Pi Zero Antenna Controller Original Masterplan

Nov 17, 2025

```
# Antenna Controller System - Master Plan

## Project Overview

**System Name:** Pi Zero 2W Antenna Controller
**Target User:** Single operator (self)
**Primary Goal:** Robust, reliable antenna selection with dual control modes
**Platform:** Ubuntu host + Raspberry Pi Zero 2W controller

## Core Objectives

- Control selection of 3 antennas plus "all off" state
- Support both SSH remote control and local pushbutton operation
- Provide immediate visual feedback via LEDs
- Maintain system state consistency between control modes
- Ensure reliable operation in RF environment
- Minimize complexity for maximum stability

## System Architecture

### Hardware Components
- **Ubuntu Host:** SSH client terminal for remote control
- **Pi Zero 2W Controller:** Main control logic, network interface
- **Buffer/Driver Board:** GPIO signal conditioning, relay drivers
- **3x 12V Relays:** Antenna switching (one per antenna)
- **2x Pushbuttons:** Local control interface
- **3x Status LEDs:** Visual feedback for antenna selection
- **RF Shielded Enclosure:** Houses Pi Zero 2W and control electronics

### Communication Flow
```
Ubuntu Host (SSH) ⟷ WiFi Network ⟷ Pi Zero 2W ⟷ GPIO ⟷ Buffers/Relays/LEDs
                                        ↑
                                   Local Buttons
```

## Core Features

### 1. Antenna Selection Control
- **States:** A1, A2, A3, OFF
- **Default State:** A1 (antenna 1 selected on startup)
- **Switching Logic:** Exclusive selection (only one antenna active at a time)
- **State Persistence:** In-memory only (resets to A1 on reboot)
```

2. Dual Control Modes
- **SSH Mode:** Text command interface from Ubuntu host
- **Local Mode:** Physical pushbutton control
- **Mode Resolution:** Most recent command/action takes precedence
- **No Mode Switching Delays:** Seamless transition between control methods

3. Control Interfaces

SSH Commands
- `A1` - Select antenna 1
- `A2` - Select antenna 2
- `A3` - Select antenna 3
- `OFF` - Disconnect all antennas
- `STAT` - Query current antenna status
- **Response Format:** Current antenna state after each command

Local Button Interface
- **Button 1 (Antenna Toggle):** Cycles A1→A2→A3→A1... continuously
- **Button 2 (OFF Toggle):**
  - First press: Switch to OFF, remember current antenna
  - Second press: Return to previously selected antenna
  - State memory persists through antenna cycling

4. Visual Feedback
- **3x LEDs:** One per antenna (A1, A2, A3)
- **LED States:** Active antenna LED ON, others OFF
- **OFF State:** All LEDs OFF
- **Immediate Updates:** LED state changes instantly with antenna selection

Technical Stack Recommendations

Software Platform
- **OS:** Raspberry Pi OS Lite (minimal footprint)
- **Language:** Python 3.9+
- **GPIO Library:** gpiozero (built-in debouncing, clean API)
- **SSH Server:** OpenSSH (standard Pi OS installation)
- **Process Management:** systemd service for auto-start

Core Libraries
- `gpiozero` - GPIO control with hardware abstraction
- `threading` - Event loop and concurrent input handling
- `logging` - System event logging for debugging
- `socket` - Network communication handling
- `time` - Timing control and delays

Architecture Pattern
- **Event-Driven Design:** Single event loop processing SSH and button inputs
- **State Machine:** Clean antenna state transitions
- **Hardware Abstraction:** gpiozero classes for buttons, LEDs, relays

System Design

GPIO Pin Allocation
```
GPIO Pin    Function        Component
```

```
--------    --------        ---------
GPIO 2      Relay Control   Antenna 1 Relay
GPIO 3      Relay Control   Antenna 2 Relay
GPIO 4      Relay Control   Antenna 3 Relay
GPIO 17     LED Output      Antenna 1 LED
GPIO 27     LED Output      Antenna 2 LED
GPIO 22     LED Output      Antenna 3 LED
GPIO 23     Button Input    Antenna Toggle Button (pull-up)
GPIO 24     Button Input    OFF Toggle Button (pull-up)
```

State Management
```python
# System State Variables
current_antenna = 1  # 1, 2, 3, or 0 (OFF)
previous_antenna = 1  # For OFF button memory
button_states = {
    'antenna_toggle': False,
    'off_toggle': False
}
```

Control Logic Flow

SSH Command Processing
1. Parse incoming command (A1/A2/A3/OFF/STAT)
2. Validate command format
3. Update system state
4. Set relay and LED outputs
5. Return status confirmation
6. Log action

Button Event Processing
1. Detect button press (via gpiozero interrupt)
2. Identify button (antenna toggle vs OFF toggle)
3. Execute appropriate state transition
4. Update hardware outputs
5. Log action

Hardware Output Control
1. Set target relay state (ON/OFF)
2. Update corresponding LED state
3. Ensure exclusive antenna selection (only one relay active)
4. Verify state consistency

