MAX32655 EMBEDDED WHAC-A-MOLE

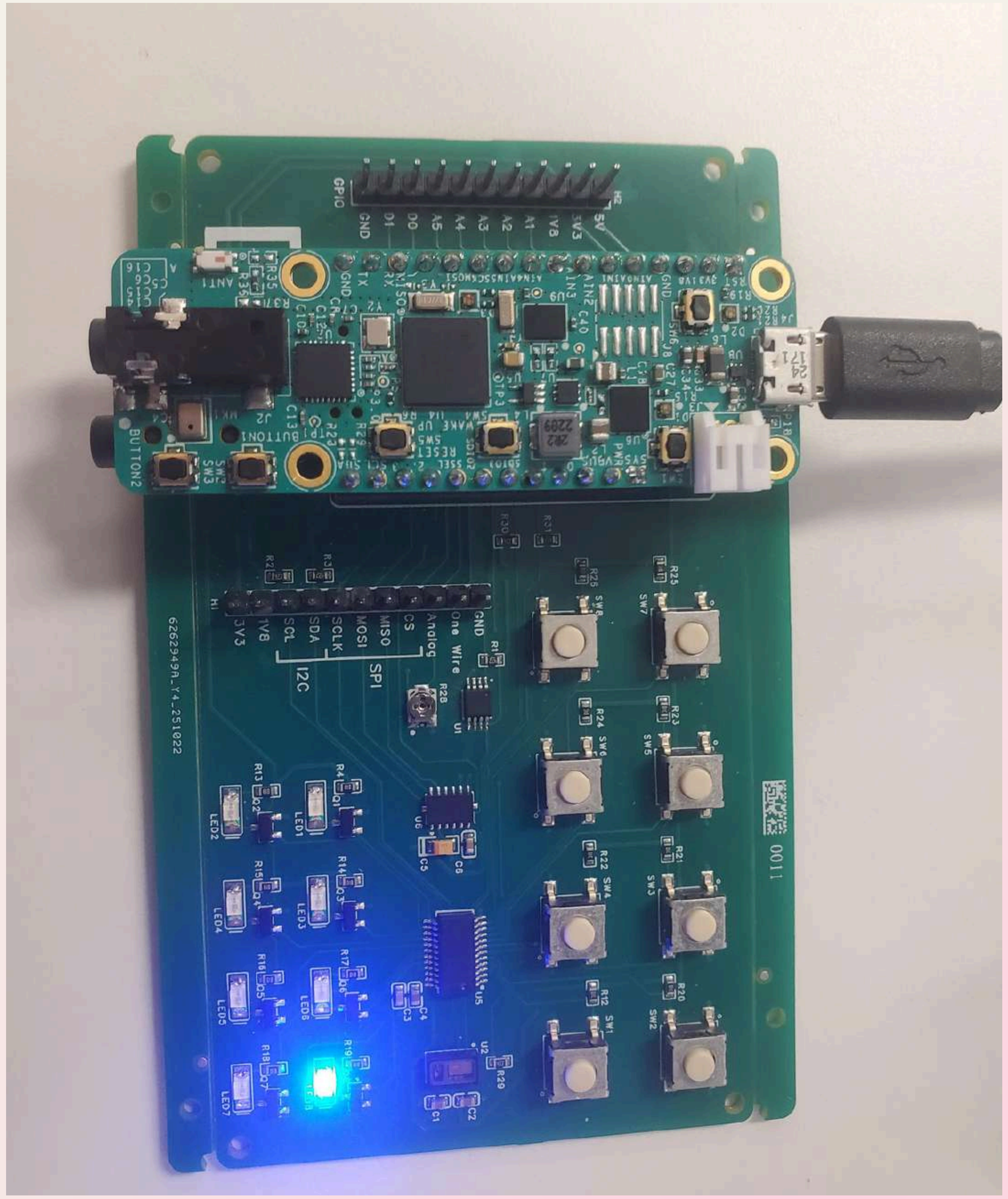
Whac-a-mole

Thomas Joyce
Darragh Luby
JJ

WHAC-A-MOLE

Runs on FreeRTOS, Maxim SDK , on the MAX32655 feather board with the ADI ISE I/O board

- There are 8 LEDs and 8 corresponding buttons
- Hit lit button before light goes out. 8 levels increasingly faster, and 5 lives.



