// all task

xTaskCreate(InitializationTask, "InitTask", 128, NULL, 3, NULL);

xTaskCreate(StateMachineTask, "StateMachine", 256, NULL, 2, NULL);

xTaskCreate(CommunicationTask, "CommTask", 256, NULL, 2, NULL);

xTaskCreate(MonitoringTask, "MonitorTask", 128, NULL, 1, NULL);

// init state

void InitializationTask(void \*argument) {

// Initialize hardware

SystemInit();

// Initialize peripherals like UART, GPIOs

PeripheralsInit();

// Initialize SIM7600

SIM7600\_Init();

// Connect to MQTT Broker

MQTT\_Connect();

// After initialization, this task can delete itself or go to sleep

vTaskDelete(NULL);

}

// state machine task

typedef enum {

IDLE,

CHECK\_MQTT\_CONNECT,

READ,

PROCESS,

PUBLISH,

RECONNECT\_MQTT,

HANDLE\_ERROR

} SystemState;

SystemState currentState = IDLE;

void StateMachineTask(void \*argument) {

while (1) {

switch (currentState) {

case IDLE:

// Stay in idle until a trigger to start is detected

if (checkForTrigger()) {

currentState = CHECK\_MQTT\_CONNECT;

}

break;

case CHECK\_MQTT\_CONNECT:

// Check and ensure MQTT is connected before proceeding

if (!MQTT\_IsConnected()) {

currentState = RECONNECT\_MQTT;

} else {

currentState = READ;

}

break;

case READ:

// Read data from the MFM383A sensor

if (readDataFromSensor(&sensorData) == SENSOR\_OK) {

currentState = PROCESS;

} else {

currentState = HANDLE\_ERROR;

}

break;

case PROCESS:

// Process the sensor data

processData(&sensorData);

currentState = PUBLISH;

break;

case PUBLISH:

// Attempt to publish the processed data

if (!publishToMQTT(sensorData)) {

currentState = HANDLE\_ERROR;

} else {

currentState = IDLE;

}

break;

case RECONNECT\_MQTT:

// Try to reconnect to MQTT

if (!MQTT\_Connect()) {

osDelay(1000); // Delay to mitigate flooding (backoff strategy)

} else {

currentState = READ;

}

break;

case HANDLE\_ERROR:

// Handle errors and define next state based on error type

handleErrors();

currentState = IDLE;

break;

}

osDelay(10); // Reduce CPU usage and allow other tasks to run

}

}

// communication task

Already have.

//monitoring task

void MonitoringTask(void \*argument) {

while (1) {

bool memStatus = checkMemoryStatus();

bool sensorStatus = checkSensorConnectivity();

bool networkStatus = checkNetworkStatus();

// Log status or take action if any check fails

if (!memStatus || !sensorStatus || !networkStatus) {

logSystemHealth("Warning: System health check failure detected");

// Taking corrective actions

if (!memStatus) {

logSystemHealth("Error: Memory status critical");

// Additional memory handling code here

}

if (!sensorStatus) {

logSystemHealth("Error: Sensor connectivity issue");

// Attempt to reinitialize sensor

}

if (!networkStatus) {

logSystemHealth("Error: Network connectivity issue");

// Reconnect network or handle communication task

}

} else {

// Optionally log that everything is fine

logSystemHealth("System health is good");

}

// Task delay to reduce CPU usage, delay period depends on system requirements

osDelay(1000); // Checks every 1000 milliseconds (1 second)

}

}

// Simulated function to check memory status

bool checkMemoryStatus() {

// Example check, replace with actual memory check

return true;

}

// Simulated function to check sensor connectivity

bool checkSensorConnectivity() {

// Example check, replace with actual sensor status check

return true;

}

// Simulated function to check network status

bool checkNetworkStatus() {

// Example check, replace with actual network check

return true;

}

// Function to log system health status

void logSystemHealth(const char\* message) {

printf("%s\n", message);

}