Data Model

System State Structure
```
AntennaState:
  - current_selection: int (0=OFF, 1=A1, 2=A2, 3=A3)
  - previous_selection: int (for OFF toggle memory)
  - relay_states: dict {1: bool, 2: bool, 3: bool}
  - led_states: dict {1: bool, 2: bool, 3: bool}
  - last_command_source: str ("SSH" | "BUTTON")
```

```
  - timestamp: datetime
```

Configuration Parameters
```
System Configuration:
  - gpio_pins: dict (relay, LED, button mappings)
  - debounce_time: float (button debounce delay)
  - startup_antenna: int (default antenna on boot)
  - ssh_port: int (SSH server port)
  - log_level: str (logging verbosity)
```

User Interface Design Principles

SSH Interface
- **Simplicity:** Single character commands where possible
- **Consistency:** Predictable command format and responses
- **Feedback:** Immediate status confirmation after each command
- **Error Handling:** Clear error messages for invalid commands

Physical Interface
- **Intuitive Operation:** Logical button functions for radio operators
- **Visual Confirmation:** Immediate LED feedback for all state changes
- **Ergonomic Design:** Two-button interface minimizes complexity
- **Reliable Response:** Debounced inputs prevent accidental triggering

Potential Challenges and Solutions

1. GPIO Reliability
Challenge: Pi Zero GPIO pins sensitive to voltage spikes from relays
Solution: Use buffer/driver circuits with proper isolation and protection
Warning: Monitor for GPIO pin damage during development/testing

2. WiFi Connectivity
Challenge: WiFi drops in RF-rich ham radio environment
Solution: Implement WiFi auto-reconnect, SSH session persistence
Warning: Plan for temporary network outages during experiments

3. Button Debouncing
Challenge: Mechanical contact bounce causing multiple triggers
Solution: gpiozero library provides software debouncing with configurable timing
Warning: Test debounce timing under various pressing speeds

4. State Synchronization
Challenge: Maintaining consistency between SSH and button control modes
Solution: Single event loop processing all inputs with state locking
Warning: Race conditions possible without proper event serialization

5. Power Supply Stability
Challenge: 12V relay switching current affecting Pi Zero power supply
Solution: Separate power domains, proper decoupling capacitors
Warning: Monitor for brown-out resets during rapid relay switching

6. RF Interference

Challenge: Strong RF fields affecting Pi Zero operation
Solution: Proper shielding, ferrite cores on cables
Warning: Test system operation at maximum legal power levels

Testing Strategy

Unit Tests
- GPIO pin control (relay activation/deactivation)
- LED state management (correct LED for each antenna)
- Button input processing (debouncing, state transitions)
- Command parsing (SSH input validation)
- State machine logic (antenna cycling, OFF toggle behavior)

Integration Tests
- SSH command end-to-end (command → relay → LED confirmation)
- Button press end-to-end (press → state change → hardware update)
- Mode switching (SSH command followed by button press)
- Startup sequence (boot → A1 selection → LED activation)
- Error recovery (invalid commands, network interruption)

System Tests
- Extended operation reliability (continuous running)
- RF environment testing (operation during transmission)
- Network robustness (WiFi disconnection/reconnection)
- Physical stress testing (rapid button pressing)
- Temperature stability (enclosed operation)

Acceptance Criteria
- ✓ System boots to A1 state within 30 seconds
- ✓ SSH commands execute with <100ms response time
- ✓ Button presses register reliably without double-triggering
- ✓ Visual LED feedback matches actual antenna selection 100% of time
- ✓ No inadvertent antenna switching during RF transmission
- ✓ 24/7 operation stability for minimum 30-day test period

Development Phases

Phase 1: Core Hardware Control
- GPIO pin configuration and testing
- Basic relay control functionality
- LED status indication
- Single antenna selection logic

Phase 2: SSH Interface
- SSH server setup and configuration
- Command parsing and validation
- Status query implementation
- Error handling and logging

Phase 3: Button Interface
- Physical button input handling
- Debouncing and state management
- Antenna cycling logic
- OFF toggle with memory function

Phase 4: Integration and Testing
- Dual-mode operation testing
- Event loop implementation
- State consistency validation
- RF environment testing

Phase 5: Deployment and Validation
- Production hardware installation
- Extended reliability testing
- Documentation and user procedures
- Backup/recovery procedures

Maintenance Considerations

Regular Monitoring
- Log file review for error patterns
- GPIO pin electrical testing
- Relay contact resistance measurement
- WiFi signal strength monitoring

Preventive Maintenance
- Relay replacement schedule (contact degradation)
- GPIO connector cleaning
- Power supply voltage verification
- Software security updates

Failure Recovery
- Pi Zero replacement procedure
- Configuration backup/restore
- GPIO pin damage troubleshooting
- Network connectivity restoration

Success Metrics

Reliability Targets
- **System Uptime:** >99.9% (excluding planned maintenance)
- **Command Response:** <100ms average latency
- **Button Response:** <50ms press-to-action delay
- **State Accuracy:** 100% LED/relay correspondence

Operational Targets
- **MTBF (Mean Time Between Failures):** >6 months continuous operation
- **Recovery Time:** <2 minutes for complete system restart
- **User Learning Curve:** <5 minutes for new operator proficiency

