Soda Dispenser
Author: Adrian Alvarez
Team Member: Seth Bolen
May 9, 2025
ECE:3360 Embedded Systems

## 1. Introduction

The goal of this project was to develop a user-friendly soda dispenser designed for convenient home use, allowing individuals to enjoy their favorite beverage at the push of a button. A key objective was to create a seamless and intuitive user experience, ensuring that the system is both accessible and easy to operate. Additionally, the design incorporated specific usage conditions to enhance functionality and security—liquid dispensing is only enabled when a cup is detected within a designated distance beneath the nozzles, and access is restricted to authorized users through RFID verification.

## 2. Implementation

For this project, we utilized the ATmega328P microcontroller as the central unit responsible for managing all hardware interactions. It handled communication and control for the HC-SR04 ultrasonic sensor, RFID-RC522 reader, two 5V single-channel relay modules, two push button switches (PBS), and two 5V water pumps. The software was developed using an ultrasonic sensor library, an RFID-RC522 library, various custom functions, and employed both SPI and UART communication protocols to interface with peripherals.

Upon startup, the microcontroller initializes all connected hardware through the software. The ultrasonic sensor continuously monitors for an object within a predefined distance, while the RFID reader simultaneously waits for a valid tag to be scanned. Once both conditions are satisfied—presence detection and tag validation—the push buttons become active. When pressed, each PBS triggers its corresponding relay, which in turn powers the connected water pump. The hardware and software components operate cohesively to ensure a responsive and reliable system.

