# Smart Room

Projekt "Smart Room" to system inteligentnego pomieszczenia oparty na mikrokontrolerze Arduino UNO, który automatyzuje oświetlenie i monitoruje warunki środowiskowe. System integruje czujniki temperatury, wilgotności oraz natężenia światła, reagując na zmiany w czasie rzeczywistym poprzez sygnalizację wizualną (LED) i dźwiękową (Buzzer). Dane prezentowane są na wyświetlaczu OLED oraz przesyłane do dedykowanej aplikacji PC, która umożliwia ich archiwizację i zdalne sterowanie.

# Smart Room

The "Smart Room" project is a smart room system based on the Arduino UNO microcontroller that automates lighting and monitors environmental conditions. The system integrates temperature, humidity, and light intensity sensors, responding to changes in real time through visual (LED) and audible (buzzer) signals. Data is presented on an OLED display and transmitted to a dedicated PC application, which enables archiving and remote control.

.

# 1. Spis treści

# 2. Cel i zakres projektu

Celem głównym projektu Smart Room jest zwiększenie komfortu użytkownika oraz poprawa efektywności energetycznej pomieszczenia poprzez automatyczne zarządzanie oświetleniem w zależności od pory dnia oraz aktualnego poziomu natężenia światła. System został zaprojektowany jako rozwiązanie modułowe, umożliwiające zarówno bieżące monitorowanie parametrów środowiskowych, jak i ich archiwizację oraz analizę historyczną.

## 2.1 Zakres funkcjonalny projektu obejmuje:

1. Akwizycja parametrów środowiskowych:
   - Ciągły pomiar temperatury i wilgotności powietrza oraz konwersja sygnałów na jednostki fizyczne.
   - Monitorowanie natężenia oświetlenia w pomieszczeniu za pomocą przetwornika rezystancyjnego (LDR).
   - Obsługa zegara czasu rzeczywistego (RTC) w celu synchronizacji zdarzeń czasowych.

2. Automatyka i sterowanie:
   - Implementacja algorytmu sterowania oświetleniem w trybie hybrydowym: na podstawie natężenia światła (zmierzch) lub harmonogramu godzinowego (RTC).
   - Obsługa trybu manualnego z możliwością wymuszenia stanu oświetlenia przez użytkownika.

3. System bezpieczeństwa i powiadomień:
   - Dwustopniowy system alarmowy (stany Warning i Critical) z histerezą programową, reagujący na przekroczenie dopuszczalnych norm klimatycznych.
   - Sygnalizacja optyczna (LED) oraz akustyczna (Buzzer) w przypadku wykrycia anomalii.

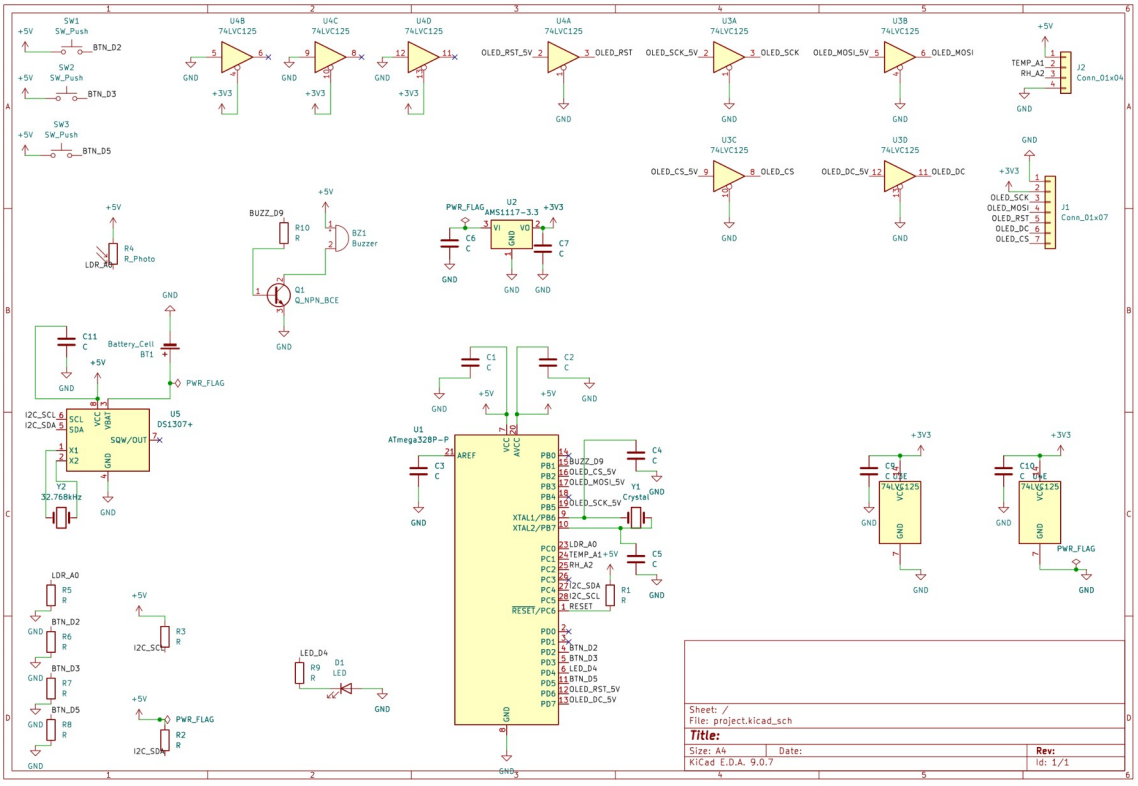
4. Lokalny interfejs użytkownika (HMI):
   - Prezentacja bieżących pomiarów, czasu systemowego oraz stanu pracy na wyświetlaczu OLED.
   - Menu konfiguracyjne obsługiwane przyciskami fizycznymi, umożliwiające edycję zakresu godzin trybu nocnego ("Night Mode").

