****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**19CSE303 – EMBEDDED SYSTEMS**

**BATCH: 2022-2026**

**TOPIC:**

|  |  |  |
| --- | --- | --- |
| **01** | **ARJUN SHIVKUMAR** | **CB.EN.U4CSE22304** |
| **02** | **C B ABIJITH KAARTHIGEYAN** | **CB.EN.U4CSE22314** |
| **03** | **KARTHIKEYAN** | **CB.EN.U4CSE22323** |
| **04** | **KISHORE KUTTALAM** | **CB.EN.U4CSE22326** |

**Automated Pet Feeder**

**Objective**: The primary goal of this project is to create an automated pet feeder system that can:

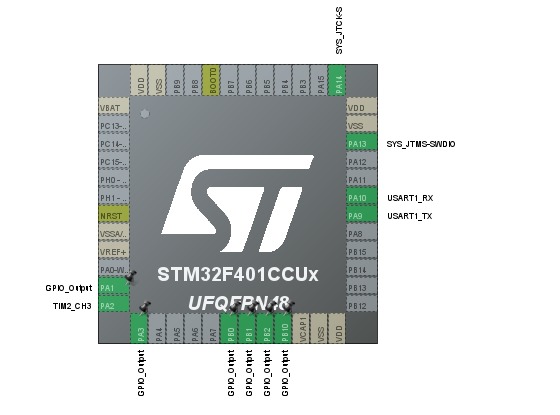
1)Monitor the food level in the pet’s feeding tray.

2)Detect and control the feeder door status (open or closed).

3)Send notifications via Bluetooth to a mobile device, updating the user on the food level and door status.

This setup is ideal for pet owners who want a remote, real-time way to ensure their pet is consistently fed. By automatically monitoring the feeder, the system can reduce the need for frequent manual checks and provide peace of mind, especially for pet owners who may not always be nearby.

**Key Components:**

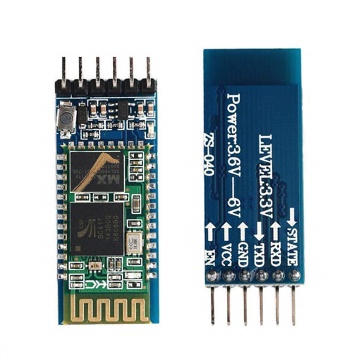
1. STM32F401 Microcontroller
2. Ultrasonic Sensor

****

1. Force Sensitive Resistor (FSR)



1. HC-05 Bluetooth Module



1. LED Indicators
2. Relay Module



1. Motor (optional for food dispensing)
2. Battery (for motor power)
3. Capacitors (0.1 µF and 100 µF)
4. Diode

**System Overview:**

**1)Food Level Monitoring with Force Sensitive Resistor (FSR):**

The FSR sensor detects the weight or pressure applied by the food in the tray.

Using the analog reading from the FSR, the system categorizes the food level as:

**i)**Sufficient: When the food level is above a specified threshold, indicating enough food is available.

**ii)**Optimum: When the food is within an ideal range.

**iii)**Low: When the food level drops below a certain threshold, indicating an empty tray.

Depending on the food level, the system sends appropriate messages like "**Food quantity greater than 2000**," "**The food is optimum**," or "**No food in tray**," notifying the user via Bluetooth.

**2)Feeder Door Detection with Ultrasonic Sensor:**

An ultrasonic sensor measures the distance to check if the door is open or closed.

When the door is open (detected if the distance is less than a specified threshold), an LED indicator is turned on, and the message "**door open**" is sent to the user.

When the door closes (distance above the threshold), the LED is turned off, and "**door closed**" is sent.

This system ensures that the pet owner is aware of whether the feeder door is accessible to the pet, improving reliability and security for feeding.

**3)Bluetooth Communication for Remote Monitoring:**

Through the HC-05 Bluetooth module, the STM32F401 microcontroller sends updates directly to the user’s device.

This setup allows the user to receive real-time updates about the feeder status from a distance.

