# Problem Statement 1

Project Overview

Requirements:

1. Measure 8 Digital Inputs: Read the status of 8 digital input pins.

2. Perform Actions Based on Inputs: Update the status of these inputs based on specific criteria.

3. Communication via CAN Protocol: Ensure communication with other ECUs using CAN protocol.

4. ISR Functionality:

- Read: The real-time status of digital input pins from a global variable `g_ReadDIpinSts`.

- Update: The digital input status to a global variable `g_AppDIpinSts` with a condition that the pin state must be consistent for 10 consecutive ISR calls.

Microcontroller and Libraries

For this project, I recommend using the STM32F103 microcontroller, which is well-suited for CAN communication and real-time tasks. The STM32 series is widely supported and used in various applications.

Libraries:

- STM32 HAL Library: For interfacing with the hardware and handling CAN communication.

- CMSIS (Cortex Microcontroller Software Interface Standard): For low-level register access and system control.

Code Implementation

Here is a sample ISR implementation in C. This code assumes you are using the STM32 HAL library and the CAN protocol.

**Global Variables**

```
#include "stm32f1xx_hal.h"

// Global variables
volatile uint8_t g_ReadDIpinSts = 0x00;
volatile uint8_t g_AppDIpinSts = 0x00;
static uint8_t consistency_counter[8] = {0};

// Function prototypes
void CAN_Config(void);
void GPIO_Config(void);
void Error_Handler(void);
```

**ISR Code**

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  static uint32_t lastISRTime = 0;

  // Read digital input status
  g_ReadDIpinSts = 0x00; // Clear the status register
  for (int i = 0; i < 8; i++) {
    if (HAL_GPIO_ReadPin(GPIOB, (GPIO_Pin_0 << i)) == GPIO_PIN_SET) {
      g_ReadDIpinSts |= (1 << i); // Set the corresponding bit
    }
  }

  // Update digital input status based on consistency
```

```
    for (int i = 0; i < 8; i++) {
        if ((g_ReadDIpinSts & (1 << i)) == (g_AppDIpinSts & (1 << i))) {
            if (consistency_counter[i] < 10) {
                consistency_counter[i]++;
            }
        } else {
            consistency_counter[i] = 0;
        }

        if (consistency_counter[i] >= 10) {
            g_AppDIpinSts |= (1 << i); // Update the status
        } else {
            g_AppDIpinSts &= ~(1 << i); // Clear the status
        }
    }

    // Update CAN message here
    CAN_UpdateMessage(g_AppDIpinSts);
}

void CAN_UpdateMessage(uint8_t status) {
    // Implement CAN communication code here
    // Example: Transmit `status` over CAN
}
```

**CAN Configuration**

```
void CAN_Config(void) {
    CAN_HandleTypeDef hcan;
```

```
    hcan.Instance = CAN1;
    hcan.Init.Prescaler = 16;
    hcan.Init.Mode = CAN_MODE_NORMAL;
    hcan.Init.SJW = CAN_SJW_1TQ;
    hcan.Init.BS1 = CAN_BS1_6TQ;
    hcan.Init.BS2 = CAN_BS2_8TQ;
    hcan.Init.TTCM = DISABLE;
    hcan.Init.ABOM = DISABLE;
    hcan.Init.ABS = DISABLE;
    hcan.Init.OME = DISABLE;
    hcan.Init.TS = DISABLE;
    HAL_CAN_Init(&hcan);