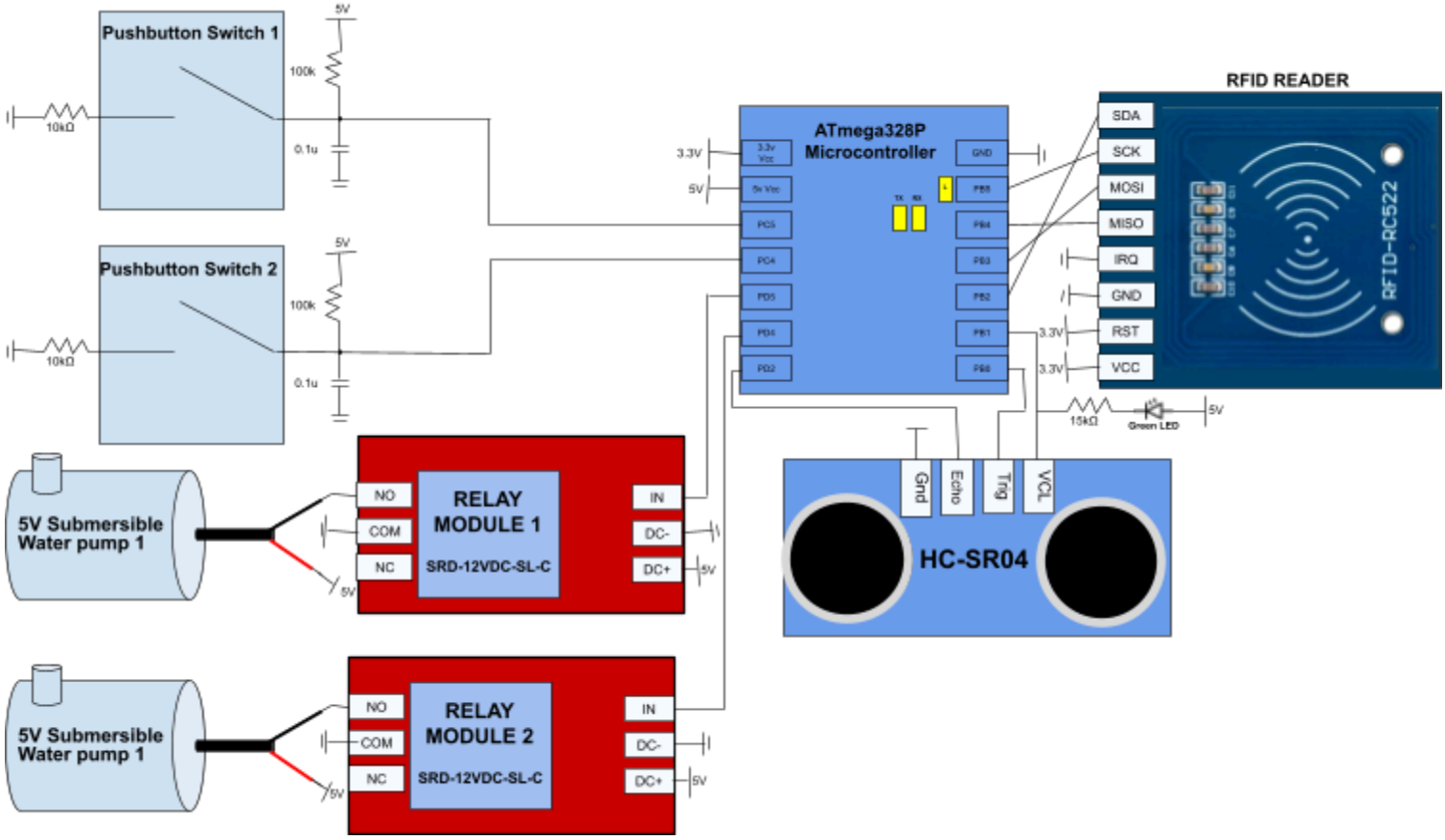



Figure 1: Implemented Circuit Schematic

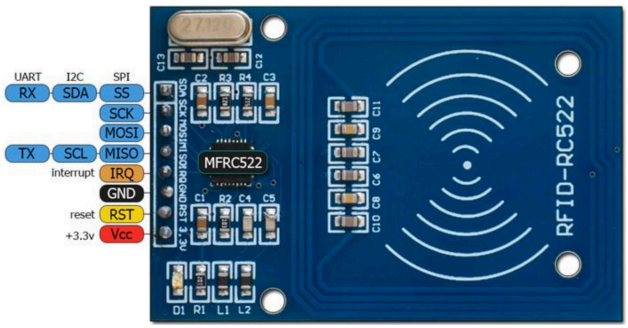



Figure 2: RFID-RC522 Pinout

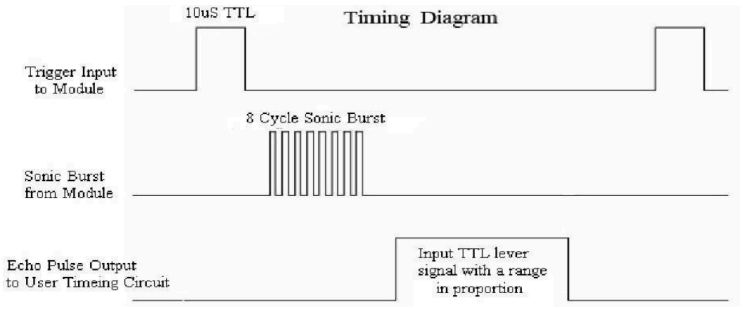
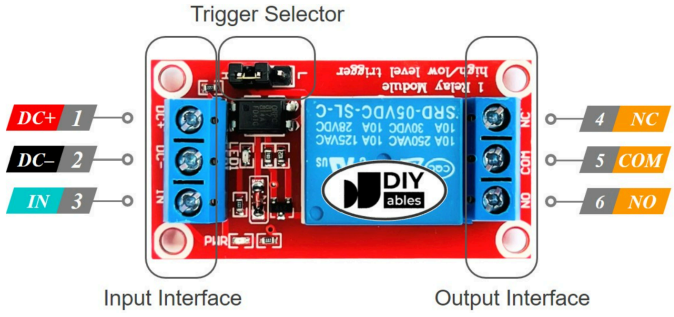