---

Document Version: 1.0
Last Updated: September 2, 2025
Status: Ready for Development

Pi Zero Hardware Configuration - 2 Antenna System

Pin Mapping Reference

Physical Pin Assignments

Physical Pin	GPIO (BCM)	Function	Direction	Configuration
9	GND	Ground	-	Common ground
11	GPIO 17	Button Input	INPUT	Pull-up enabled (NO switch to GND)
13	GPIO 27	Relay 1 + LED 1	OUTPUT	Active HIGH → ULN2803 pin 1
15	GPIO 22	Relay 2 + LED 2	OUTPUT	Active HIGH → ULN2803 pin 2
17	3.3V	Power Rail	-	LED power supply

Signal Summary

Button (GPIO 17 - Pin 11):

- Type: Normally Open (NO) switch
- Connection: Switch between Pin 11 and Pin 9 (GND)
- Pull-up: Software enabled (reads HIGH when open, LOW when pressed)

Relay 1 + LED 1 (GPIO 27 - Pin 13):

- Output: HIGH = Relay 1 energized, LED 1 on
- Current path: GPIO 27 → ULN2803 pin 1 → Relay coil
- LED path: GPIO 27 → LED → 1KΩ → GND (~3mA, safe)

Relay 2 + LED 2 (GPIO 22 - Pin 15):

- Output: HIGH = Relay 2 energized, LED 2 on
- Current path: GPIO 22 → ULN2803 pin 2 → Relay coil
- LED path: GPIO 22 → LED → 1KΩ → GND (~3mA, safe)

Visual Pin Layout (Used Pins Only)

None

```
        3.3V  (1) (2)  5V

     GPIO 2  (3) (4)  5V

     GPIO 3  (5) (6)  GND

     GPIO 4  (7) (8)  GPIO 14

    GND [9] (9) (10) GPIO 15

GPIO 17 [B] (11) (12) GPIO 18

GPIO 27 [1] (13) (14) GND

GPIO 22 [2] (15) (16) GPIO 23

   3.3V [P] (17) (18) GPIO 24
```

```
    GPIO 10 (19) (20) GND



[B] = Button input

[1] = Relay 1 control + LED 1

[2] = Relay 2 control + LED 2

[P] = Power (3.3V) for LEDs

[9] = Ground
```

System States

Antenna States

State	Relay 1 (GPIO 27)	Relay 2 (GPIO 22)	LED 1	LED 2
A1	HIGH (ON)	LOW (OFF)	ON	OFF
A2	LOW (OFF)	HIGH (ON)	OFF	ON

OFF	LOW (OFF)	LOW (OFF)	OFF	OFF

Control Modes

SSH Commands:

- `A1` - Select antenna 1
- `A2` - Select antenna 2
- `OFF` - Deactivate both antennas
- `STAT` - Query current state

Button Control:

- Press: Cycle A1 → A2 → A1 (continuous toggle)
- Does NOT cycle through OFF state
- Button only switches between active antennas

Hardware Wiring Checklist

✅ **Ground (Pin 9):**

- Connect to button common
- Connect to LED cathodes (via 1KΩ resistors)
- Common ground for all signals

✅ **Button (Pin 11 - GPIO 17):**

- Button terminal 1 → GPIO 17 (Pin 11)
- Button terminal 2 → GND (Pin 9)
- NO pull-up resistor needed (software enabled)

✅ **Relay 1 / LED 1 (Pin 13 - GPIO 27):**

- GPIO 27 → ULN2803 input pin 1
- GPIO 27 → LED 1 anode
- LED 1 cathode → 1KΩ resistor → GND
- ULN2803 output → 12V relay coil

✅ **Relay 2 / LED 2 (Pin 15 - GPIO 22):**

- GPIO 22 → ULN2803 input pin 2
- GPIO 22 → LED 2 anode
- LED 2 cathode → 1KΩ resistor → GND
- ULN2803 output → 12V relay coil

✅ **Power (Pin 17 - 3.3V):**

- Currently not used (LEDs powered through GPIO)
- Available for future expansion

Current Draw Verification

Per GPIO Pin:

- LED current: (3.3V - 2.0V) / 1000Ω ≈ 1.3mA (red LED)
- ULN2803 input: <1mA
- **Total per pin: ~2-3mA** ✅ Well within 16mA GPIO limit

System Total:

