Assignment 2 - Burglar Alarm System Report

Course: CS3514 (C-Programming for Microcontrollers)

Name: Mariana-Ionela Muntian

Partner: Teodora Nemes

1. Project Overview

a. Requirements

Use your Arduino to create a Burglar alarm system with the characteristics described below.

Input and Output to/from the system (setting parameters, inputting passwords, etc), should be from the Serial Monitor.

When the alarm is not set (unset), the Serial Monitor should display "Not Armed".
When the alarm is set, the Serial Monitor should display "Armed".
The user should enter a password to set the alarm and a password to unset the alarm.

The alarm should have 5 zone types:

(1) Entry/Exit
(2) Digital alarm
(3) Analog Alarm
(4) Continuous monitoring
(5) Unused.

The alarm should have at least 4 zones - the type of each zone should be configurable. A specific pin should correspond to a zone. The activity/state on/of a zone is associated with the activity/state on/of its associated pin.

An initialization function should be provided to configure the zones of the alarm. This configuration should include deciding on each zone type and the pin asscoiated with the zone.

- **Entry/Exit** zones should have 2 associated timers: an entry timer and an exit timer. The duration of these timers should be configurable in the initialization function.
  The exit timer should start running when the user enters the password when the alarm is unset. All entry/exit sensors should be closed before this timer expires. If not, the alarm should sound.
  The entry timer should start running when the alarm is set and an entry/exit sensor is opened. The user must input the correct password before this timer expires or the alarm should sound.
  The password should be configurable in the initialization function.

- **Digital Alarm Zones**, if opened when the alarm is set, should result in the alarm sounding immediately.

- **Analog Alarm Zones**, if these zones exceed configurable thresholds (configured in the initialization function) when the alarm is set, should result in the alarm sounding immediately.

- **Continuous Monitoring Zones** should always be closed. If opened, even when the alarm is unset, should sound the alarm immediately.

- **Unused Zones** are ignored by the alarm.

In general, when the alarm is unset, the state of the zones (apart from the continuous monitoring zones) are irrelevant. However, all non-Unused zones (apart from the Entry/Exit zones) should be closed before the alarm can be set. These zonee should be displayed on the Serial Monitor and should prevent the alarm from being set.

Structure your code for readability, correctness and efficiency.

b. Idea behind implementation

Configuration

The board has 4 buttons, each representing one zone. Each zone can be assigned one of the 5 types (the user chooses the type of zone from the serial monitor).

The first two of the zones have interrupts attached (buttons connected to pins 3 and 4). Therefore, the user can select only to assign the types:

Press ‘1’: Digital Alarm Zones
Press ‘3’: Continuous Monitoring Zones

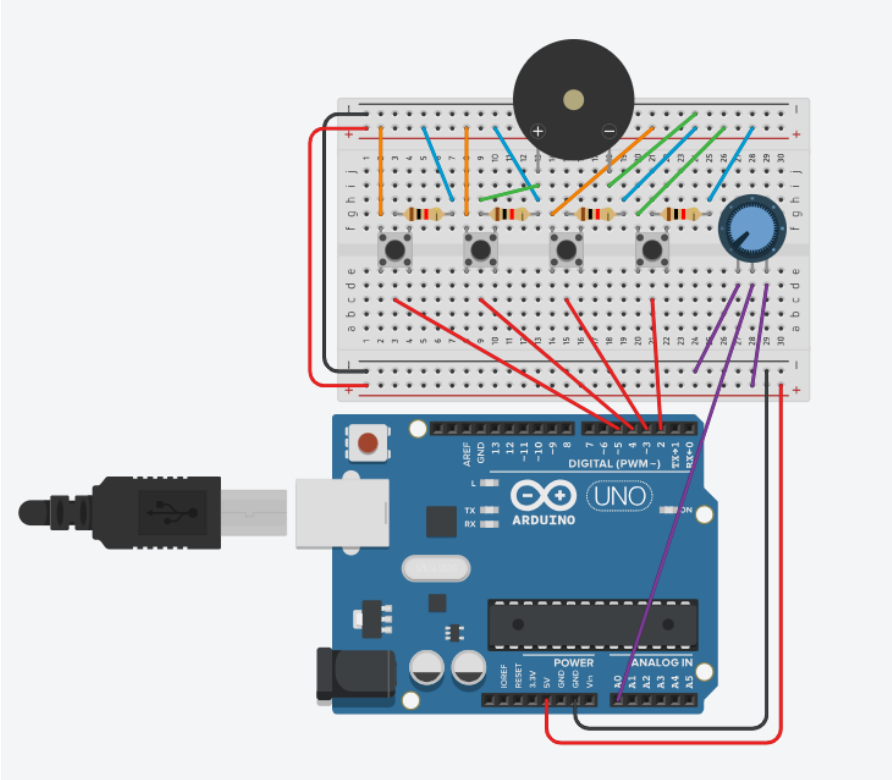
The first two of the zones have interrupts attached (buttons connected to pins 3 and 4). Therefore, the user can select only to assign the types:

Press ‘0’: Entry/Exit
Press ‘2’: Analog Alarm Zones
Press ‘4’: Unused Zones

Activating the zones

To activate one zone, the user presses the button corresponding to that zone. For each zone, a function that handles its condition is implemented. Depending on the conditions specific to each zone, the alarm will start or not. The functions corresponding to zones 1 and 2 are triggered by the interrupt while functions corresponding to zones 3 and 4 are called from the while loop.