Figure 3: HC-SR04 Timing Diagram



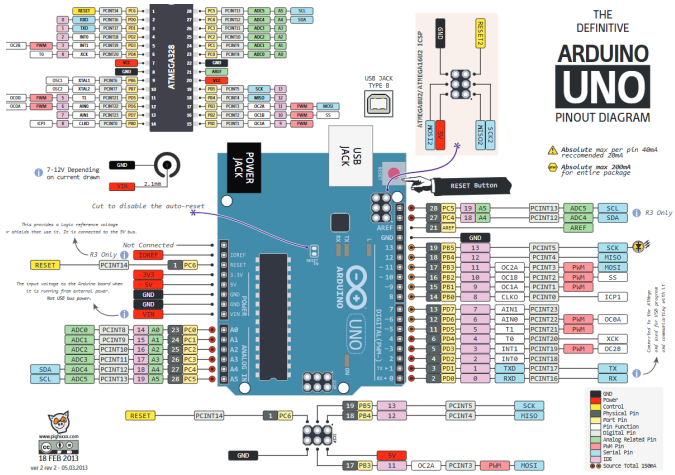



Figure 4: 5V 1-channel Relay

Figure 5: Arduino Uno Pinout Diagram

**HC-SR04 Ultrasonic Sensor**

The HC-SR04 sensor provides non-contact distance measurement by emitting an ultrasonic pulse via the TRIG pin and listening for the echo return on the ECHO pin. The time delay between sending and receiving the pulse is used to calculate distance based on the speed of sound in air. The sensor operates at 5V and requires precise pulse timing, which is handled via Ultrasonic.h. In our implementation, the sensor detects nearby objects (e.g., a hand) to initiate the authentication process.

**RFID-RC522 Reader**

The RFID-RC522 module is used to authenticate users via 13.56 MHz RFID tags. Communication between the RFID module and the microcontroller is achieved using the SPI protocol, handled via spi.h and mfrc522.h. Upon detecting a tag within its range (~3–5 cm), the reader transmits the unique identifier (UID) to the microcontroller, which compares it to a list of authorized IDs. This secure, contactless interface adds an effective access control layer to the system.

**5V 1-channel Relay**

The relay module acts as an electronically controlled switch that allows the microcontroller to activate or deactivate high-current components, such as a solenoid lock or water pump. When the relay input pin is set high (≥2V), the onboard transistor triggers the relay coil, closing the circuit between COM and NO (normally open). The relay includes an optocoupler for electrical isolation and a freewheeling diode to suppress voltage spikes from inductive loads.

**Push Button Switch (PBS) Hardware and Debouncing**

The Push Buttons were implemented with a hardware-based debouncing circuit using two 10kΩ resistors, two 0.1μF capacitors to ground, and two 100kΩ resistors to the Vcc. When buttons are pressed or released, the contacts may "bounce," causing multiple state transitions before settling to a defined state. Our debouncing circuit ensures reliable operation for both short presses and long presses.

When the button is unpressed, the 5V terminal pushes current through the 100kΩ pull-up resistor, charging the capacitor and setting the output to high. When the button is pressed, the output becomes a voltage division between 100kΩ and 10kΩ, setting the output to 0.5V, which is low. The capacitor smooths any immediate bouncing during press and release cycles. This ensures reliable operation for activating the relays. Thus reliable operation of the pumps.

### 5V Water Pump

The 5V submersible water pump operates as the output actuator, controlled via the relay module. When the relay is triggered, the pump receives 5V power and begins pumping water. It is used here as a simulated load to test the relay switching functionality and to visually indicate successful authentication. The pump's power is entirely isolated from the logic-level control signals to prevent back-current damage to the microcontroller.

