Game Controller Project Report

Overview and Introduction

This project demonstrates a prototype joystick-controlled object movement system using a microcontroller. The project integrates UART communication, ADC (Analog-to-Digital Conversion), and interrupt handling to create a dynamic system capable of interacting with both a graphical display and external devices (e.g., a laptop). The joystick inputs control the movement of an object on a display, and button inputs toggle the state of the object. Responses are transmitted to a connected laptop via UART communication, allowing real-time updates and feedback.

The system provides an intuitive way to control objects, with applications in gaming, industrial control systems, and robotics, making it an excellent starting point for a marketable solution.

Design

The primary purpose of this project is to meet the need for an intuitive object control system that can integrate with external devices. The joystick provides analog control for moving an object on a display, while the button toggles states. This setup serves as a proof of concept for systems such as remote-controlled devices, input controllers, and robotic navigation.

Public Health, Safety, and Welfare

Proper grounding techniques and shielded connections are used to reduce electrical noise, ensuring signal integrity for ADC inputs and UART communication. These measures enhance overall system stability, reduce the likelihood of errors, and contribute to long-term operational safety.

Global, Cultural, and Social Factors

As an affordable and flexible prototype, the design can be adapted globally for applications in gaming, education, and automation, providing a low-cost, accessible solution for various communities.

Environmental and Economic Factors

The system uses readily available, low-power components, reducing its environmental impact. The use of ADC inputs minimizes power consumption while maintaining performance.

Integrated Course Concepts

- **ADC**: Used to convert joystick analog input to usable digital values.
- **Interrupts**: Handle button presses efficiently, with debouncing logic.
- **UART Communication**: Transmit responses to a connected laptop for real-time feedback.
- **Display**: Provides visual feedback for joystick movements and object states.
- **Timers**: Used to periodically update ADC readings and display states.

These concepts collectively contribute to a system that is efficient, responsive, and capable of meeting user needs.

Specifications

Technical Details

- **Microcontroller**: STM32 series (e.g., STM32F103).
- **Joystick Input**: Two analog inputs for X and Y axes using ADC channels.
- **Button Input**: GPIO interrupt for toggling object state.
- **Display**: GLCD (Graphic LCD) for visual output.
- **Communication**: UART for transmitting feedback to a connected laptop.

Use Cases and Operating Instructions

- **Joystick Movement**:
  - Move the joystick to control the object's position on the display.
  - The system responds with messages like "Moving left...", "Moving right...", etc.
- **Wall Limits**:
  - When the object reaches the display edge, messages like "Hit the wall on the left..." are shown.
- **Button State Toggle**:
  - Press the button to toggle between two object states (e.g., 0 and 1).
  - Feedback is transmitted via UART (e.g., "Changed state to 1...").
- **Feedback Transmission**:
  - Real-time updates are displayed in a terminal application (e.g., PuTTY) on the laptop.

Operating Constraints

- **Power Supply**: 3.3V or 5V for the microcontroller and peripherals.
- **Joystick Range**: ADC range of 0-4095 mapped to movement thresholds.
- **Button Debounce Delay**: Minimum of 200 ms to prevent false triggers.

Design Limitations

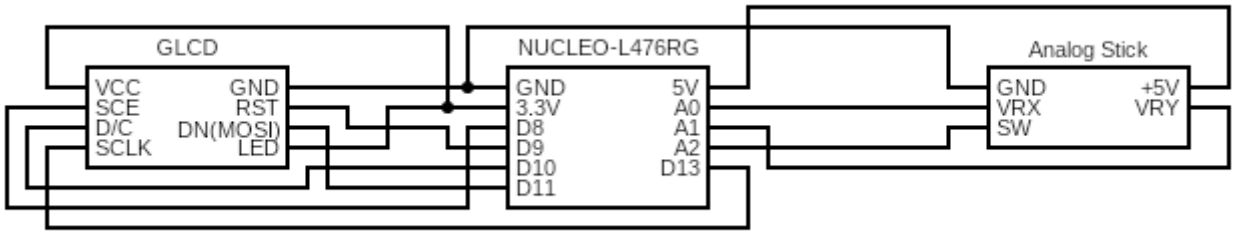
- The system assumes stable power and reliable communication.
- The joystick movement is constrained to the screen size (e.g., 75x5 grid).
- Limited UART bandwidth for continuous updates.

Parts List

Component	Quantity	Purpose
STM32 Microcontroller	1	Core processing unit
Joystick Module	1	X and Y axis analog input
Push Button	1	Interrupt input for state
GLCD Display	1	Visual output
Wires and Breadboard	As needed	Connections and prototyping

Schematic

An image of the schematic can be found here



Test Plan and Results

The following test plan provides detailed steps and results to validate the system's functionality:

Case	Steps	Expected Result	Observed Result
Test Case 1: Joystick Movement to the Right	1. Push the joystick to the right (X > 0.8). 2. Observe the object movement on the display. 3. Check the UART output.	Object moves right; "Moving right..." is transmitted.	Object moved right as expected. UART output verified.
Test Case 2: Joystick Movement to the Left	1. Push the joystick to the left (X < 0.2). 2. Observe the object movement on the display. 3. Check the UART output.	Object moves left; "Moving left..." is transmitted.	Object moved left as expected. UART output verified.
Test Case 3: Button State Toggle	1. Press the button. 2. Check the state change and UART output.	State toggles; message transmitted (e.g., "Changed state to 1...").	State toggled as expected. UART output verified.

