Embedathon 26 – ShrimpHub:

The Encrypted Reef Challenge

Final Technical Report

1. Index / Table of Contents

1. Introduction

eam Name: *Pizza Chor*

Team ID: shouryadippizzachor
Reef ID: shouryabitchakrabortypizzachor

Team Members

- **Shouryadip Chakrobarty**
- **Abhishek Agrawal**
- **Joseph Verghese**

Collaborator: ash29062 ✓

Repo Link:

1.1 Team Information

This project was developed by **Team Pizaa Chor** part of **Embedathon 26 – ShrimpHub: The Encrypted Reef Challenge**, a hardware-focused embedded systems hackathon emphasizing **real-time systems, communication protocols, and algorithmic problem solving**.

2.2 Project Overview

The challenge simulated an *ancient reef communication system*, where embedded devices must decode, synchronize, and respond to encrypted environmental signals.

Across **6 mandatory tasks and 1 bonus task**, our system demonstrated:

- Deterministic real-time behavior using **FreeRTOS**
- Reliable IoT communication via **MQTT**
- Precise timing synchronization under strict constraints
- Embedded image decoding and steganography
- Algorithmic image transformation using **Optimal Transport**
- Robust system design under hardware limitations

3. 🧠 Technologies & Architecture

3.1 Core Technologies Used

Technology	Purpose
ESP32 (ESP-IDF)	Primary embedded platform
FreeRTOS	Deterministic multitasking & scheduling
MQTT	Lightweight pub/sub communication
ESP-IDF over Arduino IDE	Fine-grained RTOS + networking control
Python	Image processing & Optimal Transport
OpenCV / PIL / NumPy	Image manipulation
scikit-image	SSIM evaluation

3.2 Why MQTT instead of Plain Arduino + RTOS?

Feature	MQTT	Serial / Plain Arduino
Decoupled communication	✅	❌
Multi-device scalability	✅	❌
Network latency control	✅	❌
Pub/Sub architecture	✅	❌
Real-time event routing	✅	⚠️

MQTT enabled **event-driven design**, aligning perfectly with **FreeRTOS task scheduling**, allowing independent subsystems to react asynchronously without blocking.

4. 🧪 Task-by-Task Documentation

🎵 Task 1: The Timing Keeper

a. Problem Description

Replicate complex RGB LED pulse patterns transmitted via MQTT JSON arrays with **±5ms accuracy**, sustained over **5 minutes without drift**.

b. Our Approach

- Created **three independent FreeRTOS tasks**, one for each RGB channel
- Parsed MQTT payloads using **cJSON**
- Used **vTaskDelayUntil()** to maintain absolute timing
- Protected shared pattern buffers with **mutex semaphores**

c. Key Insights & Challenges

- Relative delays accumulate drift → **absolute scheduling is mandatory**
- MQTT updates can corrupt active patterns → mutex required
- Independent RGB tasks prevent phase interference

e. Results & Timings

Metric	Target	Achieved
Timing Accuracy	±5ms	Within tolerance
Drift over 5 mins	0	None observed

f. Evidence

- 📁 Code: /Task1_TimingKeeper
- 🎥 Video: demo_video.mp4

⚔️ Task 2: The Priority Guardian

a. Problem Description

Handle **two MQTT data streams**, ensuring **distress messages are acknowledged within the required time period**, while continuously computing rolling averages.

b. Our Approach

Implemented **three-tier priority scheduling**:

Task	Priority	Function
Distress Task	High (3)	Immediate ACK + LED
Dispatcher	Medium (2)	Payload routing
Stream Task	Low (1)	Rolling average

Used **FreeRTOS queues** for thread-safe communication.

c. Challenges

- Prevent starvation of low-priority tasks
- Guarantee deterministic preemption
- Maintain real-time ACK publishing

d. Results

Metric	Target	Achieved
ACK Latency	<250ms	~120ms
ACK Success	100%	100%

Task 3: The Window Synchronizer

a. Problem Description

Synchronize a **physical button press** with a digital MQTT window event within **±50ms**.