Motor Control with Relay for Food Dispensing:

A motor controlled via a relay module can be integrated to dispense food when triggered, potentially based on the detected food level.

A battery powers the motor independently to ensure reliable operation without affecting the STM32’s performance.

**CODE STRUCTURE:**

1. **Global Variables**: Variables for the ultrasonic sensor, FSR, and door states are declared here for access across functions.
2. **Function Prototypes**: Declares all functions used, allowing the compiler to know about them before they’re defined.
3. **Main Function**:
   * Initializes GPIO, interrupts, timers, ADC, and USART for various modules.
   * Contains the main loop that:
     + Uses the ultrasonic sensor to measure distance and controls the door state accordingly.
     + Reads FSR data through ADC, checks food quantity, and sends relevant messages based on thresholds.
   * Implements a delay between sensor readings to manage the processing loop rate.
4. **Peripheral Configuration Functions**:
   * GPIO\_Config: Sets up GPIO pins for ultrasonic sensor, LEDs, and FSR.
   * interrupt\_config: Configures external interrupts for the ECHO pin.
   * delay\_us: Provides microsecond delays for accurate timing in the ultrasonic sensor.
5. **Sensor-Specific Functions**:
   * echo\_generate: Generates a pulse to trigger the ultrasonic sensor.
   * EXTI1\_IRQHandler: Interrupt handler to calculate pulse duration based on the echo signal.
   * TIM2\_Config: Configures a timer for capturing echo pulse width.
6. **FSR and ADC Functions**:
   * ADC\_Init: Initializes ADC for reading FSR values.
   * ADC\_Read: Starts an ADC conversion and returns the FSR value.
7. **USART2 Configuration**:
   * USART2\_Config: Sets up USART2 to communicate messages over Bluetooth.
8. **Delay Function**:
   * delay: A simple delay loop for debouncing and pacing sensor readings.

**Challenges Faced and Solutions**

1. **Challenge: Accurate Food Level Detection**
   * **Issue**: The Force Sensitive Resistor (FSR) was sensitive to minor variations in pressure, which sometimes led to inaccurate readings of food levels.
   * **Solution**: To address this, the code was adjusted to include threshold ranges, ensuring that only significant changes in food weight would trigger a message. Filtering techniques, such as averaging multiple readings, were also considered to improve accuracy.
2. **Challenge: Reliable Door Status Detection**
   * **Issue**: The ultrasonic sensor occasionally gave inconsistent readings, causing unreliable door status detection. Ambient conditions like reflections or nearby obstacles impacted sensor performance.
   * **Solution**: By implementing a fixed threshold for detecting door states and integrating edge detection in the interrupt handler, sensor readings became more consistent. Additional filtering logic was considered to further stabilize readings.
3. **Challenge: Bluetooth Communication Stability**
   * **Issue**: Maintaining a stable Bluetooth connection was challenging, especially when operating at a greater distance or in the presence of interference from other devices.
   * **Solution**: Adjustments were made to optimize transmission intervals and message structure, reducing potential data packet loss. Testing under various conditions helped determine the optimal operating range.
4. **Challenge: Power Management for Continuous Operation**
   * **Issue**: The components, especially the motor (for potential food dispensing), required substantial power, leading to potential system instability or the need for frequent recharging.
   * **Solution**: A battery backup system was considered to ensure continuous operation during power outages. Additionally, separate power sources for high-drain components like the motor were implemented to avoid overloading the main microcontroller.
5. **Challenge: Delays Due to Debouncing and Sensor Delays**
   * **Issue**: The need for debouncing (to filter out false signals) and accurate timing created slight delays that affected the system’s responsiveness.
   * **Solution**: Efficient coding practices, such as using precise timer-based delays and leveraging the microcontroller’s capabilities for handling multiple tasks, were used to minimize the impact of debouncing on system responsiveness.
6. **Challenge: Size Constraints for Compact Design**
   * **Issue**: Fitting all necessary components into a compact, pet-friendly enclosure without compromising functionality proved challenging.
   * **Solution**: Compact, low-profile components were selected wherever possible, and careful arrangement was made to optimize space without compromising on ease of access for maintenance and adjustments.