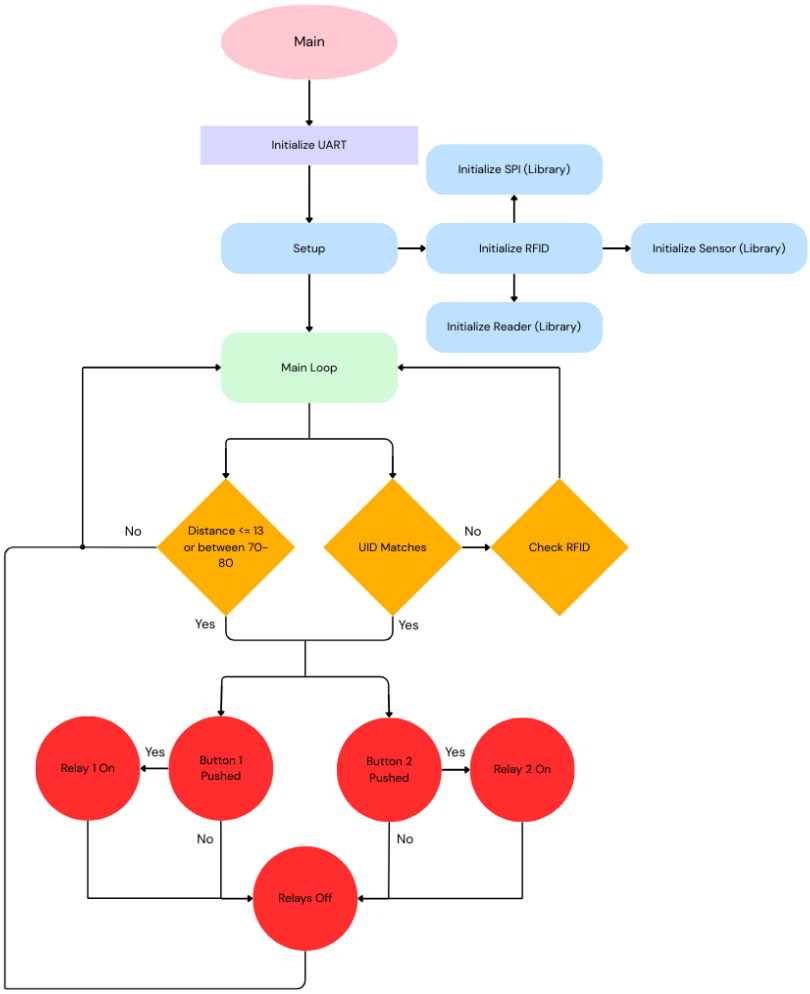



**Figure 6: Software Flow Diagram**

### Main Control Logic (Main Loop)

The main_loop() function is the core of the program. It continuously monitors both distance and RFID tag presence. If a user is detected within a valid range (either very close or in a fallback band between 70–80 cm), it initiates RFID scanning. Upon a successful UID match, the program waits for either of the two pushbuttons to be pressed. Each button triggers one of the

two relays, simulating access control or triggering an actuator. Relay states are reset at the end of each loop to ensure clean control cycles.

### Ultrasonic Sensor Library (Ultrasonic.h)

This library provides getDistance_main(), which sends a TRIG pulse and measures the duration of the ECHO signal via timer interrupts. The distance is computed from this pulse width. An interrupt service routine (ISR) handles the ECHO pin state transitions to precisely measure timing for distance calculation. The main loop uses this reading to set a distance_ok flag, which gates access to the RFID and button checks.

### RFID Communication and UID Verification (mfrc522.h / spi.h)

The RFID subsystem is initialized with init_rfid(), which sets up SPI communication and configures internal reader registers. In check_rfid(), the system scans for cards, retrieves their UID, and compares it against a hardcoded valid UID. A matching UID sets the uid_match flag, allowing access to button-triggered actions. The library manages low-level RFID transactions and UID extraction.

### Serial Communication (UART via uart_print)

The system provides real-time diagnostic messages over serial using a custom UART driver. uart_init() sets the baud rate and enables the transmitter. Functions like uart_print() and uart_print_float() format and transmit status messages, including distance measurements and RFID status, helping with debugging and verification.

Interrupt Service Routines (ISR)

1. INT0_vect (Ultrasonic Timing): Tracks the start and end of the ultrasonic ECHO pulse, updating the pulse variable with the measured timer value.

2. WDT_vect (Watchdog Timer): Ensures the microcontroller wakes up periodically even during long wait loops, using the f_wdt flag.

## 3. Experimental Methods

At the start of the project, we began by individually testing each hardware component and its corresponding software to ensure proper functionality. For the ultrasonic sensor, we used the serial monitor via UART to verify that the measured distances accurately reflected the actual distance of objects placed in front of it. The RFID reader was tested in a similar manner: we first developed software to retrieve and display the UID of any scanned RFID card using UART, and manually recorded the values. We then implemented logic to recognize a specific UID and display a message upon a successful match, confirming that the reader could differentiate valid tags.

The relay modules required minimal testing due to their straightforward operation. We confirmed functionality by writing a simple routine to set the relay pins high, activating them. We further validated this by integrating the relays with the pushbuttons,

ensuring they triggered only when pressed. The water pumps were tested by connecting them through the relays and submerging them in water to confirm reliable actuation.