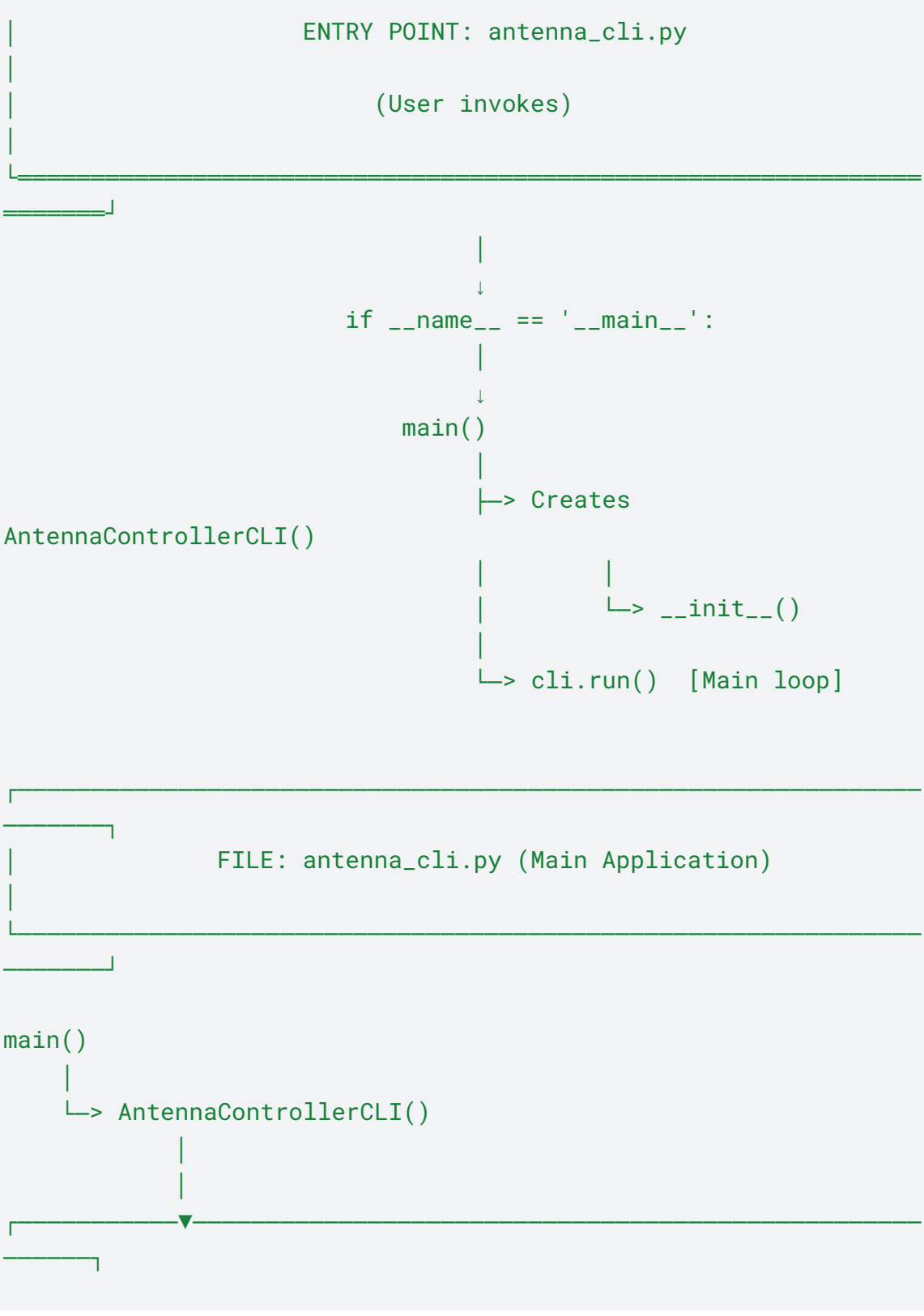
- Maximum simultaneous: 1 GPIO active = 3mA
- Pi Zero safe operating range ✅

==========

Complete System Wireframe - All .py Files

Execution Flow When Running `python3 antenna_cli.py`

```
None
┌═══════════════════════════════════════════════════════════════════════════
═══════┐
```