EMBEDDED

1 - Real-time data acquisition
2 - Bi-directional queuing mechanism
3 - Real-time Control Implementation

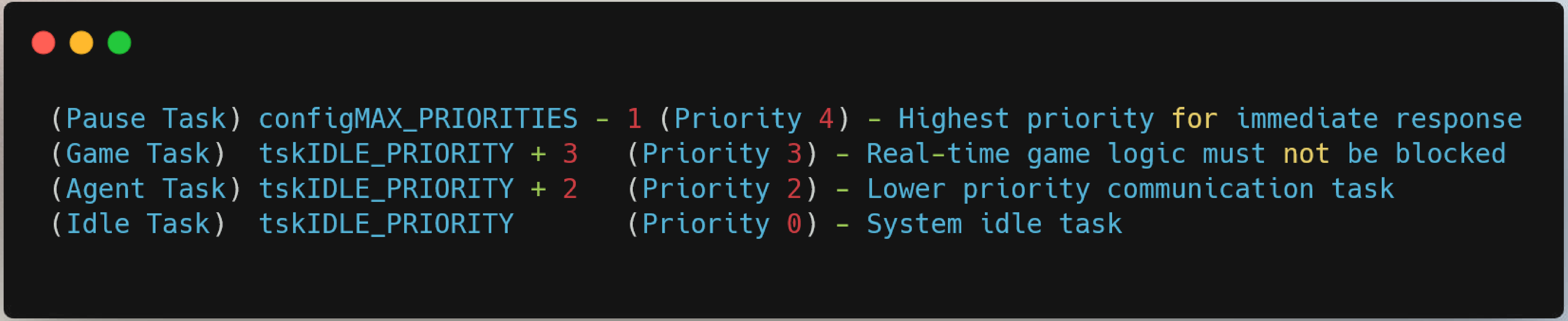
REAL-TIME DATA ACQUISITION AND CONTROL

MULTI-THREADED ARCHITECTURE WITH PRIORITY LEVELS

1. Pause commands preempt everything (safety/control)

2. Game timing remains deterministic (5ms button polling)

3. UART communication doesn't interfere with gameplay



```
(Pause Task) configMAX_PRIORITIES - 1 (Priority 4) - Highest priority for immediate response
(Game Task)  tskIDLE_PRIORITY + 3   (Priority 3) - Real-time game logic must not be blocked
(Agent Task) tskIDLE_PRIORITY + 2   (Priority 2) - Lower priority communication task
(Idle Task)  tskIDLE_PRIORITY       (Priority 0) - System idle task
```

Game task (Priority 3)

LED patterns, deterministic timing, button polls at 5ms intervals

- Must maintain deterministic 5ms button polling
- Real-time LED control cannot be delayed
- Game timing is safety-critical for fair gameplay

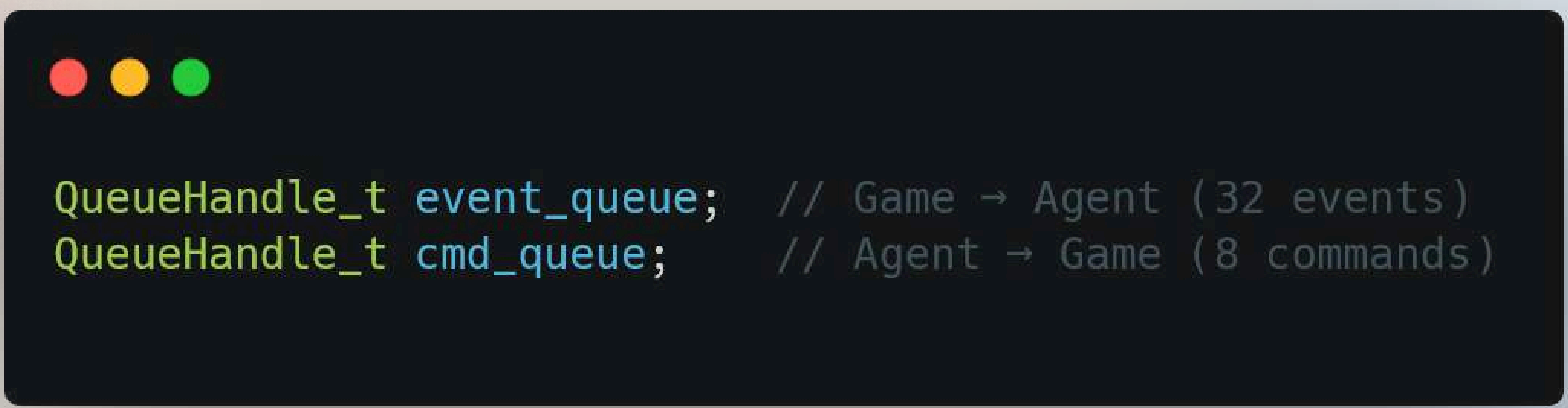
Agent Task (Priority 2)

