INTERNET OF THINGS
IBM – NAAN MUDHALVAN COURSE

PHASE 3 : DEVELOPMENT PART – 1

DEPLOYMENT OF SENSORS

Introduction:

In the context of our digital bus station signage project, the deployment of RFID sensors at the entry and exit gates serves a vital purpose. These sensors enable the system to track and manage buses as they enter and exit the station. This documentation provides insights into the deployment, simulation, firmware, and real-time database setup.

Sensor Description:

MFRC522 RFID Reader

The MFRC522 RFID reader is a key component in our system. It plays a pivotal role in reading RFID tags attached to each bus. This reader can communicate with RFID tags efficiently, making it an ideal choice for our project.

- **Role in the Project:** The MFRC522 reader is responsible for reading RFID tags on buses as they pass through the entry and exit gates. It captures the unique identification information from the tags.
- **Key Features and Specifications:**
    - High-frequency RFID reader (13.56 MHz).
    - SPI communication interface.
    - Effective range for RFID tag detection.
    - Compact form factor for easy integration.

Sensor Libraries:

To enable communication between our Arduino-based system and the RFID sensor, we utilize the following libraries:

MFRC522 Library

The MFRC522 library serves as the bridge between our system and the RFID reader. It facilitates the reading and interpretation of RFID tag data.

Additional Libraries

In addition to the MFRC522 library, our project may require additional libraries for interfacing with other components, such as the OLED display. The specific libraries will be detailed in the firmware section of this documentation.

SIMULATION IN WOKWI

Purpose of Simulation:

The primary purposes of simulation include:

- Validating the functionality of the firmware without the need for physical hardware.
- Detecting and resolving bugs or errors in a controlled environment.
- Saving time and resources during the development and testing phases.

Setting Up the Simulation:

Wokwi is the platform we use for simulation.

(*Unfortunately, Wokwi didn't natively support RFID reader simulation. However, we tried to emulate its behavior. For instance, here we used buttons and switches to mimic the action of an RFID tag being read***)**

Connections:

1. **ESP32 to OLED Display**:
    - **GND** of OLED to **GND** of ESP32
    - **VCC** of OLED to **3.3V** of ESP32
    - **SDA** of OLED to **D21** (or any other GPIO you prefer) of ESP32
    - **SCL** of OLED to **D22** (or any other GPIO you prefer) of ESP32
2. **ESP32 to Buttons**:
    - Connect one terminal of **Button1** to **D13** of ESP32 and the other terminal to **GND**
    - Connect one terminal of **Button2** to **D12** of ESP32 and the other terminal to **GND**
    - Connect one terminal of **Button3** to **D14** of ESP32 and the other terminal to **GND**

Configuring the Virtual Components:

- **RFID Reader**:

  Wokwi didn't natively support RFID reader simulation. We tried using buttons to mimic the action of an RFID tag being read.

- **OLED Display**:

  ✓ The OLED display is added to the Wokwi simulation workspace.
  ✓ Connection between the I2C pins (**SDA** and **SCL**) of the OLED is done with the corresponding pins on the NodeMCU.

- **NodeMCU**:

  (*Since the ESP8266 module is not available we tried to emulate it with ESP32***)**

  ✓ The NodeMCU component is dropped to the workspace.

  ✓ Ensure the NodeMCU is set as the primary microcontroller in our project.

  ✓ Connect is done with the other components like the OLED display and your RFID emulation setup.

Defining the Initial State of the System:

Buses & RFID Tags:

✓ Since Wokwi doesn't simulate real-world RFID tags, we are use virtual switches and buttons to represent different buses.

✓ Each button press can be treated as an RFID tag read event, and we have hard-code a specific bus ID for each event within our firmware.