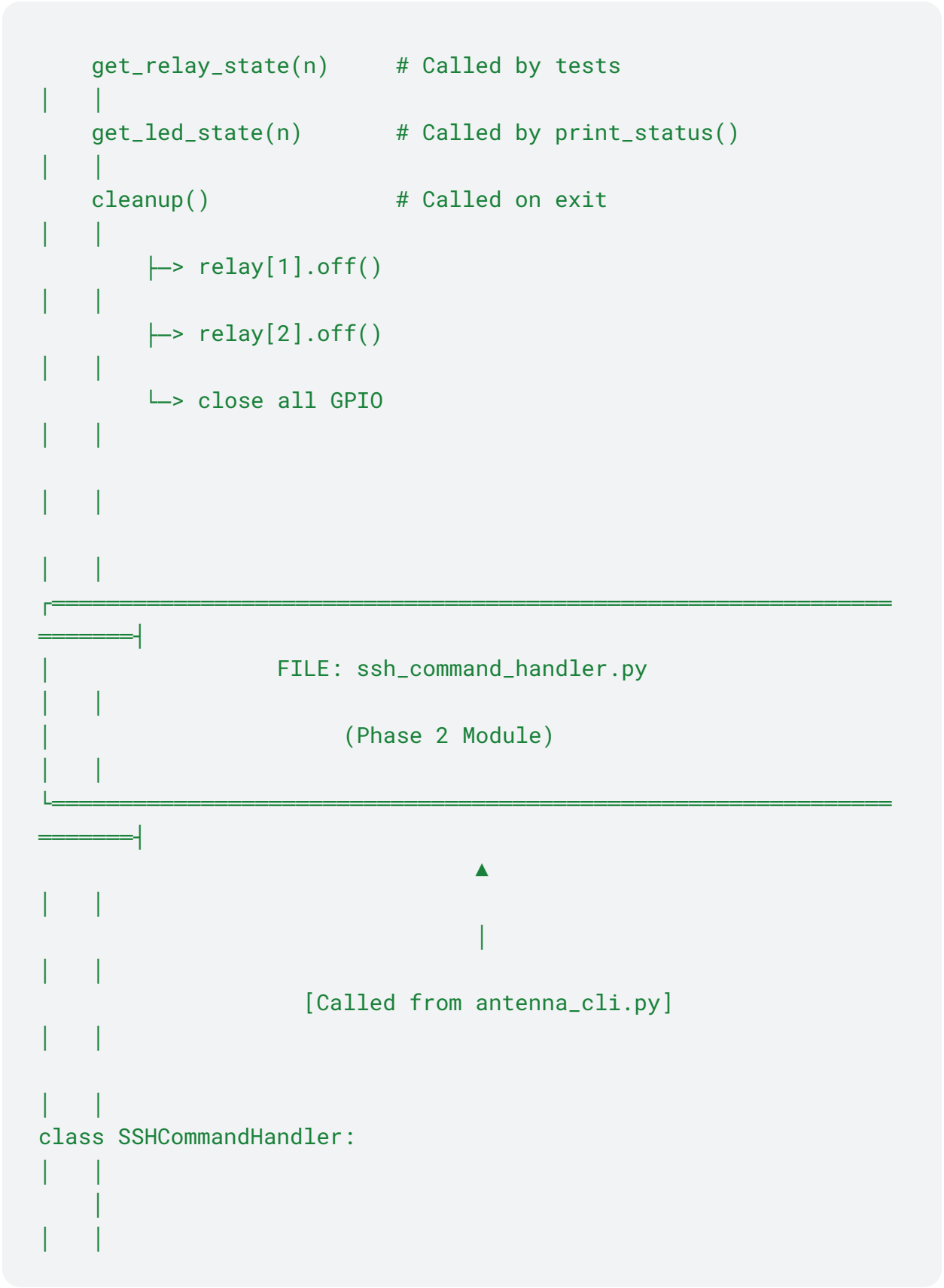

ENTRY POINT: antenna_cli.py
(User invokes)
if __name__ == '__main__':
main()
├─> Creates AntennaControllerCLI()
└─> __init__()
└─> cli.run() [Main loop]
FILE: antenna_cli.py (Main Application)
main()
└─> AntennaControllerCLI()


__init__(self)
[IMPORT & INITIALIZE 3 MODULES]
from antenna_hardware import AntennaHardware
self.hw = AntennaHardware()
Loads: antenna_hardware.py
from ssh_command_handler import SSHCommandHandler
self.ssh_handler = SSHCommandHandler(hw)
Loads: ssh_command_handler.py


from button_handler import ButtonHandler
self.button_handler = ButtonHandler(hw)
Loads: button_handler.py
import signal
import sys
signal.signal(SIGINT, handler) # Ctrl+C
signal.signal(SIGTERM, handler) # kill
signal.signal(SIGHUP, handler) ⚠️ MISSING
run() [Main command loop]


FILE: antenna_hardware.py
(Phase 1 Module)
[Called from antenna_cli.py]
class AntennaHardware:
__init__(self)
[IMPORT] from gpiozero import OutputDevice
[SYSTEM] gpiozero library
[SYSTEM] lgpio backend

```
        |                └─> [KERNEL] /dev/gpiochip0
|   |
        |
|   |
        ├─> self.relay_pins = {1: 27, 2: 22}
|   |
        ├─> self.led_pins = {1: 27, 2: 22}
|   |
        |
|   |
        ├─> Initialize OutputDevice for GPIO 27
|   |
        ├─> Initialize OutputDevice for GPIO 22
|   |
        |
|   |
        └─> self.set_antenna(1)  # Default to A1
|   |
    |
|   |
    set_antenna(n)         # Called by SSH handler
|   |
        ├─> relay[1].off()
|   |
        ├─> relay[2].off()
|   |
        └─> relay[n].on()   # If not OFF
|   |
    |
|   |
    get_current_antenna()  # Called by print_status()
|   |
        └─> return self.current_antenna
|   |
    |
|   |
```


get_relay_state(n) # Called by tests
get_led_state(n) # Called by print_status()
cleanup() # Called on exit
├─> relay[1].off()
├─> relay[2].off()
└─> close all GPIO
FILE: ssh_command_handler.py
(Phase 2 Module)
[Called from antenna_cli.py]
class SSHCommandHandler:

```
    __init__(self, hardware)
|   |
        └─> self.hardware = hardware  # Reference to Phase 1
|   |
    |
|   |
    handle_command(command)  # Main entry point from CLI
|   |
        |
|   |
        ├─> cmd = command.strip().upper()
|   |
        |
|   |
        ├─> Validate against VALID_COMMANDS
|   |
        |   └─> ['A1', 'A2', 'OFF', 'STAT']
|   |
        |
|   |
        ├─> Route command:
|   |
        |   |
|   |
        |   ├─> 'STAT' → _get_status()
|   |
        |   |               └─> hardware.get_current_antenna()
|   |
        |   |
|   |
        |   ├─> 'OFF'  → hardware.set_antenna(0)
|   |
        |   ├─> 'A1'   → hardware.set_antenna(1)
|   |
        |   └─> 'A2'   → hardware.set_antenna(2)
|   |
```