UART-MQTT bridge functionality
Sends events to dashboard

- UART communication can tolerate delays
- Event transmission is important but not time-critical
- Should never interfere with real-time control loop

BI-DIRECTIONAL QUEUING

(GAME → AGENT)
(AGENT→ GAME)


```
QueueHandle_t event_queue;  // Game → Agent (32 events)
QueueHandle_t cmd_queue;    // Agent → Game (8 commands)
```

Queue 1: Event Queue (Game → Agent)

- Producer: Game task generates events (pop results, level complete, etc.)
- *Consumer*: Agent task drains events and sends via UART
- Thread-safe FIFO with automatic blocking

Queue 2: Command Queue (Agent → Game)

- *Producer*: UART ISR receives commands from dashboard
- *Consumer*: Game task processes commands (level changes, reset, start)
- Bidirectional control from remote dashboard

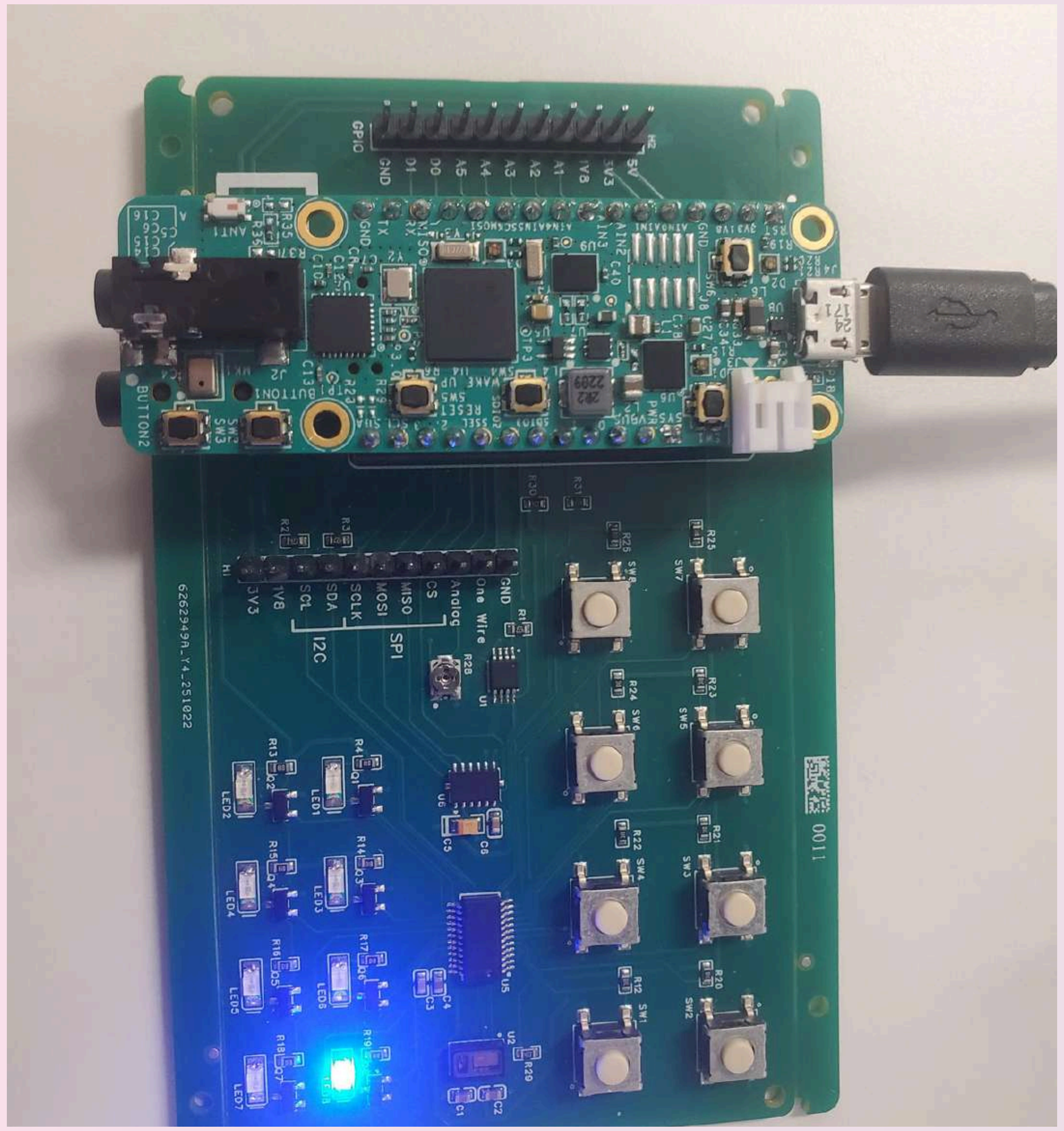
REAL-TIME CONTROL

5ms Button Polling Loop

- Deterministic timing: vTaskDelay() provides tick-accurate delays
- Low latency: Maximum 5ms detection delay
- Priority protection: Higher priority prevents preemption by communication tasks
- Millisecond precision: Reaction time measurement accurate to 1ms

CLOUD SERVER ARCHITECTURE

MQTT, Agent, Dashboard



MQTT BROKER BACKEND

- Automatic reconnection with exponential backoff

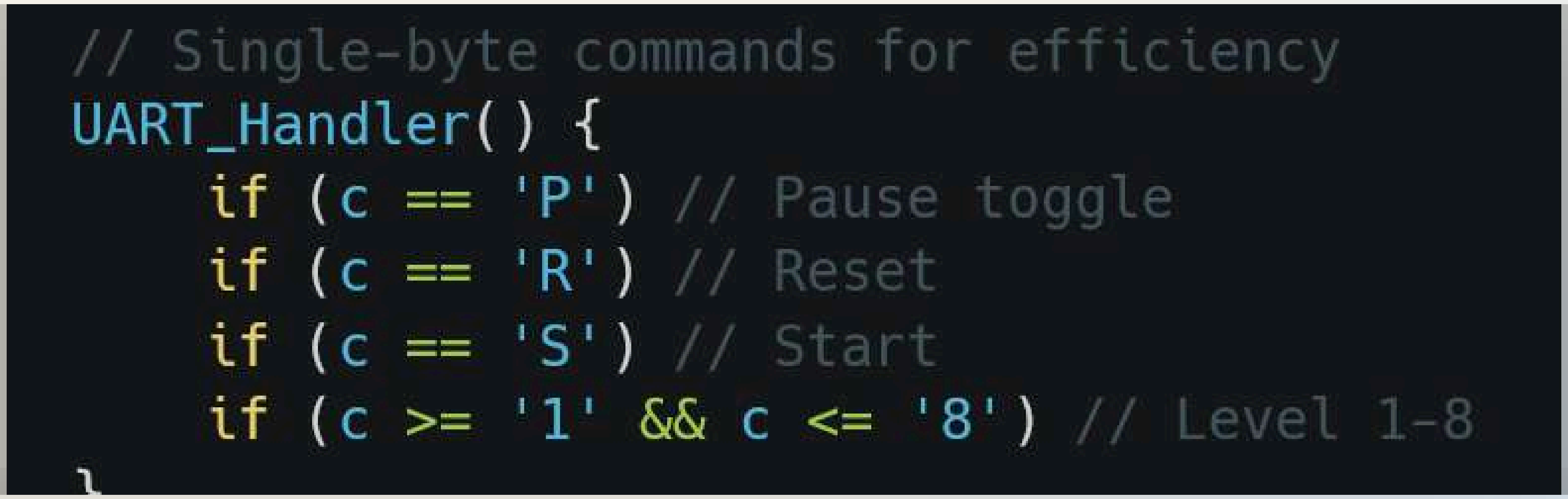
PYTHON BRIDGE AGENT

- USB: Primary connection via UART-to-USB converter
- Serial: Direct UART communication