Once all individual components were verified to work correctly, we moved on to system-level testing. We began by integrating one button, one relay, and one pump, along with the RFID reader and ultrasonic sensor, to confirm their combined operation before scaling up to the full system.

## 4. Results

The following figures were photos taken from the results video we filmed to demonstrate functionality/results. You can find the full video here: FullDevFuncVid.MOV. Our device worked as expected, shown by the photos below. The first photo shows the liquid not being dispensed when the pushbutton has been pushed, as there is no cup being sensed. The second photo shows the use of the correct RFID card turning off the blinking light, and the third photo shows the dispenser dispensing liquid after using the RFID and placing the cup in the right place in front of the sensor.


Figure 7: Device Picture 1


Figure 8: Device Picture 2


Figure 8: Device Picture 3

As seen above, we encountered a small spillage issue. We also noticed minor inconsistencies in sensor readings and button responses during pump operation. Pictured in the last photo, you can see these issues resulted in being mitigated by elevating the jugs of liquid using a toolbox. These results will be discussed more in the next section.

## 5. Discussion of Results

The device performs well overall and functions as intended when free from software bugs or hardware interference. However, some performance issues arise primarily due to electronic noise generated by running all components on a single breadboard and shared power supply. Notably, the water pumps appear to be a significant source of noise, which affects the accuracy of the ultrasonic sensor and causes erratic behavior in the pushbuttons. This conclusion was drawn from observing that the system operates reliably when the pumps are disconnected and not in use, with consistent sensor readings and stable button response.

When the pumps are active, however, the ultrasonic sensor frequently reports incorrect distance measurements, and the pushbuttons exhibit noticeable bouncing. We also observed that the current dispenser height contributed to occasional spillage and placed strain on the pumps, likely reducing their efficiency. Lowering the dispenser and its pipes could address both the spillage and pump strain, improving overall performance and reducing waste.

Despite these challenges, the overall design met our expectations, and the hardware integration was clean and well-organized. Looking ahead, several improvements could enhance the system's reliability and usability. First, housing the device in a more robust and waterproof enclosure, rather than cardboard, would improve durability and prevent damage from liquid exposure. Additionally, replacing the 5V pumps with 12V models powered by an isolated supply could reduce electronic noise while providing stronger and faster fluid movement. Lastly, upgrading the ultrasonic sensor to an infrared (IR) sensor may offer more accurate and seamless cup detection.

## 6. Conclusion

In conclusion, this project successfully demonstrated the design and implementation of a functional, automated soda dispenser that integrates distance sensing, RFID authentication, and user-activated dispensing. The system reliably controls access and operation based on user presence and authorization, providing a secure and user-friendly experience. This work highlights the potential for combining embedded systems with common household utilities to create interactive, automated devices. Moving forward, improvements in power management, sensor selection, and mechanical design could further enhance reliability and performance, paving the way for practical applications in home automation and self-serve beverage systems.

## Acknowledgements

I would like to extend my sincere gratitude to everyone who contributed to the success of this project, as well as to the many projects that came before it. This work would not have been possible without the support of my dedicated and insightful partner, the guidance of our exceptional teaching assistants, and the expertise and encouragement of Professor Beichel, whose instruction laid the foundation for understanding and completing these labs.