Implement the Firmware within the Simulation Environment:

- In the Code Editor in Wokwi:
- We have coded and have attached the code below.
- Initialized the OLED display and displayed a welcome message or initial state.
- For each button press (emulating RFID reads), fetch the hardcoded bus number, route and platform determine if it's an "Entry" or "Exit", and update the OLED display accordingly.

```
//***Coding***
#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
```

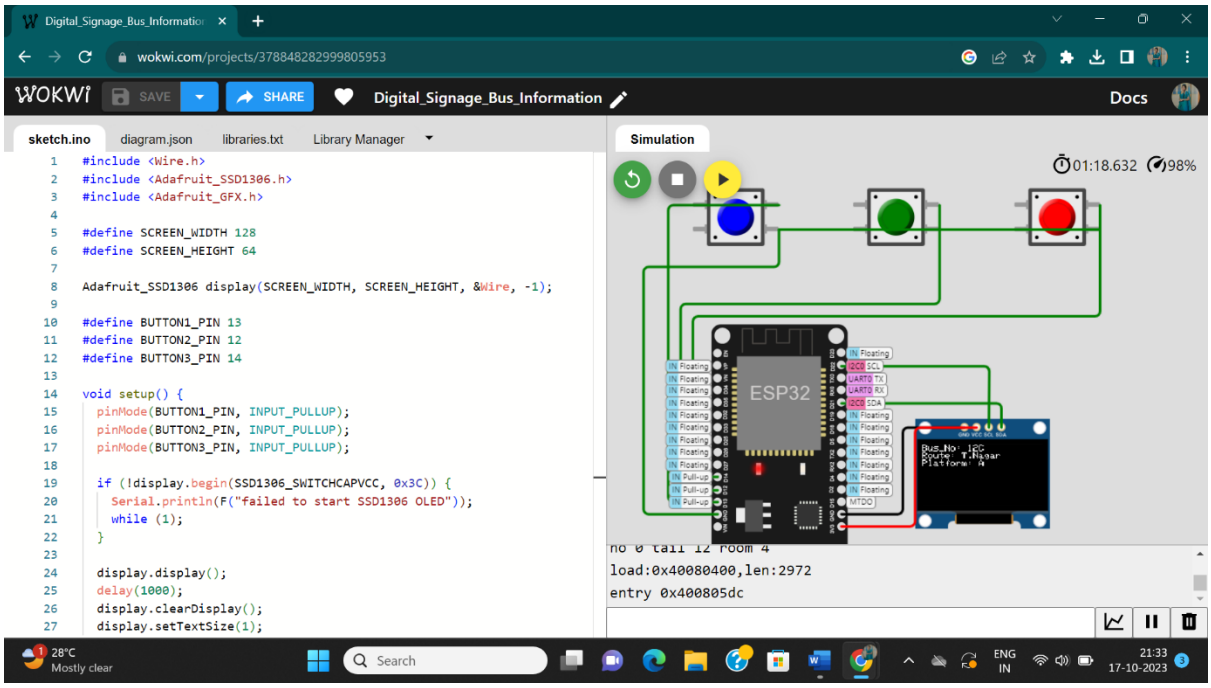
```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

#define BUTTON1_PIN 13
#define BUTTON2_PIN 12
#define BUTTON3_PIN 14

void setup() {
  pinMode(BUTTON1_PIN, INPUT_PULLUP);
  pinMode(BUTTON2_PIN, INPUT_PULLUP);
  pinMode(BUTTON3_PIN, INPUT_PULLUP);

  if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    Serial.println(F("failed to start SSD1306 OLED"));
    while (1);
  }

  display.display();
  delay(1000);
  display.clearDisplay();
  display.setTextSize(1);
}

void loop() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);

  if (!digitalRead(BUTTON1_PIN)) {
    display.setCursor(0,0);
    display.println("Bus_No: 12C");
    display.println("Route: T.Nagar");
    display.println("Platform: A");

  } else if (!digitalRead(BUTTON2_PIN)) {
    display.setCursor(0,0);
    display.println("Bus_No: 7M");
    display.println("Route: Central");
    display.println("Platform: B");
  } else if (!digitalRead(BUTTON3_PIN)) {
    display.setCursor(0,0);
    display.println("Bus_No: 72");
    display.println("Route: Vadapalani");
    display.println("Platform: C");
  }
```

```
  display.display();
  delay(2000);
}
```

//***Coding***

Run the Simulation:

- Click the "Run" or "Play" button in Wokwi to start the simulation.
- Observe the behavior of your components based on the code. For example, pressing one of the buttons should update the OLED display with the corresponding bus's status.
- Link "Click Here"



Simulation link: https://wokwi.com/projects/378848282999805953

REAL-TIME DATABASE DEVELOPMENT

Platform Used: Firebase Real-time Database – A cloud-based NoSQL database for storing and syncing data in real time.