└─> Return status string
_get_status(self)   # Private helper
└─> Format current antenna as string
FILE: button_handler.py
(Phase 3 Module)
[Called from antenna_cli.py]
class ButtonHandler:
__init__(self, hardware, button_pin=17, debounce_time=0.1)
├─> [IMPORT] from gpiozero import Button, Device
├─> [IMPORT] from gpiozero.pins.lgpio import LGPIOFactory
└─> Device.pin_factory = LGPIOFactory()
├─> self.hardware = hardware # Reference to Phase 1
├─> self.button_pin = 17

```
        ├─> self.debounce_time = 0.1
        │
        ├─> self.button = Button(17, pull_up=True,
bounce_time=0.1)
        │       │
        │       └─> [SYSTEM] gpiozero → lgpio → kernel GPIO
        │
        └─> self.button.when_pressed = self._on_button_press
                │
                └─> [ASYNC] GPIO interrupt handler registered
    │
    _on_button_press(self)  # Called by GPIO interrupt
        │
        └─> self.cycle_antenna()
    │
    cycle_antenna(self)     # Toggle A1 ↔ A2
        │
        ├─> current = hardware.get_current_antenna()
        │
        ├─> Determine next:
        │   ├─> 0 (OFF) → 1 (A1)
        │   ├─> 1 (A1)  → 2 (A2)
        │   └─> 2 (A2)  → 1 (A1)
        │
        └─> hardware.set_antenna(next)
    │
    cleanup(self)           # Called on exit
        └─> self.button.close()
            └─> [SYSTEM] Release GPIO 17
```

Complete File Dependency Tree

```
None
antenna_cli.py  [Entry Point - User runs this]
    |
    ├─── antenna_hardware.py
    |        |
    |        └─── gpiozero.OutputDevice    [External Library]
    |                 └─── lgpio           [External Library]
    |                      └─── /dev/gpiochip0  [Kernel]
    |
    ├─── ssh_command_handler.py
    |        |
    |        └─── (No external dependencies)
    |
    ├─── button_handler.py
    |        |
    |        ├─── gpiozero.Button          [External Library]
    |        ├─── gpiozero.Device          [External Library]
    |        └─── gpiozero.pins.lgpio      [External Library]
    |                 └─── lgpio           [External Library]
    |                      └─── /dev/gpiochip0  [Kernel]
    |
    ├─── signal                            [Python stdlib]
    └─── sys                               [Python stdlib]
```

Complete File Dependency Tree

```
None
antenna_cli.py  [Entry Point - User runs this]
    |
    ├─── antenna_hardware.py
    |        |
    |        └─── gpiozero.OutputDevice    [External Library]
    |                 └─── lgpio           [External Library]
    |                      └─── /dev/gpiochip0  [Kernel]
```

```
    |
    ├─── ssh_command_handler.py
    |        |
    |        └─── (No external dependencies)
    |
    ├─── button_handler.py
    |        |
    |        ├─── gpiozero.Button          [External Library]
    |        ├─── gpiozero.Device          [External Library]
    |        └─── gpiozero.pins.lgpio      [External Library]
    |                 └─── lgpio           [External Library]
    |                       └─── /dev/gpiochip0  [Kernel]
    |
    ├─── signal                            [Python stdlib]
    └─── sys                               [Python stdlib]
```

Execution Timeline (Chronological Order)

None

```
T=0ms    User types: python3 antenna_cli.py
         |
T=10ms   Python interpreter starts
         |
T=20ms   Import antenna_cli.py
         ├─> Import antenna_hardware
         |   └─> Import gpiozero, lgpio
         ├─> Import ssh_command_handler
         └─> Import button_handler
             └─> Import gpiozero (already cached)
         |
T=100ms  main() called
         |
T=110ms  AntennaControllerCLI().__init__()
         ├─> AntennaHardware() created
```

```
          |   ├─> GPIO 27 initialized (Relay 1)
          |   ├─> GPIO 22 initialized (Relay 2)
          |   └─> set_antenna(1) → A1 active
          |
          ├─> SSHCommandHandler(hw) created
          |
          ├─> ButtonHandler(hw) created
          |   └─> GPIO 17 initialized (Button with interrupt)
          |
          └─> Signal handlers registered
              ├─> SIGINT  → _signal_handler ✓
              ├─> SIGTERM → _signal_handler ✓
              └─> SIGHUP  → ⚠️ NOT REGISTERED (Test 4 failure)
          |
T=200ms   cli.run() starts
          ├─> print_banner()
          ├─> print_help()
          └─> print_status()
          |
T=250ms   [READY] Waiting for user input
          |
          └─> while True:  [Main event loop]
              |
              ├─> input() waits for command
              |   |
              |   [MEANWHILE - Button handler running async]
              |   └─> GPIO interrupt monitors Pin 17
              |
              ├─> User enters command
              ├─> ssh_handler.handle_command()
              |   └─> hardware.set_antenna()
              └─> print_status()
          |
          [Loop continues until QUIT or signal]
          |
Exit      cleanup() called
```

```
        ├─> button_handler.cleanup()
        └─> hardware.cleanup()
            └─> All GPIO released
```

