PROJECT GROUP (ESD+ESP) REPORT:

IOT based Home Automation System

Team Members: Raksha Mairpady, Rutuja Muttha, Thanuja Balabaskaran

Project Description:

The primary goal of this project is to design and implement a scalable and efficient home automation system using FPGA technology and an Android application. The system should allow users to remotely control and monitor home appliances, lighting, and other devices through a user-friendly interface.

Committed functionality:

1. Initialize the Nexys A7 FPGA board and set up the Control with Android Application (**Accomplished successfully**)
2. Establishment of connection between MQTT and ESP32(**Accomplished successfully**).
3. Establishment of connection between ESP32 and FPGA (**Accomplished successfully**).
4. Interfacing MPU 6050 sensor to FPGA for temperature measurement. (**Accomplished successfully**)
5. Interfacing the FAN to ESP32 which turns on when the user turns it on via App. (**Accomplished successfully**)
6. Interfacing Stepper_motor_one, to the FPGA which is controlled via ESP32 from the app for Opening/Closing doors of living room. (**Accomplished successfully**)

7. Interfacing Stepper_motor_two and Stepper_motor_three to the FPGA which is controlled via ESP32 from the app for Opening/Closing curtains and door of room. (**Accomplished successfully**)
8. Turning OFF/ON mini bulbs using relay, installed in different living areas using Android App. (**Accomplished successfully)**
9. Controlling the Garage door using Stepper_motor_four. (**Accomplished successfully**)
10. Controlling the Garden door using Stepper_motor_five. (**Accomplished successfully**).
11. Development of a user-friendly application in Android studio using navigation control and fragments to create separate pageviews for each living area as designed in the model. (**Accomplished successfully**).

Stretch functionality:

1. Setting a fire alert using a buzzer and MQ2 - smoke sensor and sending an alert to the user via text. (**Accomplished successfully**).
2. Controlling the plant watering using a moisture sensor and motor. (**Time was not sufficient to accomplish this**).

Implementation:

The Hardware components, software tools, Communication protocols and programming languages used are as described below:

❖ **Hardware Components** :

1) Nexys A7 FPGA x1
2) 28BYJ-48 Stepper motor x 5
3) ULN 2003 Driver x4
4) Fan x1
5) MPU 6050 sensor x1
6) Wires
7) Android Device x1
8) ESP 32 Wifi Module x1
9) Mini Bulbs x5
10) MQ2 sensor x1
11) Buzzer x1
12) 5V Relay module x1

- **Software Tools used:**
  1. Vivado for RTL and Embedded Design Development
  2. Vitis for Firmware design
  3. Android Studio for development of the APP
  4. Arduino IDE for programming ESP32
  5. Logic Analyzer for checking the data and signals coming out from FPGA
  6. Putty to view the print statements as described in the program

- **Software languages used:**
  1. System Verilog used to integrate ESP 32, Stepper Motor and Temperature sensor to Nexys A7 FPGA
  2. C is used for firmware design in vitis and for programming the ESP32 via Arduino IDE.
  3. C++ is used for IP design
  4. Kotlin is used for designing the application.

- **Communication Protocols used**:
  1. I2C to communicate with the sensor from Nexys A7 FPGA
  2. MQTT to communicate with WIFI via APP from ESP32.

Hardware and RTL:

The block diagram of our IOT based home automation is given below:

- All the stepper motors are integrated to the PMODs of Nexys A7 via ULN2003 driver.
- We have used four Pmods JA, JB, JC, JD of FPGA to connect the five stepper motors and ESP32.
- We have a top module named Nexys in the RTL design which has all the instantiations and the code logic for steppers. We have attached the MPU6050 sensor to the PMOD JB, and we have used I2C protocol for the communication.
- We are using MQTT communication protocol to publish and subscribe messages from esp32 and the android application
- Five mini bulbs are connected to esp32 via relay module.
- Fan is connected directly to esp32.
- We have the MQ2 sensor connected to esp32 along with a buzzer.