Steps:

Setting Up Firebase Real-time Database:

1. Creating a Firebase Project:

- Navigate to the Firebase website and sign in using a Google account.
- Create a new project by giving it a name and selecting a region.

2. Initializing the Real-time Database:

- Within the project dashboard, click on "Real-time Database" in the left pane.
- Choose "Create Database" and set up in either "locked" or "test" mode based on your preference. (Note: For production, ensure proper security rules are set.)

3. Defining Database Structure:

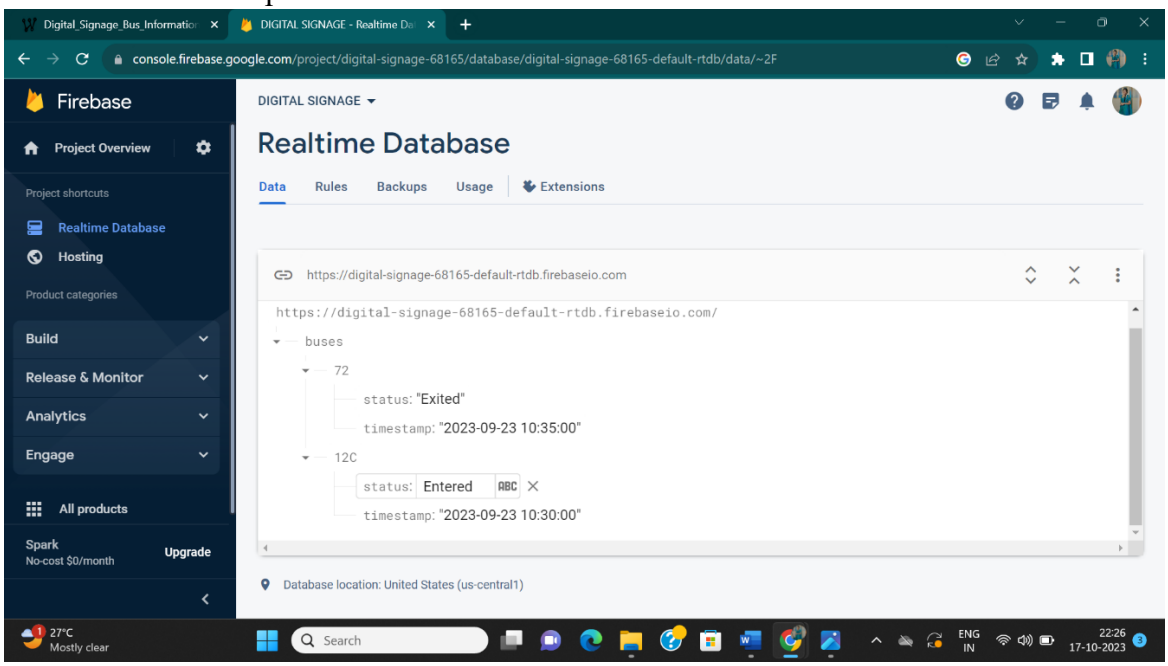
- The database can be envisioned as a JSON tree. For this project, you might structure it as follows:

```
{
 "buses": {
  "12C": {
   "status": "Entered",
   "timestamp": "2023-09-23 10:30:00"
  },
  "72": {
   "status": "Exited",
   "timestamp": "2023-09-23 10:35:00"
  }
```

}

}

- Each bus has a unique identifier (like "bus001") with associated status and timestamp.



4. Integrating with the Firmware:

- Firebase SDKs allow seamless integration with various platforms. For a platform like NodeMCU (ESP8266), relevant libraries can be used to communicate with Firebase over WiFi.
- Once integrated, the firmware can push (or pull) data to (or from) Firebase based on RFID reads or other triggers.

5. Setting Up Security Rules:

- Firebase provides a flexible rules system. Rules define who has read/write access to individual nodes or branches of your database tree.
- Ensure rules are set such that only authorized devices or users can modify or access critical data.