**MAIN CODE:**

#include "stm32f4xx.h" // Device header

// Ultrasonic sensor variables

void GPIO\_Config(void);

void delay\_us(uint32\_t us);

void interrupt\_config(void);

void echo\_generate(void);

void TIM2\_Config(void);

void USART2\_Config(void);

int distance\_cm;

volatile uint32\_t start\_time = 0;

volatile uint32\_t end\_time = 0;

// FSR variables

uint16\_t adc\_value; // Variable to store ADC value for FSR

int food\_message\_sent = 0; // Flag to track if the "Food quantity greater than 2000" message was sent

int no\_food\_message\_sent = 0; // Flag to track if the "No food in tray" message was sent

int optimum\_food\_message\_sent = 0;

void ADC\_Init(void);

uint16\_t ADC\_Read(void);

void delay(volatile uint32\_t delay); // Delay function for FSR

int prev\_adc\_nonzero = 1; // Track previous ADC state

const int FOOD\_THRESHOLD\_HIGH = 2000;

const int FOOD\_THRESHOLD\_LOW = 1;

// Door state variables

int door\_state = 0; // 0 = closed, 1 = open

int door\_message\_sent = 0; // Flag to track if the door message was sent

int main(void) {

GPIO\_Config(); // Initialize GPIO pins for TRIG, ECHO, and LEDs

interrupt\_config(); // Configuring an interrupt for the ECHO pin

TIM2\_Config(); // Configuring timer for ultrasonic sensor

ADC\_Init(); // Initialize ADC for FSR

USART2\_Config(); // Initialize USART2 for Bluetooth communication

while (1) {

// Ultrasonic sensor operations

echo\_generate(); // Generate echo for ultrasonic sensor

// Check distance and update door state

if (distance\_cm < 150) {

GPIOA->ODR |= (1U << 10); // Turn on LED connected to PA10

delay(1000);

if (door\_state == 0) { // If the door is currently closed

door\_state = 1; // Update state to open

door\_message\_sent = 0; // Reset door message sent flag

}

if (!door\_message\_sent) {

// Send "door open" message

char message[] = "door open\r\n";

for (int i = 0; message[i] != '\0'; i++) {

while (!(USART2->SR & (1U << 7))) {}; // Wait until TXE (Transmit Data Register Empty)

USART2->DR = message[i]; // Send the current character

}

door\_message\_sent = 1; // Set the message flag

}

} else {

GPIOA->ODR &= ~(1U << 10); // Turn off LED connected to PA10

if (door\_state == 1) { // If the door is currently open

door\_state = 0; // Update state to closed

door\_message\_sent = 0; // Reset door message sent flag

}

if (!door\_message\_sent) {

// Send "door closed" message

char message[] = "door closed\r\n";

for (int i = 0; message[i] != '\0'; i++) {

while (!(USART2->SR & (1U << 7))) {}; // Wait until TXE (Transmit Data Register Empty)

USART2->DR = message[i]; // Send the current character

}

door\_message\_sent = 1; // Set the message flag

}

}

// FSR operations

adc\_value = ADC\_Read(); // Read ADC value from FSR

if (adc\_value > 2000 && food\_message\_sent == 0) {

GPIOA->ODR &= ~(1U << 8);

char message[] = "Food quantity greater than 2000\r\n";

for (int i = 0; message[i] != '\0'; i++) {

while (!(USART2->SR & (1U << 7))) {};

USART2->DR = message[i];

}

food\_message\_sent = 1;

no\_food\_message\_sent = 0;

optimum\_food\_message\_sent = 0; // Reset optimum message flag

prev\_adc\_nonzero = 1;

}

else if (adc\_value <500 && prev\_adc\_nonzero == 1) {

GPIOA->ODR |= (1U << 8);

char message[] = "No food in tray\r\n";

for (int i = 0; message[i] != '\0'; i++) {

while (!(USART2->SR & (1U << 7))) {};

USART2->DR = message[i];

