Presented By:
Thomas, Darragh & JJ

A Whac-A-Mole game for Analog Devices' MAX32655 MCU

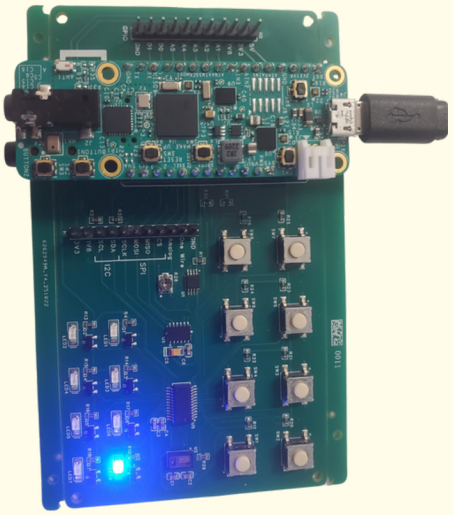
Apache-2.0 MaximSDK 3.12+

TECH

FW
- C
- Dependencies: FreeRTOS, MaximSDK

HW
- MAX32655FTHR EVKit for MAX32655 (ISE I/O Board)

Key Peripherals
- 8 LEDs
- 8 corresponding buttons



THE GAME

Press the button matching the lit LED before it turns off

8 levels – Increasing speed and difficulty
5 lives

REAL-TIME DATA ACQUISITION & CONTROL

MULTI-THREADED 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

```
#define PAUSE_TASK_P (IDLE_TASK_P + 3) // Priority 3 (Highest for immediate preemption)
#define GAME_TASK_P  (IDLE_TASK_P + 2) // Priority 2 (Game logic should not be blocked)
#define AGENT_TASK_P (IDLE_TASK_P + 1) // Priority 1 (Low priority comms)

#define IDLE_TASK_P 0 // Priority 0 (System idle)
```

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
Sends events to the dashboard

- UART communication can tolerate delays
- Event transmission is important but not time-critical
- Should never interfere with the 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 the dashboard
- *Consumer*: Game task processes commands (level changes, reset, start)
- Bidirectional control from the remote dashboard

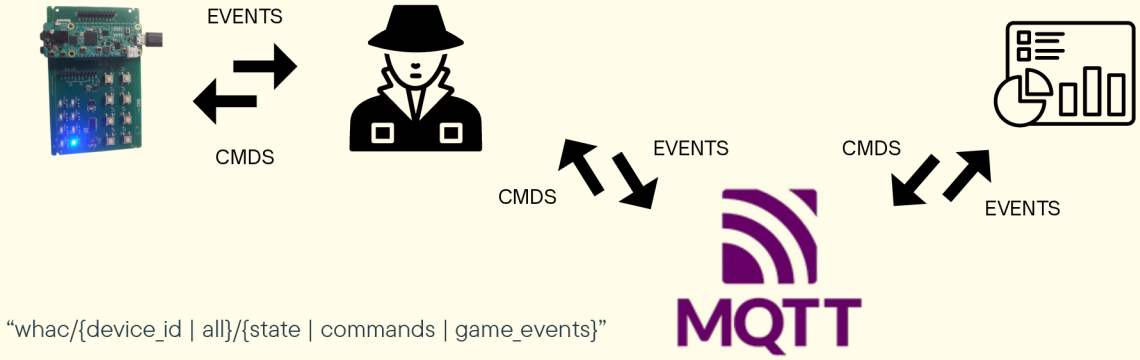
REAL-TIME CONTROL

5ms Button Polling Loop

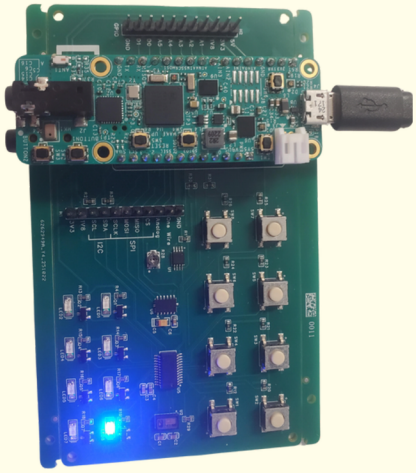
- Deterministic timing: vTaskDelay() provides tick-accurate delays
- Low latency: Maximum 5ms detection delay
- Priority protection: Higher priority prevents preemption by communication tasks
- Millisecond precision: Reaction time measurement accurate to 1ms

CLOUD SERVER ARCHITECTURE

Device / Agent / Dashboard



“whac/{device_id | all}/{state | commands | game_events}”



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

JSONL / SINGLE ASCII BYTE

AGENT → DEVICE (COMMANDS)

```
// 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
}
```

DEVICE → AGENT (EVENTS)

```
{"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

```
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!