2. Circuit Diagram



3. Code

These macros assign specific names to hardware pins:

- ZONE1 – ZONE4 represent the pins connected to the four alarm zones.
- BUZZER specifies the pin for the alarm buzzer.
- POTENTIOMETER is the analog pin used for monitoring the Analog Alarm Zone. It reads a value from a potentiometer and compares it against a pre-configured threshold to determine if the zone should trigger the alarm.

```
#define ZONE1 2 // PIN2 - has interrupt
#define ZONE2 3 // PIN3 - has interrupt
#define ZONE3 4
#define ZONE4 5
#define BUZZER 7
#define POTENTIOMETER A0 // Analog pin for threshold checks
#define READ_PORT(val) val = PIND
```

```
enum zone_types {EXIT_ENTRY, DIGITAL_ALARM, ANALOG_ALARM, CONT_MONIT,
UNUSED};
```

This structure stores the properties of a zone:

- pin: The hardware pin associated with the zone.
- type: The type of zone (from zone_types).
- threshold: A threshold value for analog zones.
- active: Tracks whether the zone is currently active.

```
//There are 5 zone types
struct Zone {
  byte pin;
  zone_types type;
  int threshold;
  bool active;
};
```

These constants define system parameters: NUM_ZONES (the total number of zones being monitored), PASSWORD_LEN (maximum password length for arming/disarming the alarm)

```
// Constants
#define NUM_ZONES 4
#define PASSWORD_LEN 6
```

These variables maintain system state: zones (stores configurations for all zones), alarmArmed (indicates if the alarm is armed), alarmTriggered (tracks if the alarm is currently triggered), timerExpired (indicates if a timer has expired) and timerSeconds (tracks the countdown for the entry/exit timers)

```
// Global variables
Zone zones[NUM_ZONES];
volatile bool alarmArmed = false;
volatile bool alarmTriggered = false;
volatile bool isTimerOn = false;
volatile bool timerExpired = false;
volatile unsigned long timerSeconds = 0;
```

These variables store the passwords for arming and disarming the alarm:

- `setPasswordEntry`: Password for arming the alarm.
- `setPasswordExit`: Password for handling exit functionality.
- `enteredPassword`: Temporarily holds the user's input for password verification.

```
// Variables related to handling the passwords
char setPasswordEntry[PASSWORD_LEN] = "1234";
char setPasswordExit[PASSWORD_LEN] = "5678";
char enteredPassword[PASSWORD_LEN];
volatile bool entryMode = false;
volatile bool exitMode = false;
bool lastAlarmStatusPrinted = false;
```

The initializeZones() function sets up the alarm system's zones by associating each zone with a specific pin, type, and initial configuration. Only the Analog Alarm zone needs a threshold value and all the zones are marked as inactive in the beginning.

```
//Init. the 4 zones
void initializeZones() {
  zones[0] = {ZONE1, UNUSED, 0, false};
  zones[1] = {ZONE2, UNUSED, 0, false};
  zones[2] = {ZONE3, UNUSED, 500, false}; // Example analog threshold
  zones[3] = {ZONE4, UNUSED, 0, false};
}
```

The setupTimer1 function configures Timer1 to operate in CTC mode with a 1 Hz frequency for countdown functionality. It calculates the compare match value based on the system clock, sets the timer prescaler, and enables the compare match interrupt. The stopTimer1 function halts Timer1 by disabling the prescaler and the interrupt. The ISR(TIMER1_COMPA_vect) handles the timer interrupt, increments the elapsed time, and triggers the alarm if the timer expires during entry or exit modes.

```
//Timer
void setupTimer(unsigned int seconds) {
  isTimerOn = true;
  cli();
  TCCR1A = 0;
  TCCR1B = 0;

  // The system clock on an Arduino is 16 MHz, so to count seconds, we need
to set the timer to count at a frequency that is one count per second.
  // CS12 selects a prescaler of 256
```

```
  OCR1A = (16000000 / 256) - 1;
  TCCR1B |= (1 << WGM12) | (1 << CS12);
  TIMSK1 |= (1 << OCIE1A);
  timerSeconds = seconds;
  sei();
}

void stopTimer() {
  isTimerOn = false;
  cli();
  TCCR1B = 0;
  TIMSK1 &= ~(1 << OCIE1A);
  sei();
}

ISR(TIMER1_COMPA_vect) {
  static unsigned long elapsed = 0;
  elapsed++;
  if (elapsed >= timerSeconds) {
    timerExpired = true;
    elapsed = 0;
    stopTimer();
    Serial.println("Timer expired! Alarm triggered.");
    alarmTriggered = true;
    PORTD |= (1 << BUZZER); // Activate buzzer
  }
}
```