## Appendix A - Source Code

```
/*
 * FinalProject.c
 *
 * Created: 4/28/2025 3:55:00 PM
 * Author : Adrian Alvarez & Seth Bolen
 */

#define F_CPU 16000000UL  // 16 MHz clock
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <Ultrasonic.h>
#include <utils.h>
#include <spi.h>
#include <mfrc522.h>

// Define UART baud rate
#define BAUD 9600 // Baud rate for serial communication
#define MYUBRR F_CPU/16/BAUD-1  // Baud rate register calculation

#define RELAY_PIN1 PD5 // Relay control pin 1
#define BUTTON1_PIN PC4 // First pushbutton input (was relay before)
#define BUTTON2_PIN PC5 // Second pushbutton input (new pin)
#define RELAY_PIN2 PD4 // Relay control pin 2 (was button before)
#define TRIG_PIN PB0 // Trigger pin
#define ECHO_PIN PD2 // Echo pin

// Global variables for sensor
uint16_t distance;
uint8_t diagnostics;
volatile uint16_t pulse;
volatile uint8_t iIRC;
volatile int f_wdt;

// Global variables for RFID
uint8_t SelfTestBuffer[64];
uint8_t byte;
uint8_t str[MAX_LEN];
volatile uint8_t uid_match;
volatile uint8_t distance_ok;

////////////////////////////////////////////
// Setups everything
////////////////////////////////////////////
void setup(void) {
```

```
    // Set relay pins
    DDRD |= (1 << RELAY_PIN1) | (1 << RELAY_PIN2);//

    // Set button pin as input
    DDRC &= ~(1 << BUTTON1_PIN);
        DDRC &= ~(1 << BUTTON2_PIN);

        // Initializes RFID
        init_rfid();
        // Initializes sensor
        init_ultrasonic();

    // Enable pull-up resistor on button pin
    PORTC |= (1 << BUTTON1_PIN);
        PORTC |= (1 << BUTTON2_PIN);

    // Initialization for global variables
        distance = 0;
        diagnostics = 0;
        iIRC = 0;
        f_wdt = 1;
        uid_match = 0;
        distance_ok = 0;

        PORTD &= ~((1 << RELAY_PIN1) | (1 << RELAY_PIN2));
}

////////////////////////////////////////////
// Setups RFID
////////////////////////////////////////////
void init_rfid(){
        spi_init();
        //_delay_ms(1000);

        //init reader
        mfrc522_init();

        byte = mfrc522_read(ComIEnReg);
        mfrc522_write(ComIEnReg,byte|0x20);
        byte = mfrc522_read(DivIEnReg);
        mfrc522_write(DivIEnReg,byte|0x80);

        //_delay_ms(1500);
}

////////////////////////////////////////////
// For serial monitor
////////////////////////////////////////////
void uart_init(unsigned int ubrr) {
        UBRR0H = (unsigned char)(ubrr>>8);
        UBRR0L = (unsigned char)ubrr;
        UCSR0B = (1<<TXEN0); // Enable transmitter
        UCSR0C = (1<<UCSZ01)|(1<<UCSZ00); // 8-bit data
}

// Transmit a single character over UART
void uart_transmit(char data) {
        while (!(UCSR0A & (1 << UDRE0))); // Wait until transmit buffer is empty
        UDR0 = data; // Load data into the UART data register
```

```
}

// Send a string over UART
void uart_print(const char* str) {
        while (*str) { // Loop through each character until null terminator
                uart_transmit(*str++); // Transmit current character and move to next
        }
}

// Convert a float to string and print over UART
void uart_print_float(float num) {
        char buffer[16]; // Buffer to hold converted float string
        dtostrf(num, 5, 2, buffer); // Convert float to string with 2 decimal places
        uart_print(buffer); // Send the resulting string over UART
}

////////////////////////////////////////////
// Main loop
////////////////////////////////////////////
void main_loop(void) {
        distance = getDistance_main(&diagnostics);

        // Check and set distance_ok flag
        if (distance <= 13 || (distance <= 80 && distance >= 70)) {
                distance_ok = 1;
        }
        else {
                distance_ok = 0;
        }

        uart_print("Distance: "); // Print label
        uart_print_float(distance); // Print measured distance
        uart_print(" cm\r\n"); // Print units and newline

        // RFID check - updates rfid_ok flag inside
        if (uid_match == 0){
                check_rfid();
        }

        // Only check buttons if both conditions are met
        while (distance_ok && uid_match) {
                distance = getDistance_main(&diagnostics);

                // Check and set distance_ok flag
                // its not a bug its a feature ;)
                if (distance <= 13 || (distance <= 75 && distance >= 65)) {
                        distance_ok = 1;
                }
                else {
                        uid_match = 0;
                        distance_ok = 0;
                }

                uart_print("Distance: "); // Print label
                uart_print_float(distance); // Print measured distance
                uart_print(" cm\r\n"); // Print units and newline

                // --- BUTTON 1 CHECK ---
                if (!(PINC & (1 << BUTTON1_PIN))) {