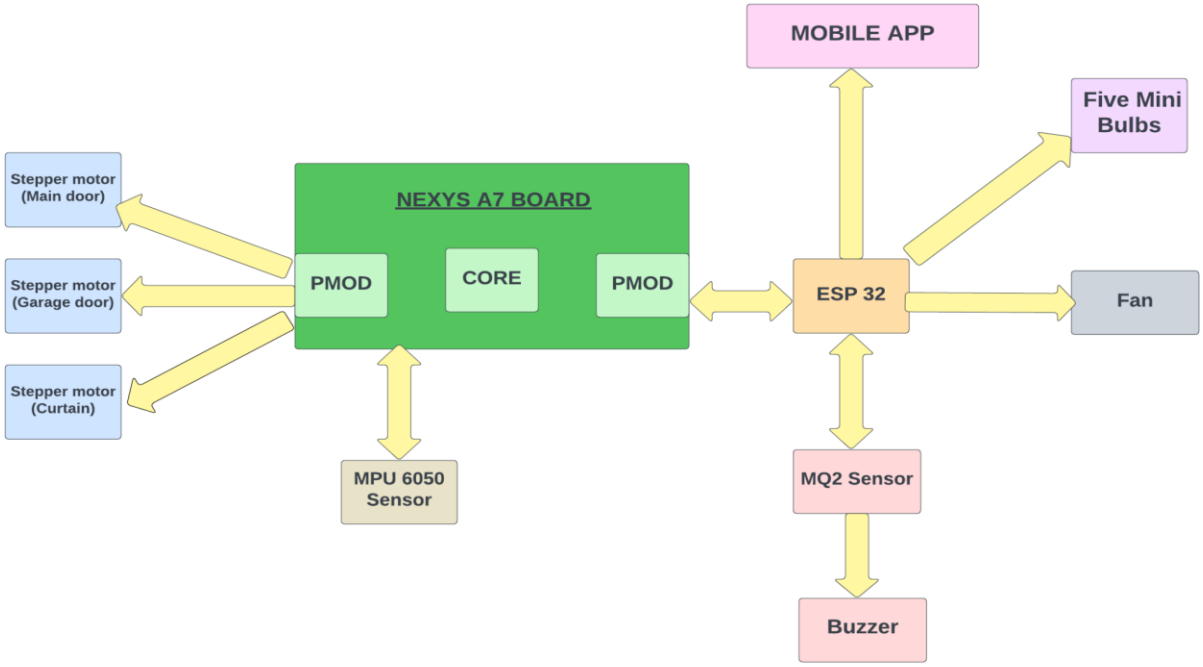

Figure 1: Block diagram

The following are the specifications of the Hardware components used:

- **NEXYS A7 FPGA:**The external connections being used at the board are the programming port, PMOD JA, PMOD JB, PMOD JC and PMOD JD.

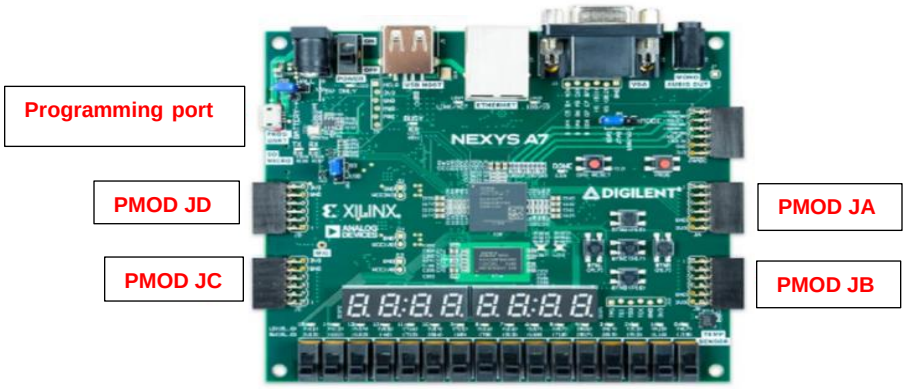

Figure 2: Nexys A7 FPGA

- **28BYJ-48 Stepper Motor:** We have used 5 stepper motors totally in the project, each one has its own different functionality. The 28BYJ-48 is a 5-wire unipolar stepper motor that runs on 5V. Because the motor does not use contact brushes, it has a relatively precise movement and is quite reliable. Despite its small size, the motor delivers a decent torque of 34.3 mN/m at a speed of around 15 RPM. It provides good torque even at a standstill and maintains it as long as the motor receives power. The 28BYJ-48 stepper motor has five wires. The 28BYJ-48 has two coils, each of which has a center tap. These two center taps are connected internally and brought out as the 5th wire (red wire). Together, one end of the coil and the center tap form a Phase. Thus, 28BYJ-48 has a total of four phases. The red wire is always pulled HIGH, so when the other lead is pulled LOW, the phase is energized. The stepper motor rotates only when the phases are energized in a logical sequence known as a step sequence.