The handlePassword function validates the user-provided password and manages the alarm system's state accordingly. If the entered password matches the expected value, it either arms or disarms the system, depending on the current state, and resets any active entry or exit modes.

```
// Function that checks if the password that I enter matches the existing
password.
bool handlePassword(const char actualPassword[]) {
  if (Serial.available() > 0) {
    int len = Serial.readBytesUntil('\n', enteredPassword, PASSWORD_LEN - 1);
    enteredPassword[len] = '\0'; // Null-terminate the string

    if (strcmp(enteredPassword, actualPassword) == 0) {
      return true;  // Password correct
    } else {
```

```
      Serial.println("Incorrect password.");
      return false;  // Password incorrect
    }
  }
  return false; // No password entered
}

//Configure the types of the zones based on the user choices
void configureZones() {
  for (int i = 0; i < NUM_ZONES; i++) {
    Serial.print("Configure ZONE ");
    Serial.print(i + 1);
    Serial.println(":");

    if (i == 0 || i == 1) {
      Serial.print("\n");
      Serial.println("1: Digital Alarm");
      Serial.println("3: Continuous Monitoring");
    } else {
      Serial.print("\n");
      Serial.println("0: Entry/Exit");
      Serial.println("2: Analog Alarm");
      Serial.println("4: Unused");
    }

    // Wait for valid input
    while (true) {
      if (Serial.available() > 0) {
        char choice = Serial.read();

        Serial.print("\nReceived choice: ");
        Serial.println(choice);

       // We have added this condition because it was accidently reading the
spaces or newline before we had time to write the input
        if (choice == '\n' || choice == '\r') {
          continue;
        }

        // Clear the rest of the serial buffer if needed
        while (Serial.available() > 0) {
          Serial.read(); // Discard any other bytes in the buffer
        }

        // Validate input based on zone number
```

```
        if ((i < 2 && (choice == '1' || choice == '3')) ||
            (i >= 2 && (choice == '0' || choice == '2' || choice == '4'))) {
          zones[i].type = static_cast<zone_types>(choice - '0');
          Serial.print("Zone ");
          Serial.print(i + 1);
          Serial.print(" set to type ");
          Serial.println(zones[i].type);
          break;  // Exit loop once valid input is received
        } else {
          Serial.println("Invalid choice. Please try again.");
        }
      }
    }
  }
}
```

This array maps zone types to their corresponding functions.

```
// Function pointer array definition
void (*zoneActions[])() = {
  exit_entry_func,
  digital_alarm_func,
  analog_alarm_func,
  continuous_monit,
  unused_zone_func
};

//Based on which button (which corresponds to a zone) is pressed, call the
appropriate zone action
void handleZonePress(int zoneIndex) {
  if (zones[zoneIndex].type != UNUSED) {
    zoneActions[zones[zoneIndex].type]();
  }
}
```

The armAlarm function sets the alarmArmed flag to true and notifies the user via the Serial Monitor that the alarm is armed. The disarmAlarm function resets the alarmArmed and alarmTriggered flags, turns off the buzzer, and informs the user that the alarm has been disarmed.

```
//This function arms the alarm (when the user wants to exit)
```

```
void armAlarm() {
  alarmArmed = true;
  lastAlarmStatusPrinted = false;
  Serial.println("Alarm armed.");
}

//This function arms the alarm
void disarmAlarm() {
  alarmArmed = false;
  alarmTriggered = false;
  digitalWrite(BUZZER, LOW);
  lastAlarmStatusPrinted = false;
  Serial.println("Alarm disarmed.");
}
```