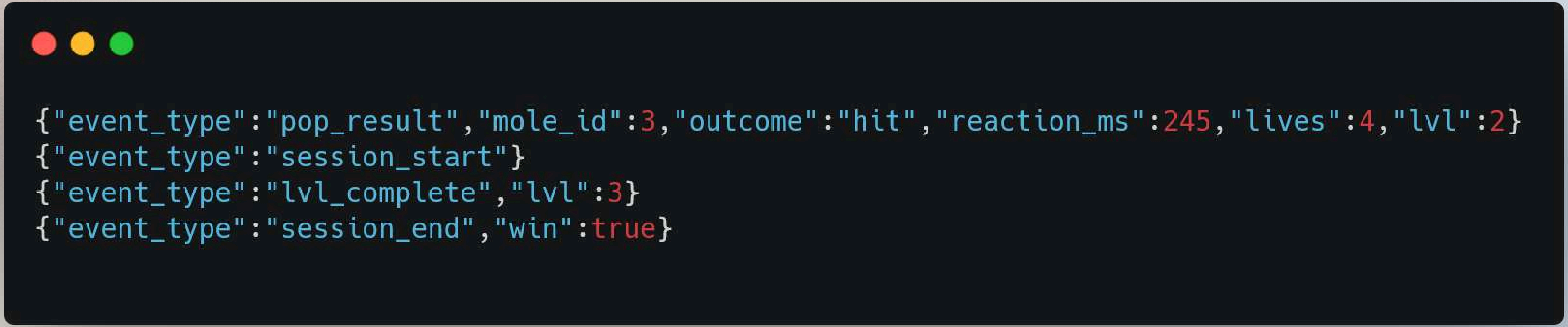
PROPRIETARY COMMUNICATION PROTOCOL

JSON

Agent → Device (Commands)
Device → Agent (Events)



```
// Single-byte commands for efficiency
UART_Handler() {
    if (c == 'P') // Pause toggle
    if (c == 'R') // Reset
    if (c == 'S') // Start
    if (c >= '1' && c <= '8') // Level 1-8
}
```

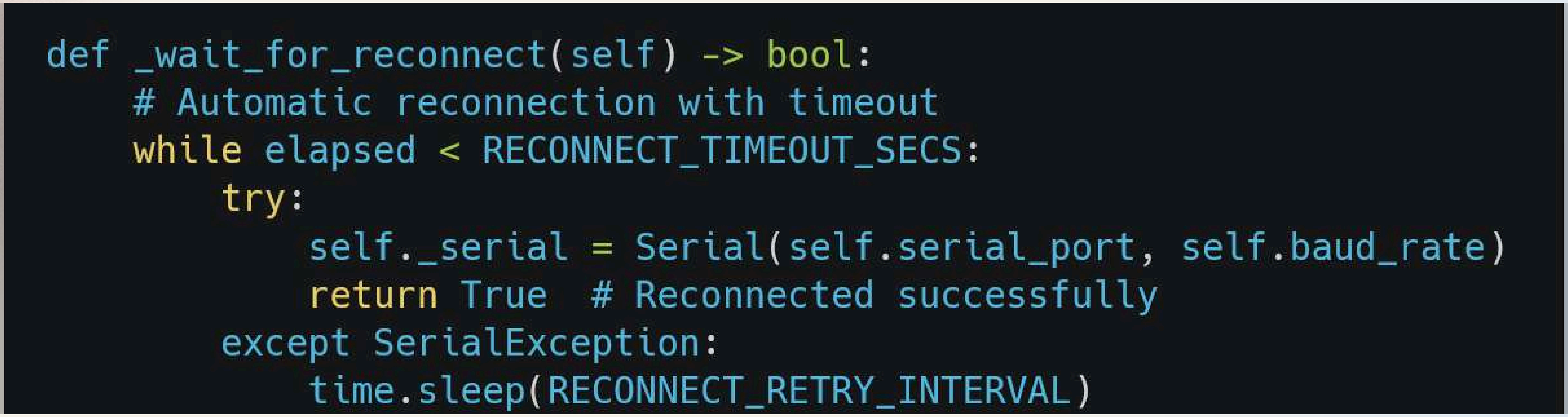


```
{"event_type":"pop_result","mole_id":3,"outcome":"hit","reaction_ms":245,"lives":4,"lvl":2}
{"event_type":"session_start"}
{"event_type":"lvl_complete","lvl":3}
{"event_type":"session_end","win":true}
```

AGENT DISCONNECT TOLERANCE

The embedded device operates independently of agent connection

- Game logic never blocks on communication
- Events queued when agent disconnected
- Automatic drain when agent reconnects
- No data loss during brief disconnections



```python
def _wait_for_reconnect(self) -> bool:
    # Automatic reconnection with timeout
    while elapsed < RECONNECT_TIMEOUT_SECS:
        try:
            self._serial = Serial(self.serial_port, self.baud_rate)
            return True  # Reconnected successfully
        except SerialException:
            time.sleep(RECONNECT_RETRY_INTERVAL)
```

LIVE DEMO!