b. Our Approach

- Button handled using **GPIO interrupts**
- Window events timestamped from MQTT
- Validation via FreeRTOS message queues
- RGB LEDs indicate system state

c. Key Insight

Interrupt-driven inputs + timestamp validation outperform polling in tight timing windows.

d. Results

METRIC	TARGET	ACHIEVED
Sync Accuracy	±50ms	±34ms
Successful Syncs	≥3	Met

🔐 Task 4: The Silent Image

a. Problem

The reef transmitted **non-readable image data** with no explicit message.

b. Insight

The message was embedded in **pixel relationships**, not pixel values.

🔍 How the Steganographic Extraction Works (Task 4)

Instead of embedding data in absolute pixel values (traditional LSB methods), this solution extracts information from **relationships between color channels** in each pixel.

Step-by-step idea:

1. **Image Reconstruction**
   The transmitted image is first reconstructed *exactly* (lossless) and converted to RGB format. This is critical because even minor compression artifacts would destroy relational patterns.
2. **Pixel Relationship Analysis**
   For every pixel, the algorithm compares color channels:
   a. If **Red > Green**, it records a binary 1
   b. Otherwise, it records a binary 0

This comparison encodes information without modifying visible color values.

3. **Binary Stream Formation**
   These binary values are collected sequentially across the image, forming a long binary bitstream.

4. **Binary → ASCII Decoding**
   The bitstream is grouped into 8-bit chunks and converted into ASCII characters.
   Non-printable characters are ignored to reduce noise.
5. **Message Termination Detection**
   Decoding stops once a logical message terminator (e.g., }) is detected, ensuring clean extraction.

Why this works:

- The method hides data in **relative color dominance**, not in raw pixel bits
- Human vision cannot perceive these relational encodings
- The image remains visually unchanged while still carrying structured data

This approach aligns with the task's core insight:

The hidden information depended on relationships, not absolute values.

c. Solution

- Reconstructed PNG losslessly
- Analyzed RGB channel relationships
- Extracted steganographic JSON payload

d. Extracted Message

{
  "key": "ACE!!",
  "target_image_url": "https://shorturl.at/0i9FY"
}

Task 5: The Pixel Sculptor

a. Problem

Rearrange pixels to match a target structure while:

- Preserving colors

- Minimizing movement
- Achieving **SSIM ≥ 0.70**

b. Approach

Used **approximate Optimal Transport**:

- Block-wise (8×8) decomposition
- Luminance-based sorting
- Local greedy matching
- Structural preservation

c. Results

Metric	Target	Achieved
SSIM	≥0.70	≥0.75
Color Fidelity	Exact	Preserved

🌊 Bonus Task: The Plankton Whisper

Problem

Create a **mood-based system** reacting to Serial JSON commands.

Solution

- Parsed incoming JSON:

{ "text": "CALM", "heartbeat": 800 }

- OLED displays mood text
- LED blink rate maps to heartbeat
- Mood changes dynamically

5. Overall Performance Summary

- Mandatory Tasks Completed: **4 / 6**
- Bonus Tasks Completed: **1**
- Average Timing Accuracy: **Within tolerance**
- Final SSIM (Task 5): ≥ **0.75**
- ACK Success Rate: **100%**

6. References & Acknowledgments

- FreeRTOS Documentation
- ESP-IDF Programming Guide
- MQTT Specification
- Introduction To RTOS – DigiKey
  https://www.youtube.com/playlist?list=PLEBQazB0HUyQ4hAPU1cJED6t3DU0h34bz
- ESP32 MQTT in Action – How To Subscribe and Publish Eith ESP-IDF – Controllers Tech
  https://www.youtube.com/watch?v=jCzsN81Gvo0&t=73s
- FREE RTOS Documentation
  https://www.freertos.org/Documentation/00-Overview

Special thanks to:

- Embedathon 26 organizers