Integration of Tektork IoT kit with Firebase

1. Introduction

1.1. Overview of ESP32 in IoT Systems

The ESP32 is a versatile and powerful microcontroller featuring integrated Wi-Fi and Bluetooth capabilities, making it a popular choice for Internet of Things (IoT) applications. It supports a broad range of sensors and peripherals and provides robust wireless communication, which is essential for connecting embedded devices to cloud platforms and local servers. The ESP32's low power consumption, rich peripheral sets, and extensive software ecosystem enable developers to implement diverse IoT solutions, including data acquisition, device control, and remote monitoring. This microcontroller bridges the gap between physical sensors and IoT software frameworks such as MQTT, Node-RED, and cloud services like ThingSpeak and Firebase, facilitating seamless device-to-cloud integration and real-time data visualization.

1.2. Objective

This manual aims to guide users through the comprehensive setup, programming, and deployment of various IoT projects by interfacing Tektork IoT Kit with multiple popular platforms and technologies including ThingSpeak, MQTT brokers, Firebase cloud databases, Node-RED for flow-based programming, and the TIG Stack (Telegraf, InfluxDB, Grafana) for advanced data storage, monitoring and visualization. The objective is to provide a step-by-step reference combining hardware setup, firmware development, platform configuration, and dashboard creation, enabling readers to design scalable, networked IoT systems from scratch with detailed technical insights and best practices.

1.3. Required Hardware and Software

Hardware Requirements

For this manual and related projects, the hardware setup is kept minimal and focused on essential components:

- Tektork IoT Kit: The core microcontroller module with built-in Wi-Fi and Bluetooth connectivity, serving as the primary IoT device.
- USB Type-C Cable: Used to both power the ESP32 board and program it from a PC or laptop.
- PC or Laptop: Required for software development, programming, and interfacing with cloud platforms and local systems.

This simplified hardware setup optimizes project accessibility, making it feasible for users to start IoT development with the Tektork IoT Kit, connecting cable, and a host computer without additional peripheral sensors or modules.

Software Requirements:

The software environment consists of tools and platforms necessary for programming the ESP32, managing MQTT communications, cloud data aggregation, and IoT visualization:

- Arduino IDE or ESP-IDF: Primary development environments for writing and uploading firmware to the ESP32 microcontroller.
- MQTT Broker Service: Public brokers like HiveMQ or Mosquitto provide the messaging backbone for IoT device communication.
- ThingSpeak Account: A cloud-based IoT data platform for acquiring, storing, and visualizing data from connected devices.
- Firebase Console: Google's cloud service offering a real-time database, authentication, and hosting for IoT backend needs.
- Node-RED: A visual flow-based programming tool for creating IoT workflows and dashboards without coding complexity.
- TIG Stack Components:
  Telegraf: An agent for collecting and processing time-series data.
  InfluxDB: A time-series optimized database for storing sensor data.
  Grafana: A dashboard platform to visualize and analyze IoT data effectively.
- Supporting Libraries and Tools: MQTT client libraries for ESP32, HTTP client libraries, and SDKs compatible with ESP32 and relevant platforms.

This software setup targets versatility and scalability, enabling development from basic sensor data visualization up to advanced real-time monitoring and control systems.

2. Setting Up the Tektork IoT Kit Development Environment:

Refer the Document tilted

a) Tektork-Kit-ArduinoIDE-Driver-Installation-Notes.pdf

b) Tektork-IoT-Board-Hardware-Manual-V1.0.pdf

3. Firebase with ESP32

3.1. Introduction to Firebase Realtime Database

Firebase Realtime Database is a cloud-hosted NoSQL database that enables real-time data synchronization between client devices and the cloud. It stores data in a JSON format, which makes it flexible for hierarchical data organization. With Firebase, data updates are instantly reflected across all connected clients, making it ideal for real-time IoT applications like sensor monitoring and remote control systems. Its features include offline data persistence, scalability, and simplified integration with various platforms, including embedded devices like ESP32. This enables seamless remote monitoring and control of sensors and actuators over the internet.

3.2. Setting Up a Firebase Project

- Create a Firebase Project
  Navigate to the Firebase Console and click on "Add project". Give your

project a name (e.g., ESP32 IoT Project) and follow the setup wizard. You can enable or disable Google Analytics as needed.

- Enable Realtime Database
  In your project dashboard, select "Realtime Database" from the sidebar. Click "Create Database", choose your location (Asia-Southeast1 as per your code), and start in "Test mode" for open access during development. Later, you can secure your database with rules.
- Add Authentication Method
  Under "Authentication," enable "Email/Password" sign-in method if you intend to use email/password authentication, as in your code. Create a user with the email and password.
- Get Configuration Details
  From "Project Settings" under the "General" tab, copy the "API Key" and "Database URL", which will be used in your ESP32 code to connect to Firebase.

This setup allows your ESP32 to authenticate and exchange sensor data securely with Firebase.