Memory/Object Relationships

```
None
AntennaControllerCLI (Main object)
    |
    ├─> self.hw (AntennaHardware instance)
    |       |
    |       ├─> self.relays = {1: OutputDevice(27), 2:
OutputDevice(22)}
    |       └─> self.current_antenna = 1
    |
    ├─> self.ssh_handler (SSHCommandHandler instance)
    |       |
    |       └─> self.hardware → [REFERENCE to self.hw above]
    |
    └─> self.button_handler (ButtonHandler instance)
            |
            ├─> self.hardware → [REFERENCE to self.hw above]
            └─> self.button = Button(17)
                    |
                    └─> .when_pressed → [CALLBACK to
_on_button_press()]
```

Key: All three modules share the **same** `AntennaHardware` instance via reference, ensuring state consistency.

==========

Main Application - Interactive CLI

python

```python
# src/antenna_cli.py
"""
Antenna Controller - Interactive CLI Interface
Simple command-line interface for testing antenna control system
"""

import sys
import signal
from antenna_hardware import AntennaHardware
from ssh_command_handler import SSHCommandHandler
from button_handler import ButtonHandler

# Use modern lgpio
from gpiozero import Device
from gpiozero.pins.lgpio import LGPIOFactory
Device.pin_factory = LGPIOFactory()


class AntennaControllerCLI:
    """Interactive CLI for antenna control"""

    def __init__(self):
        """Initialize hardware and handlers"""
        print("Initializing Antenna Controller...")

        try:
            self.hw = AntennaHardware()
            self.ssh_handler = SSHCommandHandler(self.hw)
            self.button_handler = ButtonHandler(self.hw)

            # Setup signal handler for clean shutdown
            signal.signal(signal.SIGINT, self._signal_handler)
            signal.signal(signal.SIGTERM, self._signal_handler)

            print("✓ Hardware initialized")
```

```
            print("✓ Button handler active (GPIO 17)")
            print("✓ Ready for commands\n")

        except Exception as e:
            print(f"✗ Initialization failed: {e}")
            sys.exit(1)

    def _signal_handler(self, sig, frame):
        """Handle Ctrl+C and termination signals"""
        print("\n\nShutting down...")
        self.cleanup()
        sys.exit(0)

    def cleanup(self):
        """Clean up resources"""
        print("Cleaning up GPIO...")
        self.button_handler.cleanup()
        self.hw.cleanup()
        print("✓ Cleanup complete")

    def print_banner(self):
        """Print welcome banner"""
        print("=" * 60)
        print("    ANTENNA CONTROLLER - Interactive CLI")
        print("    2-Antenna System with Physical Button
Control")
        print("=" * 60)
        print()

    def print_help(self):
        """Print available commands"""
        print("Available Commands:")
        print("  A1      - Select Antenna 1")
        print("  A2      - Select Antenna 2")
        print("  OFF     - Deactivate all antennas")
        print("  STAT    - Show current antenna status")
```

```
        print("  HELP    - Show this help message")
        print("  QUIT    - Exit program")
        print()
        print("Physical Button:")
        print("  Press button to toggle A1 ↔ A2")
        print("  (Button does not cycle through OFF state)")
        print()

    def print_status(self):
        """Print current system status"""
        current = self.hw.get_current_antenna()

        print("\n" + "-" * 40)
        print("Current Status:")

        if current == 0:
            print("  Antenna: OFF")
            print("  LED 1:   ○ (off)")
            print("  LED 2:   ○ (off)")
        elif current == 1:
            print("  Antenna: A1 [ACTIVE]")
            print("  LED 1:   ● (ON)")
            print("  LED 2:   ○ (off)")
        elif current == 2:
            print("  Antenna: A2 [ACTIVE]")
            print("  LED 1:   ○ (off)")
            print("  LED 2:   ● (ON)")

        print("-" * 40 + "\n")

    def run(self):
        """Main interactive loop"""
        self.print_banner()
        self.print_help()
        self.print_status()
```

```
        print("Enter commands (type HELP for command list):\n")

        while True:
            try:
                # Get user input
                command = input("antenna> ").strip()

                # Skip empty input
                if not command:
                    continue

                # Check for local commands first
                cmd_upper = command.upper()

                if cmd_upper in ['QUIT', 'EXIT', 'Q']:
                    print("\nExiting...")
                    self.cleanup()
                    break

                elif cmd_upper == 'HELP':
                    self.print_help()
                    continue

                elif cmd_upper == 'STATUS':
                    # Alternative to STAT
                    cmd_upper = 'STAT'

                # Process command through SSH handler
                response =
self.ssh_handler.handle_command(command)

                # Print response
                if "ERROR" in response.upper():
                    print(f"  ✗ {response}")
                else:
                    print(f"  ✓ {response}")
```

```
                # Show status after state changes (not for STAT
query)
                if cmd_upper != 'STAT':
                    self.print_status()

            except EOFError:
                # Handle Ctrl+D
                print("\nExiting...")
                self.cleanup()
                break

            except Exception as e:
                print(f"  ✗ Error: {e}")


def main():
    """Entry point"""
    try:
        cli = AntennaControllerCLI()
        cli.run()
    except KeyboardInterrupt:
        print("\n\nInterrupted. Exiting...")
    except Exception as e:
        print(f"Fatal error: {e}")
        sys.exit(1)


if __name__ == '__main__':
    main()
```