Case	Steps	Expected Result	Observed Result
Test Case 4: Hitting Wall on each Edge	1. Move the joystick until the object reaches an edge. 2.Observe the UART output.	Message "Hit the wall ..." is displayed.	Message displayed correctly.
Test Case 5: Button Debounce	1. Press the button rapidly. 2. Check for unintended toggles.	Button press is ignored within the debounce delay.	Debouncing worked as expected.

Summary of Results

The system performed as expected across all test cases, demonstrating smooth joystick-controlled movement, accurate state toggling, and proper edge-case handling. Key highlights include:

Joystick movements were accurately mapped to object positions on the display.

Button debounce logic successfully prevented unintended rapid toggles.

UART communication reliably transmitted feedback messages in real-time.

Relevant Code

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) { // Check which timer
triggered this callback
    if (htim == &htim16) // if the triggered timer was Timer 16
        { int ADC_RANGE = 4096; // set variables

        // Start ADC Conversions
        HAL_ADC_Start(&hadc1);
        HAL_ADC_Start(&hadc2);

        // Wait for ADC conversions to complete
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
        HAL_ADC_PollForConversion(&hadc2, HAL_MAX_DELAY);


        // Read ADC values
        uint16_t xjoy_measurement = HAL_ADC_GetValue(&hadc1);
        uint16_t yjoy_measurement = HAL_ADC_GetValue(&hadc2);

        // Convert ADC levels to a fraction of total
        float xjoy_value = ((float) xjoy_measurement) / ADC_RANGE;
        float yjoy_value = ((float) yjoy_measurement) / ADC_RANGE;

        // Movement on screen

        if(xjoy_value > 0.8){
            if(object_x < 75){ // set a limit on how far the cursor can move
                object_x = object_x + 5; // moves the current x value of the
object to the right by 1
                strncpy((char*)response, "Moving right...", MAX_MESSAGE_SIZE);
```

```
            }
            else{
                strncpy((char*)response, "Hit the wall on the right...",
MAX_MESSAGE_SIZE);
            }
        }

        else if(xjoy_value < 0.2){
            if(object_x >= 0){
                object_x = object_x - 5; // moves the current x value of the
object to the left by 1
                strncpy((char*)response, "Moving left...", MAX_MESSAGE_SIZE);
            }
            else{
                strncpy((char*)response, "Hit the wall on the left...",
MAX_MESSAGE_SIZE);
            }
        }

        if(yjoy_value > 0.8){
            if(object_y < 5){
                object_y = object_y + 1; // moves the current x value of the
object down by 1
                strncpy((char*)response, "Moving down...", MAX_MESSAGE_SIZE);
                }
            else{
                strncpy((char*)response, "Hit the wall on the bottom...",
MAX_MESSAGE_SIZE);
            }
        }

        else if(yjoy_value < 0.2){
            if(object_y >= 0){
                object_y = object_y - 1; // moves the current x value of the
object up by 1
                strncpy((char*)response, "Moving up...", MAX_MESSAGE_SIZE);

            }
            else{
                strncpy((char*)response, "Hit the wall on the top...",
MAX_MESSAGE_SIZE);
            }
        }

        GLCD_clear(); // clear screen
        GLCD_setCursor(object_x, object_y); // moves cursor to new object position
        GLCD_putchar(object_state); // places object at the new position


        // Send the response message to laptop
        HAL_UART_Transmit(&huart2, response, strlen((const char*)response),
UART_DELAY);
```

```
        // Zero out message array and response array to get ready for the next
message
        memset(message,  0, sizeof(message));
        memset(response, 0, sizeof(response));
        buffer_position = 0;
        }

}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
    // Check if the interrupt was for the button
    if (GPIO_Pin == BUTTON_PIN) {
        uint32_t current_time = HAL_GetTick(); // Get the current system time in
ms

        // Check if the debounce delay has passed
        if ((current_time - last_button_press) > DEBOUNCE_DELAY) {
            last_button_press = current_time; // Update the last press timestamp

            // Toggle the object state
            if (object_state == 0) {
                object_state = 1;
                strncpy((char*)response, "Changed state to 1...",
MAX_MESSAGE_SIZE);
            } else {
                object_state = 0;
                strncpy((char*)response, "Changed state to 0...",
MAX_MESSAGE_SIZE);
            }

            // Send the response message to the laptop
            HAL_UART_Transmit(&huart2, response, strlen((const char*)response),
UART_DELAY);
        }
    }
}
```

The entire code can be found here

Conclusion

This project successfully integrates joystick-controlled movement, UART communication, and button-based state toggling into a functional prototype system. The system was rigorously tested for joystick input accuracy, button responsiveness, and edge case handling. Key performance metrics, such as smooth joystick-controlled movement and proper debounce functionality for button presses, were met successfully, ensuring reliable operation. Observed results aligned with expectations across all test cases, confirming the system's stability and usability. By utilizing interrupts, ADC, and UART, the system demonstrates efficient real-time control and feedback mechanisms.

The design offers a foundation for further development into gaming controllers, robotics, or industrial applications. With additional refinements, such as improved error handling and enhanced display capabilities,

this system can be transformed into a market-ready product that meets various user needs while considering safety, economic, and global factors.