Technical Module Description: TouchButtons.py

Version: 1.0 Date: January 29, 2025

Authors: Henk Stevens, Olaf Mastenbroek, Onno Janssen

Organization: Stichting Oradio

License: GNU GPL

# 1. Purpose and Overview

The `TouchButtons.py` module manages the physical touch-sensitive buttons on the Oradio device. It detects button presses with fast response times and distinguishes between short, long, and extra-long presses. This module directly interfaces with the Raspberry Pi GPIO pins and supports event-driven callbacks for efficient detection. It also provides a standalone test mode for development and diagnostics without requiring the full Oradio system.

# 2. Key Features

* Real-time GPIO-based touch button detection with debounce handling.
* Support for normal press, long press (6 seconds), and extra-long press (16 seconds).
* Immediate audible feedback using the 'Click' system sound for each button press.
* Integration with the Oradio state machine for seamless playback control and mode switching.
* Test mode support: blinks LEDs corresponding to button presses for standalone testing.
* Threaded long-press detection logic to avoid blocking the main program flow.

# 3. Press Detection and State Transitions

Button names and GPIO mappings are defined in the global `BUTTONS` dictionary. On a button press, the script determines the button's name and immediately plays the 'Click' sound as feedback. If integrated with the Oradio state machine, the module calls `state\_machine.transition()` to trigger the appropriate state based on the pressed button.

Long and extra-long presses are handled by a background thread (`\_detect\_long\_press`). For the 'Play' button, a long press initiates the Web Service mode (`StateWebService`), while an extra-long press forces Access Point mode (`StateWebServiceForceAP`). Other buttons currently log an error message if long or extra-long presses occur.

# 4. Standalone Use and Testing

When the script is executed directly (`python3 TouchButtons.py`), it runs in standalone test mode. In this mode, pressing any button will blink the corresponding LED briefly, without interacting with the Oradio state machine.

This test functionality is intended for hardware verification and development, ensuring correct GPIO operation and quick feedback during assembly or troubleshooting.

# 5. Detailed Explanation of Working Mechanism

## 5.1 GPIO Setup and Event Detection

The module configures each button’s GPIO pin as an input with an internal pull-up resistor. GPIO event detection is set up with `GPIO.add\_event\_detect()` on the falling edge (button pressed), using a debounce time of 10 milliseconds to avoid false triggers due to signal bouncing.

The `\_button\_callback()` function is called immediately upon a valid button press event. This function identifies which button was pressed using the `gpio\_to\_button` mapping, plays the 'Click' system sound, and initiates either LED feedback (in test mode) or state transitions (in operational mode). To prevent multiple triggering, a debounce function, is implemented, ignoring the button press for a BUTTON\_DEBOUNCE\_TIME. Typical this time is set around 300 ms. Which is a compromise between quick response time and not multiple triggers when touching the button.

## 5.2 Long Press and Extra-Long Press Detection

To detect long and extra-long presses without blocking the main thread, a separate thread is started upon each button press. This thread (`\_detect\_long\_press`) tracks the duration of the button press using `time.monotonic()`, checking whether the button remains pressed. If the button is held for longer than `LONG\_PRESS\_DURATION` (6 seconds), the `\_long\_press\_handler()` is triggered. If the button continues to be held until `EXTRA\_LONG\_PRESS\_DURATION` (16 seconds), the `\_extra\_long\_press\_handler()` is called.

These handlers determine the appropriate action based on the button pressed. For example, the 'Play' button triggers the Web Service start or forces Access Point mode, while other buttons log an error message since long presses are not currently supported for them.

## 5.3 Threading Model and Non-Blocking Design

All press handling, including sound playback and LED control, is performed in separate daemon threads. This allows the system to remain responsive and prevents the blocking of other operations, such as touch detection on other buttons or ongoing system processes.

By leveraging threading, the module achieves efficient detection of press durations while ensuring that system sounds and LED indicators operate asynchronously. The use of locks and thread safety mechanisms (e.g., the `button\_press\_times` dictionary) prevents race conditions in multi-threaded environments.