```

```
                        if (!(PINC & (1 << BUTTON1_PIN))) {
                                // Activate relay 1
                                PORTD |= (1 << RELAY_PIN1);

                                // Wait for button release
                                while (!(PINC & (1 << BUTTON1_PIN))) {
                                        _delay_ms(10);
                                }
                        }
                }

                // --- BUTTON 2 CHECK ---
                if (!(PINC & (1 << BUTTON2_PIN))) {
                        if (!(PINC & (1 << BUTTON2_PIN))) {
                                // Activate relay 2
                                PORTD |= (1 << RELAY_PIN2);

                                // Wait for button release
                                while (!(PINC & (1 << BUTTON2_PIN))) {
                                        _delay_ms(10);
                                }
                        }
                }

                PORTD &= ~((1 << RELAY_PIN1) | (1 << RELAY_PIN2)); // Sets the relays off
        }

        _delay_ms(100); // Polling interval
}

void check_rfid(){
        uart_print("In RFID");
        uart_print("\r\n");

        byte = mfrc522_request(PICC_REQALL, str);
        uid_match = 0;  // Reset match flag each scan

        if (byte == CARD_FOUND) {
                byte = mfrc522_get_card_serial(str);
                if (byte == CARD_FOUND) {
                        // Check for matching UID: 61 24 18 02 5F
                        if (str[0] == 0x61 && str[1] == 0x24 && str[2] == 0x18 && str[3] == 0x02 &&
str[4] == 0x5F) {
                                uart_print("Card match");
                                uart_print("\r\n");
                                uid_match = 1;
                        }
                        else{
                                uart_print("No card match");
                                uart_print("\r\n");
                        }
                        _delay_ms(2500); // Wait before next read attempt
                }
        }

        _delay_ms(1000); // General polling delay
}
```

```
/* ****************************************************************
Name:           Ultrasonic Interrupt
Inputs:         none
Outputs:internal timer
Description:calculates elapsed time of a measurement
**************************************************************** */
ISR(INT0_vect)
{
        switch (iIRC)
        {
                case 0: // When logic changes from LOW to HIGH
                {
                        iIRC = 1;
                        TCCR1B |= (1<<CS11);
                        break;
                }
                case 1:
                {
                        /* reset iIRC */
                        iIRC = 0;
                        /* stop counter */
                        TCCR1B &= ~(1<<CS11);
                        /* assign counter value to pulse */
                        pulse = TCNT1;
                        /* reset counter */
                        TCNT1=0;
                        break;
                }
        }
}

/* ****************************************************************
Name:           Watchdog Interrupt
Inputs:         none
Outputs:f_wdt
Description:wakes up processor after internal timer limit reached (8 sec)
**************************************************************** */
ISR(WDT_vect)
{
        /* set the flag. */
        if(f_wdt == 0)
        {
                f_wdt = 1;
        }
        //else there is an error -> flag was not cleared
}

////////////////////////////////////////////////
// Main
////////////////////////////////////////////////
int main(void) {
        uart_init(MYUBRR); // Initialize UART with calculated baud rate
    setup();
    while (1) {
        main_loop();
    }
}
```

## Appendix B: References

**asif-mahmud.** *MIFARE-RFID-with-AVR: MIFARE RC522 Module Library for Atmel MCU*. GitHub. Accessed May 9, 2025. https://github.com/asif-mahmud/MIFARE-RFID-with-AVR.

**DIYables.** *Relay 5V 1-Channel Product Page*. DIYables. Accessed May 9, 2025. https://diyables.io/products/relay-5v-1-channel.

**Handson Technology.** *RC522 RFID Development Kit Datasheet*. Handson Technology. Accessed May 9, 2025. https://www.handsontec.com/dataspecs/RC522.pdf.

**Microchip Technology.** *ATmega48A/PA/88A/PA/168A/PA/328/P Datasheet*. 2018. Microchip Technology. Accessed May 9, 2025. https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A.pdf.

**Ovidiu22.** *HC-SR04: A C-library for the HC SR-04 Ultrasonic Sensor*. GitHub. Accessed May 9, 2025. https://github.com/Ovidiu22/HC-SR04.

**SparkFun Electronics.** *Ultrasonic Ranging Module HC-SR04 Datasheet*. SparkFun Electronics. Accessed May 9, 2025. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.