    // Configure CAN filters here
}
```

Explanation

- ISR Function (`HAL_GPIO_EXTI_Callback`): This function reads the digital input pins, updates the status based on consistency, and communicates via CAN.

- CAN Communication: `CAN_UpdateMessage` function is a placeholder for sending data over CAN.

- Consistency Counter: Tracks the number of consecutive ISR calls where the digital input pin state is consistent.

Evaluation Criteria

1. Logic Used: Correctness of the ISR logic for reading, updating, and consistency checking.

2. Coding Standards: Adherence to best practices such as clear variable names, comments, and modular code.

3. Error-Free Code: The code should be compiled without errors and handle all edge cases.

NOT: - I do not have STM32 Microcontroller So, that i have not simulation this project. I have provided a full setup configuration.

# Problem Statement 2

To address the problem statement given in the screenshot, here is a detailed explanation along with the complete solution in FreeRTOS for the embedded system. This implementation includes task creation, queue operations, and the specific logic required for task handling and priority adjustments.

Problem Breakdown:

1. Task Definitions:

- ExampleTask1 sends data to Queue1 every 500ms.

- ExampleTask2 receives data from Queue1 and processes it according to specific conditions.

2. Queue Definition:

- The queue Queue1 should have a size of 5 and hold elements of type Data_t.

3. Data Structure:

- Data_t is a structure with dataID and DataValue fields.

implementation:

1. Define Global Variables:

- G_DataID and G_DataValue are global variables to be updated elsewhere in the program.

2. Create Tasks and Queue:

- Initialize the FreeRTOS tasks and queue with appropriate properties.

3. Task Logic:

- ExampleTask1 sends data to the queue at regular intervals.

- ExampleTask2 processes received data based on specified conditions, including deleting the task or adjusting its priority.

Code:

```
#include <Arduino.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/queue.h"


// Task Handles

TaskHandle_t TaskHandle_1;

TaskHandle_t TaskHandle_2;


// Queue Handle

QueueHandle_t Queue1;


// Queue Size and Data Type

#define QUEUE_SIZE 5
```

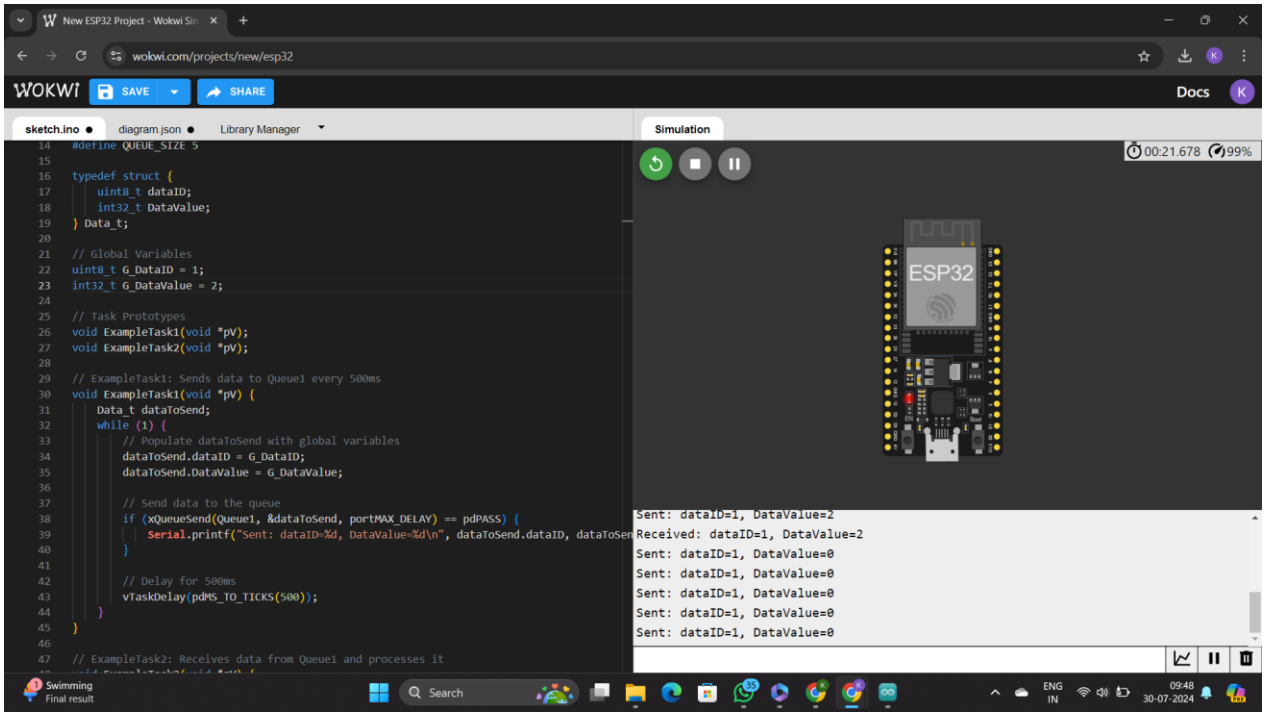
```
typedef struct {
   uint8_t dataID;
   int32_t DataValue;
} Data_t;