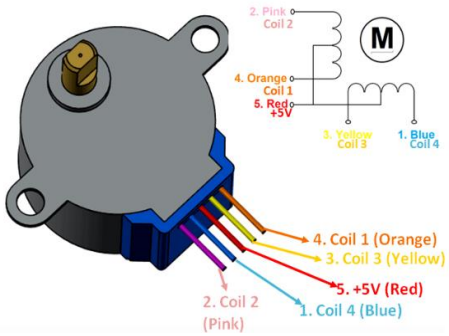

Figure 3: 28BYJ-48 Stepper Motor

- **ULN 2003 Driver:** Because the 28BYJ-48 stepper motor consumes a significant amount of power, it cannot be controlled directly by a microcontroller such as ESP32. To control the motor, a driver IC such as the ULN2003 is required. Therefore, this motor typically comes with a ULN2003-based driver board

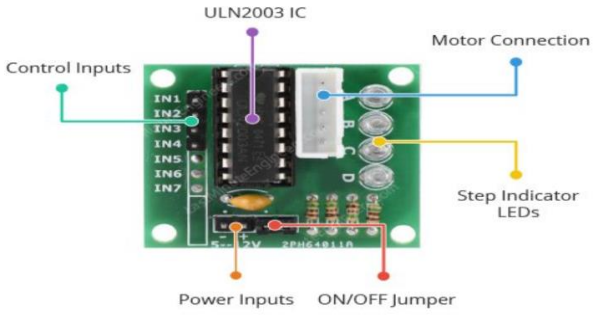

Figure 4: ULN 2003 Driver

The ULN2003, known for its high current and high voltage capability, provides a higher current gain than a single transistor and allows a microcontroller's low voltage low

current output to drive a high current stepper motor.The ULN2003 consists of an array of seven Darlington transistor pairs, each of which can drive a load of up to 500mA and 50V. This board utilizes four of the seven pairs.

- **MPU 6050 Sensor:**The MPU6050 includes an embedded temperature sensor that can measure temperatures from -40 to 85°C with a ±1°C accuracy. Note that this temperature measurement is of the silicon die itself, not the ambient temperature. The module communicates with the Arduino via the I2C interface.

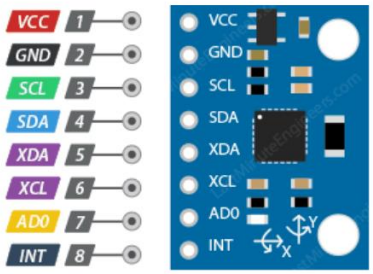

Figure 5: MPU 6050 Sensor

- **ESP 32 Wifi Module:**The ESP32 is a series of low-cost and low-power System on a Chip (SoC) microcontrollers developed by Espressif that include Wi-Fi and Bluetooth wireless capabilities and dual-core processor. the ESP32 consumes very little power compared with other microcontrollers, and it supports low-power mode states like deep sleep to save power. the ESP32 can easily connect to a Wi-Fi network to connect to the internet (station mode), or create its own Wi-Fi wireless network (access point mode) so other devices can connect to it—this is essential for IoT and Home Automation projects—you can have multiple devices communicating with each other using their Wi-Fi capabilities.

Figure 6: ESP 32 Wifi Module

- **MQ2 sensor:** The MQ-2 is a smoke and combustible gas sensor from Winsen. It can detect flammable gas in a range of 300 - 10000ppm. It's most common use is domestic gas leakage alarms and detectors with a high sensitivity to propane and smoke. We

have interfaced this sensor with a buzzer in such a way that the buzzer starts buzzing when moved above a certain threshold.

Figure 7: MQ2 sensor

Figure 8: Buzzer

- **Mini Bulbs:** We have used 5 mini bulbs that are controlled using 5V relay module.

Figure 9: Mini bulbs

Figure 10: Relay Module

- **Fan:** This small fan is attached in the kitchen living area of our design model

Figure 11: Relay Module

DESIGN FLOW:

We have Five different living spaces such as living area, kitchen, garage, bed room and garden. Each living space is given a particular color and represented the same way in the Application. Home page of the android app has all the five living spaces defined. The following applications can be operated in the respective pages:

1) **Living Room**: Here we have on and off buttons to connect and disconnect to MQTT server. Light can be turned off and on, fan can be turned off and on, additionally we can see the temperature readings sensed by mpu6050 displayed in the App.

