# IoT Input Pedal Project Documentation

## 1 Project Overview

The IoT Input Pedal Project is a dual-microcontroller system utilizing an STM32F103C8T6 and an ESP8266 to detect pedal press events, timestamp them using a Real-Time Clock (RTC) synchronized via NTP, and transmit the data to a server over the internet. The STM32 handles pedal detection and RTC management, while the ESP8266 manages WiFi connectivity, NTP synchronization, and HTTP communication with the server.

## 2 System Architecture

The system comprises two main components:

- STM32 Side: Responsible for detecting pedal presses via external interrupts, managing the RTC, and communicating with the ESP8266 via UART with DMA.
- ESP8266 Side: Handles WiFi connectivity, NTP time synchronization, serial communication with the STM32, and HTTP POST requests to the server.

### 2.1 Communication Flow

1. The ESP8266 synchronizes time with an NTP server and sends the epoch time to the STM32 via UART (e.g., epoch:1749109618).
2. The STM32 updates its RTC and sends an acknowledgment (successfully).
3. Upon detecting a pedal press on GPIO PA5, the STM32 debounces the input using Timer2, retrieves the current time and date from the RTC, and sends it to the ESP8266 (e.g., Time:12:45:08 Date:25-06-07).
4. The ESP8266 packages the data into a JSON payload and sends it to the server via an HTTP POST request.

## 3 Features

### 3.1 STM32 Features

- Real-Time Clock (RTC) synchronization using epoch time from ESP8266.
- Software debounce for pedal press detection using Timer2.

- UART communication with DMA and IDLE line detection.
- Pedal press detection via EXTI on GPIO PA5.
- Sends timestamped logs and acknowledgments to ESP8266.

### 3.2 ESP8266 Features

- Connects to a WiFi network using predefined credentials.
- Synchronizes time with pool.ntp.org (offset: +12,600 seconds for Iran).
- Reads serial input from STM32 and sends it as JSON via HTTP POST.
- Updates NTP time every 6 seconds.
- Provides serial monitor feedback for debugging.

## 4 Hardware Requirements

- STM32F103C8T6: For pedal detection, RTC, and UART communication.
- ESP8266: For WiFi, NTP, and HTTP communication (e.g., NodeMCU, Wemos D1 Mini).
- Pedal Switch: Connected to STM32 GPIO PA5.
- LED (Optional): On STM32 PB10 for status indication.

## 5 Software Requirements

- STM32:
  - Development Environment: Keil uVision, PlatformIO, or VS Code.
  - Language: C.
  - Libraries: STM32 HAL for RTC, UART, DMA, Timer2, and EXTI.
- ESP8266:
  - Development Environment: Arduino IDE or PlatformIO.
  - Language: C++.
  - Libraries: ESP8266WiFi, ESP8266HTTPClient, WiFiUdp, NTPClient, ArduinoJson.
- Network: WiFi network and a server endpoint (e.g., http://192.168.1.5/IoT_input_detection.php

# 6 Setup Instructions

## 6.1 STM32 Setup

1. Configure GPIO PA5 for EXTI interrupt and PB10 for LED (optional).
2. Set up UART1 with DMA for communication with ESP8266.
3. Initialize RTC and Timer2 for debounce logic.
4. Upload the firmware using Keil uVision or PlatformIO.

## 6.2 ESP8266 Setup

1. Install required libraries via Arduino Library Manager.
2. Update ssid, password, and endpoint_url in the code.
3. Set offset_time to 12,600 seconds (Iran) or adjust for your time zone.
4. Upload the code using Arduino IDE or PlatformIO.
5. Monitor output via Serial Monitor (baud rate: 115200).

# 7 Code Structure

## 7.1 STM32 Code

- Main Loop: Monitors UART for epoch time and processes pedal interrupts.
- UART Handler: Uses HAL_UARTEx_ReceiveToIdle_DMA to receive messages starting with epoch:.
- EXTI Handler: Detects pedal presses, debounces using Timer2, and sends timestamped logs.
- RTC Functions: Converts epoch time to RTC format and retrieves timestamps.

## 7.2 ESP8266 Code

- setup(): Initializes serial, WiFi, and NTP client.
- loop(): Checks for NTP updates and processes serial input.
- wificonfig(): Connects to WiFi and prints IP address.
- uart_reader(): Reads serial input from STM32.
- time_update_event(): Triggers NTP updates every 6 seconds.
- http_post(): Sends JSON payloads to the server.

# 8 Usage

1. Power on both microcontrollers.
2. Ensure ESP8266 connects to WiFi and synchronizes time.
3. Press the pedal to trigger an interrupt on STM32.
4. STM32 sends the timestamp to ESP8266, which forwards it to the server.
5. Monitor Serial Monitor for debugging (ESP8266) or LED status (STM32).

# 9 Troubleshooting

- WiFi Issues (ESP8266): Verify SSID/password and WiFi range.
- NTP Sync Failure: Check internet and pool.ntp.org accessibility.
- HTTP Errors: Ensure server endpoint is reachable and accepts JSON.
- UART Issues: Confirm baud rate (115200) and wiring between STM32 and ESP8266.
- Pedal Detection: Verify PA5 wiring and debounce settings.

# 10 Future Improvements

- Implement a full UART state machine for STM32.
- Add timeout/validation for epoch time reception.
- Queue multiple pedal presses during UART busy states.
- Enable low-power modes for both microcontrollers.
- Enhance error handling for network and server failures.

# 11 Languages and Tools

- STM32: C, Keil uVision, PlatformIO, VS Code, Git.
- ESP8266: C++, Arduino IDE, PlatformIO, Git.