3.3. Connecting ESP32 to Firebase(Wi-Fi and Token Setup)

Below is the full Arduino sketch for connecting the ESP32 to Firebase Realtime Database. It uses the Firebase ESP Client library with token helper addons to manage authentication tokens automatically.

```
#include <WiFi.h>
#include <FirebaseESP32.h> // By Mobizt

#define WIFI_SSID "S23FE"
#define WIFI_PASSWORD "Sanjey123@"

// Firebase project details - note https:// and trailing slash in database_url
#define FIREBASE_HOST "https://esp32-12b7b-default-rtdb.firebaseio.com/"
#define FIREBASE_AUTH "AIzaSyBla5H-b7BtCG4R4bGjv7rwAew4l4A6TVk"

// Use pins from your image:
// POT (RP1) --> GPIO 36
// LDR (R12) --> GPIO 39
#define POT_PIN 36
#define LDR_PIN 39

FirebaseData fbData;
FirebaseConfig config;
FirebaseAuth auth;

void setup() {
  Serial.begin(115200);

  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```

```
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());

  // Configure Firebase
  config.database_url = FIREBASE_HOST;
  auth.api_key = FIREBASE_AUTH;

  Firebase.begin(&config, &auth);
  Firebase.reconnectWiFi(true);
}

void loop() {
  int potValue = analogRead(POT_PIN); // Read potentiometer
  int ldrValue = analogRead(LDR_PIN); // Read LDR

  Serial.printf("POT: %d, LDR: %d\n", potValue, ldrValue);

  // Send POT value
  if (Firebase.setInt(fbData, "/sensors/pot", potValue)) {
    Serial.println("POT value sent to Firebase");
  } else {
    Serial.printf("POT ERROR: %s\n", fbData.errorReason().c_str());
  }

  // Send LDR value
  if (Firebase.setInt(fbData, "/sensors/ldr", ldrValue)) {
    Serial.println("LDR value sent to Firebase");
  } else {
    Serial.printf("LDR ERROR: %s\n", fbData.errorReason().c_str());
  }

  delay(10000); // Wait 10 seconds before sending again
}
```

Line-by-Line Explanation:

- #include <WiFi.h>: Includes the Wi-Fi library to connect the ESP32 to a Wi-Fi network.
- #include <FirebaseESP32.h>: Includes the Firebase library by Mobizt for Firebase Realtime Database integration.

- #define WIFI_SSID "S23FE" and #define WIFI_PASSWORD "Sanjey123@": Define Wi-Fi credentials.
- #define FIREBASE_HOST "https://esp32-12b7b-default-rtdb.firebaseio.com/": The URL of your Firebase Realtime Database (includes https and trailing slash as required).
- #define FIREBASE_AUTH "AIzaSyBla5H-b7BtCG4R4bGjv7rwAew4l4A6TVk": The Firebase project API key (used for authentication).
- #define POT_PIN 36 and #define LDR_PIN 39: GPIO pins connected to the potentiometer and LDR sensors respectively.
- FirebaseData fbData;, FirebaseConfig config;, FirebaseAuth auth;: Objects to manage Firebase connection, configuration, and authentication.
- void setup(): The initialization routine.
  - Serial communication begins for debugging.
  - Wi-Fi connection begins with provided SSID and password.
  - The code waits (looping with delay) until the ESP32 connects to Wi-Fi.
  - Prints Wi-Fi connection success and IP address.
- Firebase configuration is set with the database URL and API key.
- Firebase.begin(&config, &auth);: Connects to Firebase using the configuration and authentication.
- Firebase.reconnectWiFi(true);: Automatically reconnect Wi-Fi if disconnected.
- void loop(): Main program loop running repeatedly.
  - Reads analog value from the potentiometer pin.
  - Reads analog value from the LDR pin.

- Prints sensor values on the Serial Monitor with formatted output.

- Sends the potentiometer value to Firebase Realtime Database location /sensors/pot using Firebase.setInt().
  - On success prints confirmation.
  - On failure prints the error message obtained from Firebase library.
- Sends the LDR value similarly to /sensors/ldr with the same success/error handling.
- Waits for 10 seconds (delay(10000)) before repeating the loop.

3.4. Writing and Reading Data with ESP32

Writing Data

- Use Firebase.setInt(fbData, "/path/to/node", value) to send an integer value to a specific location in the Firebase Realtime Database.
- In your program, this is demonstrated by sending the potentiometer value to /sensors/pot and the LDR value to /sensors/ldr.
- On successful write, the library returns true and you can print confirmation.

- If the write fails, use fbData.errorReason() to get the error message to troubleshoot.

Reading Data

- Reading integer data is done using Firebase.getInt(fbData, "/path/to/node").
- If successful, data can be accessed with fbData.to<int>() and used accordingly.
- On failure, use fbData.errorReason() for error details.

3.5. Working Model Demonstration(Firebase Dashboard Control and Monitoring)

The deployed system allows:

- Monitoring:
  The sensor values (potentiometer and LDR) are continuously uploaded to Firebase and can be visualized in the Firebase Dashboard or third-party tools like Grafana. This real-time visualization helps in analyzing sensor behavior over time.
- Control:
  By extending the database schema to include control commands (e.g., toggling LEDs), users can modify the database entries through Firebase or a web interface. The ESP32 continually polls these entries and executes commands, enabling remote device control.
- Practical Implementation:
  Power the ESP32, ensure Wi-Fi connectivity, and observe serial logs for successful data uploads. Simultaneously, monitor the Firebase Dashboard for updated sensor readings. Adjust control flags remotely to observe the system's response.

This seamless combination of cloud storage, real-time synchronization, and remote control makes Firebase an excellent backend for IoT applications with ESP32.