5. Aplikacja PC i Telemetria:
   - Wizualizacja danych pomiarowych w czasie rzeczywistym na dynamicznych wykresach (Python/Matplotlib).

- Archiwizacja danych telemetrycznych do plików formatu CSV (Data Logging).
- Zdalna konfiguracja: Możliwość zmiany parametrów pracy urządzenia (interwał próbkowania, sterowanie oświetleniem) z poziomu komputera poprzez interfejs UART.

# 3. Opis systemu

## 3.1 Schemat elektryczny




## 3.2 Zastosowane czujniki i moduły

W projekcie wykorzystano następujące elementy sprzętowe:

- Arduino UNO – platforma z mikrokontrolerem ATmega328P, pełniąca rolę jednostki centralnej systemu.
- Symulacja czujnika środowiskowego (Potencjometry/SHT Analog) – sygnały wejściowe podłączone do pinów analogowych A1 i A2, odwzorowujące odczyty temperatury i wilgotności (zamiast cyfrowej magistrali I2C).
- LDR (Fotorezystor) – czujnik natężenia światła pracujący w układzie dzielnika napięcia.

- RTC DS1307 – moduł zegara czasu rzeczywistego komunikujący się przez magistralę I2C, odpowiedzialny za śledzenie aktualnej godziny.
- OLED SSD1306 (128×64) – graficzny wyświetlacz danych sterowany za pomocą interfejsu SPI (piny: DC, RST, CS).
- Elementy wejściowe – 3 przyciski podłączone do pinów cyfrowych (2, 3, 5).
- Elementy wyjściowe – Buzzer pasywny (pin 9) oraz dioda LED (pin 4) pełniące funkcję sygnalizacyjną.

## 3.3 Wykorzystane biblioteki programistyczne

Do realizacji projektu zastosowano następujące biblioteki:

Arduino:

- Adafruit_SSD1306 oraz Adafruit_GFX – biblioteki graficzne obsługujące sterownik wyświetlacza OLED, umożliwiające rysowanie tekstu oraz kształtów geometrycznych.
- RTClib – biblioteka wysokiego poziomu do obsługi modułu zegara czasu rzeczywistego (DS1307) poprzez magistralę I2C.
- SPI.h oraz Wire.h – biblioteki standardowe obsługujące odpowiednio magistralę SPI (dla wyświetlacza) oraz I2C (dla RTC).

Aplikacja PC:

- Tkinter (moduł standardowy) – biblioteka służąca do budowy graficznego interfejsu użytkownika (okna, przyciski, układ strony).
- pySerial – moduł realizujący obsługę portu szeregowego (COM/UART) do dwukierunkowej komunikacji z mikrokontrolerem.
- Matplotlib – biblioteka do generowania dynamicznych wykresów przebiegów czasowych osadzonych w oknie aplikacji.
- Threading oraz Queue – moduły standardowe wykorzystane do implementacji wielowątkowości, zapewniające płynny odbiór danych bez blokowania interfejsu graficznego.

# 4. Implementacja

Poniższa sekcja zawiera szczegółową analizę implementacji oprogramowania sterującego systemem monitorowania parametrów środowiskowych. Rozwiązanie oparto na architekturze klient-serwer, gdzie rolę serwera danych pełni mikrokontroler (ATmega328P), a klienta – aplikacja PC.