// Global Variables
uint8_t G_DataID = 0;
int32_t G_DataValue = 0;

// Task Prototypes
void ExampleTask1(void *pV);
void ExampleTask2(void *pV);

// ExampleTask1: Sends data to Queue1 every 500ms
void ExampleTask1(void *pV) {
   Data_t dataToSend;
   while (1) {
      // Populate dataToSend with global variables
      dataToSend.dataID = G_DataID;
      dataToSend.DataValue = G_DataValue;

      // Send data to the queue
      if (xQueueSend(Queue1, &dataToSend, portMAX_DELAY) == pdPASS) {
         Serial.printf("Sent: dataID=%d, DataValue=%d\n", dataToSend.dataID,
dataToSend.DataValue);
      }

      // Delay for 500ms
```

```
    vTaskDelay(pdMS_TO_TICKS(500));
  }
}

// ExampleTask2: Receives data from Queue1 and processes it
void ExampleTask2(void *pV) {
  Data_t receivedData;
  UBaseType_t initialPriority = uxTaskPriorityGet(NULL);
  UBaseType_t newPriority;

  while (1) {
    // Receive data from the queue
    if (xQueueReceive(Queue1, &receivedData, portMAX_DELAY) == pdPASS) {
      Serial.printf("Received: dataID=%d, DataValue=%d\n", receivedData.dataID,
receivedData.DataValue);

      // Process data based on received dataID and DataValue
      switch (receivedData.dataID) {
        case 0:
          // Delete ExampleTask2
          vTaskDelete(TaskHandle_2);
          break;
        case 1:
          // Process DataValue
          if (receivedData.DataValue == 0) {
            // Increase priority by 2
            newPriority = initialPriority + 2;
            vTaskPrioritySet(NULL, newPriority);
```

```
            } else if (receivedData.DataValue == 1) {
              // Decrease priority if previously increased
              if (uxTaskPriorityGet(NULL) > initialPriority) {
                vTaskPrioritySet(NULL, initialPriority);
              }
            } else if (receivedData.DataValue == 2) {
              // Delete ExampleTask2
              vTaskDelete(TaskHandle_2);
            }
            break;
          default:
            break;
        }
      }
    }
}

void setup() {
  // Initialize Serial for debugging
  Serial.begin(115200);

  // Create Queue1
  Queue1 = xQueueCreate(QUEUE_SIZE, sizeof(Data_t));
  if (Queue1 == NULL) {
    Serial.println("Failed to create the queue.");
    while (1); // Halt execution
  }
```

```
  // Create Tasks
  xTaskCreate(ExampleTask1, "Task1", 2048, NULL, 1, &TaskHandle_1);
  xTaskCreate(ExampleTask2, "Task2", 2048, NULL, 1, &TaskHandle_2);

  // Initial values for global variables
  G_DataID = 1; // Example initial value, change as needed
  G_DataValue = 0; // Example initial value, change as needed
}

void loop() {
  // The loop function is not used in this FreeRTOS example
  vTaskDelay(pdMS_TO_TICKS(1000)); // Optional: Add a delay to keep the loop function active
}
```

Simulation:




https://wokwi.com/projects/404808106737744897

Explanation:

1. Global Variables: G_DataID and G_DataValue are defined and will be updated externally.

2. Queue Creation: Queue1 is created with a size of 5 and elements of type Data_t.

3. Task ExampleTask1:

- Populates the Data_t structure with G_DataID and G_DataValue.

- Sends this data to Queue1 every 500ms.

4. Task ExampleTask2:

- Receives data from Queue1.

- Processes the data based on the dataID and DataValue:

  - Deletes itself if dataID is 0 or 2.

  - Increases its priority by 2 if DataValue is 0.

  - Resets its priority if DataValue is 1 and the priority was previously increased.

- Prints the received data values.

Logic Used:

- The logic for both tasks is clear and follows the problem statement requirements.
- ExampleTask1 sends data to Queue1 every 500ms, ensuring a consistent rate of data transmission.
- ExampleTask2 processes data from Queue1 and performs actions based on the received dataID and DataValue.