The exit_entry_func function manages the behavior of the Entry/Exit Zone (ZONE4).

```
// Function that handles the action that must happen in entry OR exit zone
void exit_entry_func() {

  // If we are in entry mode
  if (alarmArmed) {
    Serial.println("Entry detected. Enter the entry password to disarm the
alarm.");
    entryMode = true;
    exitMode = false;

    setupTimer(15);
    bool passwordCorrect = false;

    // Wait for the user to insert the correct password for 15 seconds
    while (isTimerOn) {
      if (handlePassword(setPasswordEntry)) { // Check if password is correct
        passwordCorrect = true;
        stopTimer();
        break;
      }
    }

    if (passwordCorrect) {
      Serial.println("Correct password.");
      disarmAlarm();
    } else {
      Serial.println("Incorrect password OR timeout expired.");
      Serial.println("ALARM SOUND IS ON!");
```

```
      digitalWrite(BUZZER, HIGH);
    }
  } else {
    if (!exitMode) {
    Serial.println("Exit detected. Enter the exit password to arm the
alarm.");
      exitMode = true;
      entryMode = false;

      setupTimer(15);
      bool passwordCorrect = false;

      while (isTimerOn) {
        if (handlePassword(setPasswordExit)) {
          passwordCorrect = true;
          stopTimer();
          break;
        }
      }

      // If the password was correct, arm the alarm
      if (passwordCorrect) {
        armAlarm();
      } else {
        Serial.println("Password not entered correctly or timeout expired.");
        Serial.println("Triggering alarm sound...");
        digitalWrite(BUZZER, HIGH); // Trigger buzzer sound
      }
    }
  }
}
```

If the alarm system is armed and the zone is triggered, it sets the alarmTriggered flag to true, activates the buzzer by setting its pin HIGH.

```
void digital_alarm_func() {
  if (alarmArmed) {
    Serial.println("Digital alarm was triggered!");
    alarmTriggered = true;
  }
}
```

The analog_alarm_func function monitors the Analog Alarm Zone by reading the analog value from the potentiometer connected to the POTENTIOMETER pin. If the system is armed and the sensor value exceeds the predefined threshold for the zone, the function sets the alarmTriggered flag to true, activates the buzzer by setting its pin HIGH.

```
void analog_alarm_func() {
  int sensorValue = analogRead(POTENTIOMETER);
  if (alarmArmed && sensorValue > zones[2].threshold) {
    Serial.println("Analog alarm was triggered!");
    alarmTriggered = true;
  }
}
```

The continuous_monit function handles the Continuous Monitoring Zone, which requires the zone to always remain closed. When an interrupt signals that the zone has been opened, the function sets the alarmTriggered flag to true

```
// Continous monitoring zone can start even if the alarm is not armed
void continuous_monit() {
  Serial.println("Continuous Monitoring Zone was triggered!");
  alarmTriggered = true;
}

void unused_zone_func() {
  Serial.println("Unused zone triggered. No action taken.");
}

// Interrupt handlers for Zones 1 and 2
void zone1ISR() { handleZonePress(0); }
void zone2ISR() { handleZonePress(1); }
```

The setup() function initializes the system's hardware and software configurations. It begins serial communication at a baud rate of 9600 for user interaction. The DDRD and PORTD registers configure the BUZZER pin as an output and the zone pins (ZONE1–ZONE4) as inputs with internal pull-up resistors enabled. The initializeZones() function sets up the zone configurations, linking each pin to a specific zone type. Interrupts are attached to ZONE1 and ZONE2 for the Continuous Monitoring and Digital Alarm zones, respectively, triggering their associated functions when a falling edge (button press) is detected.

```
void setup() {
  Serial.begin(9600);

  //  BUZZER is OFF initially
  // Configure buzzer pin
  DDRD |= (1 << BUZZER);
  PORTD &= ~(1 << BUZZER);
```

```
  DDRD &= ~((1 << ZONE1) | (1 << ZONE2) | (1 << ZONE3) | (1 << ZONE4));
  PORTD |= ((1 << ZONE1) | (1 << ZONE2) | (1 << ZONE3) | (1 << ZONE4));

  initializeZones();
  configureZones();

  // Attach interrupts for ZONE1 and ZONE2 (DIGITAL_ALARM and CONT_MONIT
zones)
  attachInterrupt(digitalPinToInterrupt(ZONE1), zone1ISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(ZONE2), zone2ISR, FALLING);
}
```

The loop() function continuously monitors the alarm system's state and handles user interactions.

```
void loop() {

  //Zones 3 and 4 correspond to one zones that do not use interrupts
   uint8_t portState;
  READ_PORT(portState);  // Read the entire port

  if (portState & (1 << ZONE3)) {  // Check if ZONE3 is HIGH
    handleZonePress(2);
}

if (portState & (1 << ZONE4)) {  // Check if ZONE4 is HIGH
    handleZonePress(3);
}


  // Print the alarm status only once
  if (!lastAlarmStatusPrinted) {
    Serial.println(alarmArmed ? "Alarm Armed" : "Alarm Not Armed");
    lastAlarmStatusPrinted = true; // Flag that the message has been printed
  }

  // Activate the buzzer if the alarm is triggered
  if (alarmTriggered) {
    digitalWrite(BUZZER, HIGH); // Activate buzzer
  } else {
    digitalWrite(BUZZER, LOW); // Deactivate buzzer
  }

  delay(100); // Small delay for stability}
```