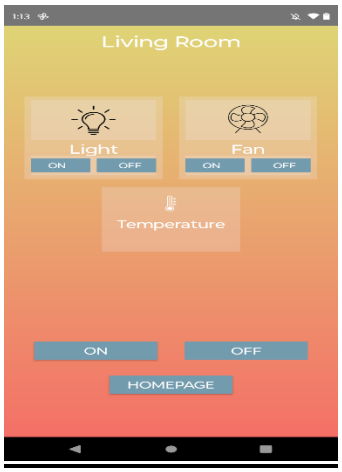

Figure 12: Living Room page in the android app

2) **Kitchen:**

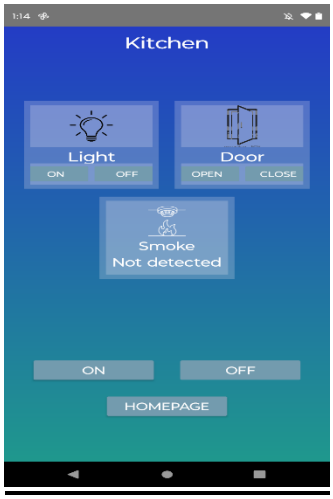

Figure 13: Kitchen page in the android app

Here we have on and off buttons to connect and disconnect to MQTT server. Light can be turned off and on, door can be opened and closed, additionally we can see the caution message displayed if the smoke is detected from mq2 sensor.

3) **Bedroom**: Here we have on and off buttons to connect and disconnect to MQTT server. Light can be turned off and on, door can be opened and closed, and curtains can be opened and closed

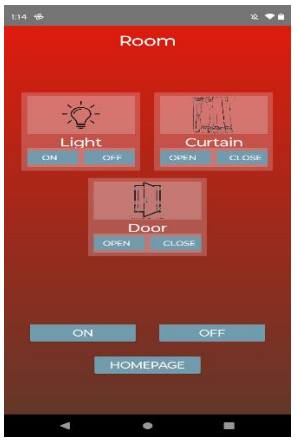

Figure 14: Bed Room page in the android app

4) **Garage:** Here we have on and off buttons to connect and disconnect to MQTT server. Light can be turned off and on, door can be opened and closed.

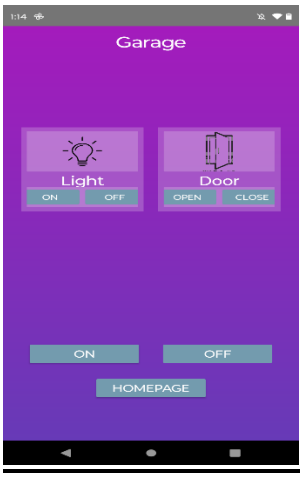

Figure 15: Garage page in the android app

5) **Garden**:

Figure 13: Garden page in the android ap

Here we have on and off buttons to connect and disconnect to MQTT server. Light can be turned off and on, door can be opened and closed.

Firmware program design flow (designed in Vitis):

The program flow is as defined below:

- **Initialization and Setup:**

1. Header Comments and Include Files:

- The file begins with comments providing an overview of its purpose and the authors.

- Various header files are included to enable access to necessary libraries and functions.

2. Constants and Definitions:

- Constants are defined for clock frequencies, peripheral addresses, delay duration, and motor directions.

- Peripheral device IDs and base addresses are defined for peripherals like Nexys4IO and AXI IIC.

3. Global Variables and Flags:

- Global variables are declared to track the running status of motors and maintain motor control parameters.

- Flags like `motor_running1`, `motor_running2`, ..., `motor_running10` indicate whether each motor is currently running or not.

- Other variables hold information like the number of steps per revolution, current step number, and motor movement direction.

4. Peripheral Instances:

- Instances of XIic (for I2C communication) and XGpio (for GPIO control) are declared to interact with respective peripherals.

5. Function Prototypes: Prototypes for initialization functions, delay function, and motor control functions are declared for later use.

- **Main Function:**

1. Initialization: The main function begins by initializing the platform and necessary peripherals. The initialization status is checked, and the program terminates if initialization fails.