File: ssh_command_handler.py (Updated for 2 antennas)

python

Python

```python
# src/ssh_command_handler.py
"""
SSH Command Handler - 2 Antenna System
Parses and executes antenna control commands via SSH
"""


class SSHCommandHandler:
    """Handles SSH command parsing and execution"""

    # Valid commands for 2-antenna system
    VALID_COMMANDS = ['A1', 'A2', 'OFF', 'STAT']

    def __init__(self, hardware):
        """
        Initialize command handler with hardware reference

        Args:
            hardware: AntennaHardware instance
        """
        self.hardware = hardware

    def handle_command(self, command):
        """
        Parse and execute command, return status response

        Args:
            command (str): Command string from SSH input

        Returns:
            str: Response message with current state or error
        """
        # Strip whitespace and convert to uppercase
        cmd = command.strip().upper()

        # Handle empty command
```

```
        if not cmd:
            return "ERROR: Empty command"

        # Validate command
        if cmd not in self.VALID_COMMANDS:
            return f"ERROR: Invalid command '{command}'. Valid:
A1, A2, OFF, STAT"

        # Execute command
        if cmd == 'STAT':
            return self._get_status()
        elif cmd == 'OFF':
            self.hardware.set_antenna(0)
            return "Status: OFF"
        elif cmd == 'A1':
            self.hardware.set_antenna(1)
            return "Status: A1"
        elif cmd == 'A2':
            self.hardware.set_antenna(2)
            return "Status: A2"

    def _get_status(self):
        """
        Get current antenna status without changing state

        Returns:
            str: Current antenna status
        """
        current = self.hardware.get_current_antenna()

        if current == 0:
            return "Status: OFF"
        else:
            return f"Status: A{current}"
```

Fixed antenna_hardware.py:

python

Python

```python
# src/antenna_hardware.py
"""
Antenna Hardware Control - 2 Antenna System
Controls relays and LEDs for antenna selection on Pi Zero 2W
LEDs are wired in parallel with relay drivers (same GPIO pins)
"""

from gpiozero import OutputDevice


class AntennaHardware:
    """Hardware abstraction for antenna control system"""

    def __init__(self):
        """Initialize GPIO pins and set default state"""
        # GPIO pin mappings - 2 antenna system
        # Relays and LEDs share same pins (LEDs in parallel with relay drivers)
        self.relay_pins = {1: 27, 2: 22}
        self.led_pins = {1: 27, 2: 22}  # Same as relay pins (hardware parallel)

        # Initialize GPIO devices
        # Only create ONE device per pin (relay OutputDevice controls both relay and LED)
        self.relays = {}

        # Setup relays (active high) - this also drives the LEDs
        for antenna_num, pin in self.relay_pins.items():
            self.relays[antenna_num] = OutputDevice(pin, active_high=True, initial_value=False)

        # Set default state (A1 on startup)
        self.current_antenna = 0
        self.set_antenna(1)
```

```
    def set_antenna(self, antenna_num):
        """
        Set active antenna (1, 2) or OFF (0)
        Ensures exclusive selection - only one antenna active at
a time

        Args:
            antenna_num (int): Antenna to activate (1, 2) or 0
for OFF
        """
        # Validate input - now only 0, 1, 2 valid
        if antenna_num not in [0, 1, 2]:
            return

        # Turn off all relays (and LEDs) first
        for i in [1, 2]:
            self.relays[i].off()

        # Turn on selected antenna (if not OFF)
        if antenna_num != 0:
            self.relays[antenna_num].on()

        # Update current state
        self.current_antenna = antenna_num

    def get_current_antenna(self):
        """
        Get currently selected antenna

        Returns:
            int: Current antenna (1, 2) or 0 for OFF
        """
        return self.current_antenna

    def get_relay_state(self, antenna_num):
```

```
        """
        Get relay state for specific antenna

        Args:
            antenna_num (int): Antenna number (1, 2)

        Returns:
            bool: True if relay is active, False otherwise
        """
        if antenna_num not in [1, 2]:
            return False

        return self.relays[antenna_num].is_active

    def get_led_state(self, antenna_num):
        """
        Get LED state for specific antenna
        Since LEDs are wired parallel to relays, LED state =
relay state

        Args:
            antenna_num (int): Antenna number (1, 2)

        Returns:
            bool: True if LED is on, False otherwise
        """
        # LED state is same as relay state (hardware parallel
connection)
        return self.get_relay_state(antenna_num)

    def cleanup(self):
        """Clean up GPIO resources"""
        # Turn off all outputs
        for i in [1, 2]:
            self.relays[i].off()
```

```
        # Close GPIO devices
        for relay in self.relays.values():
            relay.close()
```