manifests avoid repeated conversions. Prepared media supports startup without the original source images.

`LetterCatalog` supplies stroke geometry. `LetterLesson` and `CharacterAnimation` produce optional animated frames. `WireFrames` and `AnimationPresenter` reuse RGB565 frames and update changed display regions. The hardware display uses `ST7789`. Simulation selects `PreviewDisplay` and saves a PNG instead of accessing GPIO or SPI.

Audio

`StreamingAudioPlayer` sends short PCM blocks to ALSA through `aplay` in a worker thread. Software volume changes affect the current stream without restarting the recording. The default volume is 20%, with a configurable 50% maximum in the current configuration. Simulation produces no sound. Audio files remain user-supplied inputs.

Startup and shutdown

systemd supervises the app through `ServiceWatchdog` readiness and heartbeat notifications. An optional early boot splash releases the display through a local Unix socket before the player claims its GPIOs. Resource cleanup stops audio and releases the display and ADC when the process exits. The app does not shut down the Raspberry Pi after inactivity.

Consequences

- The player runs offline and requires no web server on the Pi.
- Hardware-free tests exercise the control loop, media handling, and player actions.
- Physical wiring, display appearance, and acoustic volume require checks on the assembled device.
- The Linux app depends on Raspberry Pi OS, ALSA, and Pi device libraries.
- The Pico wiring view documents a proposed pin assignment. It does not imply that the Linux app runs on a Pico.

Alternatives

A browser-based player on the Pi adds a browser runtime and display stack to a device with physical controls. A single synchronous audio call blocks control handling during playback. Separate Pico firmware can use the same peripherals, but needs its own media handling, timing, and PIO audio implementation.

Source map

Responsibility	Source
Command dispatch and control loop	<code>app/app.py</code>
Configuration	<code>app/config.py</code>
Player state and actions	<code>app/player.py</code>
ADC and display drivers	<code>app/hardware.py</code>
Calibration, debounce, and press guard	<code>app/buttons.py</code>
Media conversion and manifests	<code>app/media.py</code>
Audio worker	<code>app/audio.py</code>
Frame encoding and presentation	<code>app/frames.py</code>
Letter lessons and paths	<code>app/letter_lesson.py</code> , <code>app/letter_paths.py</code>
Service supervision and splash handoff	<code>app/service.py</code> , <code>app/splash.py</code> , <code>app/display_handoff.py</code>