2. Continuous Operation:

- The main loop continuously runs, handling incoming data and controlling motors based on commands.

- It starts by reading temperature data from the MPU6050 sensor and processing it for output.

- It then reads discrete signals from GPIO pins connected to an ESP32 module for motor control.

- Based on received signals, it determines which motors to control and in which direction (start/stop)

3. Motor Control:

- For each motor, the program checks the corresponding GPIO signal from the ESP32.

- If the signal indicates starting, it sets the motor's running status and direction accordingly.

- The motor control logic then steps through the motor's rotation sequence based on its running status and direction.

- Each motor's stepping logic is encapsulated in dedicated functions (`stepMotorOne`, `stepMotorTwo`, etc.).

- **Helper Functions:**

1. Initialization and Setup Functions:

- `do_init`: Initializes the Nexys4 driver and checks the status for successful initialization.

- `InitializeIIC`: Initializes the I2C peripheral instance (currently a placeholder).

- `gpioinit`: Initializes GPIO instances for stepper outputs and inputs from the ESP32, checking for initialization status.

- `setting_direction`: Configures the data direction for GPIO instances for stepper outputs and motors

2. Utility Functions: `delay`: Creates a simple delay loop to introduce delays in operations, used for motor control timing

3. Motor Control Function:`stepMotorOne`, `stepMotorTwo`, etc.Control the rotation of respective stepper motors by executing a step in their rotation sequence based on the given step index and direction. These functions select the correct sequence (CW or CCW) and trigger the corresponding coil activation pattern via GPIO.

ESP32 program code design flow:

The code flow of the program dumped in esp32 is as shown below:

1. Include Libraries and Define Constants:

- Libraries such as Arduino, WiFi, AsyncMqttClient, and AsyncTCP are included.

- Pin mappings and constants are defined for components such as LEDs, stepper motors, buzzer, gas sensor, WiFi credentials, MQTT server details, gas threshold, and duration for stepper motor control.

2.WiFi Connection:

- The `connectToWiFi()` function attempts to connect to the specified WiFi network using the provided SSID and password.

- It continuously checks the WiFi connection status until connected, printing dots to the serial monitor.

- Once connected, it prints the IP address to the serial monitor.

3. MQTT Setup:

- The `setupMQTT()` function sets up MQTT communication.

- It defines a callback function `callback()` to handle incoming MQTT messages.

- It subscribes to various MQTT topics related to home automation, indicating which topics the device is interested in receiving messages from.

4. Setup Function:

- Serial communication is initialized for debugging purposes.

- WiFi connection is established by calling `connectToWiFi()`.

- MQTT setup is executed by calling `setupMQTT()` and setting the MQTT server details.

- Pin modes are set for various components including LEDs, buzzer, gas sensor, and stepper motors.

5. Main Loop:

- Gas sensor reading is taken using analogRead().

- If the MQTT client is disconnected, it resets the state of various components to their default states.

- If the gas sensor reading exceeds the threshold, a message indicating gas detection is published over MQTT, and the buzzer is turned on.

- The MQTT client periodically re-subscribes to MQTT topics to ensure continuous communication.

- There's a 100 milliseconds delay at the end of each loop iteration.

6. MQTT Message Handling (Callback Function):

- The `callback()` function handles incoming MQTT messages.

- It checks the received topic and performs specific actions based on the topic:

  - For lighting control topics, it turns corresponding LEDs on or off.

  - For fan control topics, it turns the fan on or off.

  - For temperature and smoke sensor topics, it prints the received data to the serial monitor and sends it to the apllication

  - For stepper motor control topics, it activates specific stepper motors to perform actions such as opening or closing doors or curtains.

7. Stepper Motor Control:

- Stepper motors are controlled within the `callback()` function based on the received MQTT topics.

- When a topic related to stepper motor control is received, the corresponding stepper motor pins are set to HIGH to move the motor in a specific direction.

- After a specified duration (`high Duration`), the stepper motor pins are set to LOW to stop the motor.

Android studio design :

The combined overall code flow

1. MainActivity.kt

- Entry point of the application.

- Sets up the main activity layout and navigation component.

2. InitialPage Fragment (initialpage.kt)

- Displays the initial page of the home automation application.

- Contains a button to navigate to the homepage.

3. Homepage Fragment (homepage.kt)

- Displays the homepage of the home automation application.

- Provides buttons to navigate to different rooms and functionalities.