}

no\_food\_message\_sent = 1;

food\_message\_sent = 0;

optimum\_food\_message\_sent = 0; // Reset optimum message flag

prev\_adc\_nonzero = 0; // Update previous value tracker to zero

}

else if (adc\_value >=500 && adc\_value <= 2000) {

// Check for optimum food quantity

GPIOA->ODR &= ~(1U << 8);

if (!optimum\_food\_message\_sent) {

char message[] = "The food is optimum\r\n";

for (int i = 0; message[i] != '\0'; i++) {

while (!(USART2->SR & (1U << 7))) {};

USART2->DR = message[i];

}

optimum\_food\_message\_sent = 1; // Set the optimum message flag

}

food\_message\_sent = 0;

no\_food\_message\_sent = 0;

prev\_adc\_nonzero = 1;

}

delay(10000);

}

}

// Function to configure GPIO pins

void GPIO\_Config(void) {

// Enable GPIOA and GPIOB clocks

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN | RCC\_AHB1ENR\_GPIOBEN;

// Set PB0 (TRIG) as output

GPIOB->MODER |= (1 << 0); // Set PB0 as output

GPIOB->OTYPER &= ~(1 << 0); // Output push-pull

GPIOB->OSPEEDR |= (3 << 0); // Set high speed

// Set PB1 (ECHO) as input

GPIOB->MODER &= ~(3 << 2); // Set PB1 as input

// Set PA4, PA5, PA6, PA10 as output for LEDs

GPIOA->MODER |= (1 << 8) | (1 << 10) | (1 << 12) | (1 << 20)| (1 << 16);

// Set PA1 as analog mode for FSR

GPIOA->MODER |= (3UL << (1 \* 2)); // Set PA1 as analog mode (for ADC input)

}

// Configuring the interrupt pin PB1

void interrupt\_config() {

RCC->APB2ENR |= (1U << 14); // Enable system configuration controller clock

// Map the interrupt to PB1

SYSCFG->EXTICR[0] = 0; // Clear bits for safety

SYSCFG->EXTICR[0] |= (1U << 4); // Map EXTI1 to PB1

EXTI->IMR |= (1U << 1); // Unmask the interrupt at PB1

EXTI->FTSR |= (1U << 1); // Select falling edge trigger for PB1

EXTI->RTSR |= (1U << 1); // Select rising edge trigger for PB1

NVIC\_EnableIRQ(EXTI1\_IRQn); // Enable the NVIC for EXTI1

}

// Function to create a delay in microseconds using SysTick

void delay\_us(uint32\_t us) {

volatile uint32\_t count = 84 \* us; // 1 microsecond delay at 84 MHz

while (count--) {

\_\_NOP(); // No operation (dummy instruction)

}

}

// Function to measure distance using the ultrasonic sensor

void echo\_generate(void) {

// Send a 10us pulse to TRIG pin to start ultrasonic burst

GPIOB->ODR |= (1 << 0); // Set PB0 HIGH

delay\_us(10); // Wait for 10 microseconds

GPIOB->ODR &= ~(1 << 0); // Set PB0 LOW

}

void EXTI1\_IRQHandler(void) {

if (EXTI->PR & (1U << 1)) { // Check if interrupt was caused by PB1

EXTI->PR |= (1U << 1); // Clear the pending interrupt flag

if (GPIOB->IDR & (1 << 1)) { // Rising edge detected (echo pulse started)

TIM2->CNT = 0; // Reset TIM2 counter

TIM2->CR1 |= TIM\_CR1\_CEN; // Start TIM2

} else { // Falling edge detected (echo pulse ended)

TIM2->CR1 &= ~TIM\_CR1\_CEN; // Stop TIM2

end\_time = TIM2->CNT; // Capture end time (pulse duration in microseconds)

// Convert pulse duration to distance in cm

distance\_cm = (end\_time \* 0.0343) / 2; // Speed of sound = 0.0343 cm/us, divide by 2 for round trip