4. LivingRoom Fragment (livingroom.kt)

- Manages controls and status updates for devices in the living room.

- Connects to an MQTT broker to send and receive messages related to device controls and status.

- Handles UI interactions for controlling lights and fan.

- Displays temperature updates received from MQTT.

5. Kitchen Fragment (kitchen.kt)

- Manages controls and status updates for devices in the kitchen.

- Connects to an MQTT broker to send and receive messages related to device controls and status.

- Handles UI interactions for controlling lights and door.

- Displays smoke detection updates received from MQTT.

6. Bedroom Fragment (bedroom.kt)

- Manages controls and status updates for devices in the bedroom.

- Connects to an MQTT broker to send and receive messages related to device controls and status.

- Handles UI interactions for controlling lights, door, and curtain.

- Checks for internet connectivity and initializes MQTT client.

- Provides functionality to connect/disconnect from the MQTT broker.

- Subscribes to MQTT topics for receiving updates.

- Publishes messages to control devices and updates UI accordingly.

7. Garden Fragment (garden.kt)

- Manages controls and status updates for devices in the garden.

- Connects to an MQTT broker to send and receive messages related to device controls and status.

- Handles UI interactions for controlling lights and door.

- Checks for internet connectivity and initializes MQTT client.

- Provides functionality to connect/disconnect from the MQTT broker.

- Subscribes to MQTT topics for receiving updates.

- Publishes messages to control devices and updates UI accordingly.

8. MQTT Client Setup (MqttClient.kt)

- Manages MQTT communication with the broker.

- Connects to the broker, subscribes to topics, and publishes messages.

- Handles success and failure callbacks for MQTT actions.

9. Constants File (Constants.kt)

- Contains constant values used throughout the application, such as MQTT server URI, topics, and messages.

10. Data Binding Files (Fragment*.xml)

- XML layout files corresponding to each fragment.

- Define the layout and UI elements for each fragment.

Overall Flow:

1. MainActivity sets up the main activity layout and navigation.

2. InitialPage Fragment displays the initial page with a button to navigate to the homepage.

3. Homepage Fragment provides navigation to different rooms and functionalities.

4. LivingRoom, Kitchen, Bedroom, and Garden, garage Fragments manage device controls and status updates for specific areas using MQTT.

5. MQTT Client handles communication with the MQTT broker for sending and receiving messages.

6. Constants file stores constant values used throughout the application.

7. Data binding files bind XML layout elements to Kotlin code for efficient UI access.

Problems Faced (Fail Vlog):

1) We successfully transmitted data from the ESP32 to another Pmod module, but encountered challenges when attempting to utilize this data to control a stepper motor. Subsequently, with assistance from Omkar, we identified that the root cause lay within the RTL (Register Transfer Level) design. Specifically, the Embsys wrapper generated was not being wrapped correctly, resulting in operational issues. Additionally, we discovered that internally hardwiring the Pmod within the RTL was restricting our ability to make desired modifications in the firmware.
2) We encountered difficulties when attempting to illuminate a miniature bulb using the ESP32. Subsequently, we addressed this challenge by integrating a relay to facilitate control over the bulb, effectively resolving the issue.
3) During the development of our application, we encountered various Gradle deprecation warnings and cache-related issues. However, we successfully resolved these issues by implementing necessary Gradle modifications tailored to the requirements of our application.

4) Integrating all the parts of our design together gave us little trouble as the components were responding differently on integration.

Division of Labor:

- Integration of components (stepper motor, led, esp 32, fan, sensors) to Nexys A7 by Raksha Mairpady.

- IP drivers code design for all the sensors and motors by Rutuja Muttha

- Android app creation By Thanuja Balabaskaran.

References:

1) https://randomnerdtutorials.com/getting-started-with-esp32/
2) https://lastminuteengineers.com/28byj48-stepper-motor-arduino-tutorial/
3) https://lastminuteengineers.com/mpu6050-accel-gyro-arduino-tutorial/#:~:text=The%20MPU6050%20includes%20an%20embedded,itself%2C%20not%20the%20ambient%20temperature.

4) https://www.circuits-diy.com/control-28byj-48-stepper-motor-with-uln2003-driver/
5) https://developer.android.com/guide/navigation/design/actions
6) https://medium.com/@mr.appbuilder/navigating-android-fragments-with-the-navigation-component-part-1-1d238e000313