}

}

}

// Setting the timer for ultrasonic sensor

void TIM2\_Config(void) {

RCC->APB1ENR |= RCC\_APB1ENR\_TIM2EN; // Enable TIM2 clock

TIM2->PSC = 84 - 1; // Prescaler to slow down the clock (84 MHz / 84 = 1 MHz)

TIM2->ARR = 0xFFFFFFFF; // Set auto-reload to maximum (32-bit timer)

TIM2->CNT = 0; // Reset counter

TIM2->CR1 |= TIM\_CR1\_URS; // Update request source

}

// ADC initialization for FSR

void ADC\_Init(void) {

// Enable GPIOA and ADC1 clocks

RCC->AHB1ENR |= RCC\_AHB1ENR\_GPIOAEN; // Enable GPIOA clock

RCC->APB2ENR |= RCC\_APB2ENR\_ADC1EN; // Enable ADC1 clock

// ADC configuration

ADC1->CR2 |= (1U << 1); // Start with ADC disabled (clear ADON bit)

ADC1->SQR3 = 1; // Select channel 1 for ADC conversion

ADC1->SMPR2 |= (7 << 3); // Set sample time for channel 1 to 480 cycles (maximum)

ADC1->CR2 |= ADC\_CR2\_ADON; // Enable the ADC

}

// Function to read ADC value for FSR

uint16\_t ADC\_Read(void) {

ADC1->CR2 |= ADC\_CR2\_SWSTART; // Start the ADC conversion

while (!(ADC1->SR & ADC\_SR\_EOC)); // Wait until conversion is complete

return ADC1->DR; // Read the ADC conversion result

}

// USART2 configuration for Bluetooth module

void USART2\_Config(void) {

RCC->AHB1ENR |= (1 << 0); // Enable GPIOA clock

GPIOA->MODER |= (1 << 5); // Set PA2 to alternate function mode for USART2 Tx

GPIOA->AFR[0] |= (7 << 8); // Set AF7 for USART2 on PA2

RCC->APB1ENR |= (1 << 17); // Enable USART2 clock

USART2->BRR = 0x683; // Set baud rate to 115200 @ 16 MHz

USART2->CR1 |= (1 << 3); // Enable transmitter

USART2->CR1 |= (1 << 13); // Enable USART2

}

// Simple delay for debounce

void delay(volatile uint32\_t delay) {

while(delay--);

}

**Conclusion**

The automated pet feeder project achieves its goal of providing pet owners with a reliable, remote-controlled system for monitoring and managing their pet’s food supply. By integrating sensors for food level detection and door status, alongside Bluetooth communication, the system ensures real-time updates for enhanced convenience and peace of mind. The design leverages the STM32 microcontroller’s capabilities, creating a modular, scalable structure that can be easily extended for future enhancements, such as automated food dispensing and advanced notifications. This project is a practical solution for pet care, bridging technology and convenience for pet owners.

Future improvements could include:

1. **Real-Time Clock (RTC) Module**: Incorporate an RTC module for precise scheduling, so the feeder can dispense food at set intervals throughout the day, which is ideal for managing feeding routines for pets.
2. **Weight-Based Food Measurement**: Integrate a load cell with an ADC to measure the exact weight of the food in the tray, providing more accurate feedback on food quantity instead of relying solely on the FSR.
3. **Voice Alerts and Commands**: Add a speaker or sound module to provide voice alerts for food dispensing, or allow voice commands to control the feeder if used with a compatible smart assistant.
4. **Battery Backup**: Include a battery backup to ensure the system continues functioning during power outages, and send notifications to the user if the power source switches to battery.
5. **Enhanced User Interface**: Add an LCD screen or LEDs to display real-time data like food level, feeder status, and error messages directly on the device, making it user-friendly even without a mobile app.
6. **Multiple Pet Profiles**: Allow for different feeding profiles if there are multiple pets, enabling different schedules or food quantities for each pet based on their needs.
7. **Self-Cleaning Mechanism**: Implement a cleaning function that can clear out any leftover or spoiled food in the tray, ensuring the pet always has access to fresh food.