## 4.1 Konfiguracja rejestrów

**Listing 4.1.** Kod źródłowy funkcji ioInitRegisters

```
1     void ioInitRegisters() {
2       DDRD |= (1 << DDD4);
3       DDRB |= (1 << DDB1);
4       DDRD &= ~((1 << DDD2) | (1 << DDD3) | (1 << DDD5));
5       PORTD |=  (1 << PORTD2) | (1 << PORTD3) | (1 << PORTD5);
6       PORTD &= ~(1 << PORTD4);
7       PORTB &= ~(1 << PORTB1);
8     }
```

Funkcja realizuje niskopoziomową konfigurację portów wejścia/wyjścia (GPIO) poprzez bezpośrednie operacje na rejestrach mikrokontrolera, zamiast użycia funkcji pinMode():

- Rejestry DDR (Data Direction Register): Ustawiają kierunek przepływu danych.
  - Bity DDD4 (Port D) oraz DDB1 (Port B) są ustawiane na 1, co konfiguruje piny diody LED i Buzzera jako wyjścia.
  - Bity DDD2, DDD3, DDD5 są zerowane, co ustawia piny przycisków jako wejścia.

- Rejestry PORT:
  - Dla wejść (przyciski): Ustawienie bitów na 1 włącza wewnętrzne rezystory podciągające (Internal Pull-up), co jest konieczne dla stabilnego odczytu przycisków zwierających do masy.
  - Dla wyjść (LED, Buzzer): Wyzerowanie bitów zapewnia bezpieczny stan początkowy (urządzenia są wyłączone przy starcie systemu).

## 4.2 Algorytm logiki sterowania

**Listing 4.2.** Kod źródłowy funkcji updateLogic

```
1     void updateLogic() {
2       bool dark = ldr < LDR_LIMIT;
3       bool evening = isEvening();
4
5        bool warningTemp = (tempC < TEMP_WARN_LOW) || (tempC >
      TEMP_WARN_HIGH);
6        bool criticalTemp = (tempC < TEMP_CRIT_LOW) || (tempC >
      TEMP_CRIT_HIGH);
7        bool warningHum = (humP < HUM_WARN_LOW) || (humP >
      HUM_WARN_HIGH);
8        bool criticalHum = (humP < HUM_CRIT_LOW) || (humP >
      HUM_CRIT_HIGH);
9
10      bool warning  = warningTemp || warningHum;
```

```
11      bool critical = criticalTemp || criticalHum;
12
13      warningState = warning;
14      criticalState = critical;
15      buzzState = warning || critical;
16
17      if (mode == AUTO) baseLedOn = (dark || evening);
18      else baseLedOn = manualLED;
19
20      ledState = baseLedOn || warning || critical;
21    }
```

Funkcja ta pełni rolę głównego węzła decyzyjnego systemu. Jest wywoływana cyklicznie w pętli głównej i odpowiada za przetworzenie surowych danych pomiarowych na stany wyjść (LED, Buzzer). Jej działanie można podzielić na trzy etapy:

1. Weryfikacja progów alarmowych:
   - Aktualne odczyty temperatury (tempC) i wilgotności (humP) są porównywane ze zdefiniowanymi stałymi dla stanów ostrzegawczych (WARN) i krytycznych (CRIT).
   - Zastosowano sumę logiczną (OR), co oznacza, że przekroczenie progu przez *dowolny* z parametrów (temperatura lub wilgotność) aktywuje odpowiednią flagę alarmową (warning, critical).

2. Sterowanie oświetleniem roboczym:
   - Tryb AUTO: Decyzja o włączeniu oświetlenia bazowego (baseLedOn) zależy od czujnika światła (zmienna dark – gdy LDR spada poniżej limitu) lub harmonogramu godzinowego (zmienna evening – odczyt z RTC).
   - Tryb MANUAL: Oświetlenie bazowe zależy wyłącznie od flagi manualLED ustawianej przez użytkownika przyciskiem lub komendą z PC.

3. Priorytetyzacja wyjść (Safety Override):
   - Finalny stan diody LED (ledState) jest wynikiem sumy logicznej oświetlenia bazowego oraz stanów alarmowych.
   - Kluczowa cecha: Wystąpienie alarmu (ostrzegawczego lub krytycznego) wymusza włączenie diody LED, niezależnie od tego, czy system jest w trybie automatycznym, czy oświetlenie zostało ręcznie wyłączone. Gwarantuje to wizualną sygnalizację zagrożenia w każdych warunkach.

## 4.3 Główna pętla programu

**Listing 4.3.** Kod źródłowy funkcji loop

```
1     void loop() {
2       readCmd();
3       handleButtons();
4
5       ledAlarmUpdate(warningState, criticalState, baseLedOn);
6       buzzerPassive(warningState, criticalState);
7
8       if (millis() - lastReadSend >= intervalMs) {
9         lastReadSend = millis();
10        sendUART();
11      }
12
13      if (millis() - lastReadMain >= 250) {
14        lastReadMain = millis();
15        ldr = analogRead(PIN_LDR);
16        tempC = adcToTemp(analogRead(PIN_SHT_TEMP));
17        humP  = adcToHum(analogRead(PIN_SHT_HUM));
18        now = rtc.now();
19        updateLogic();
20        drawOLED();
21      }
22    }
```

Funkcja loop stanowi serce programu, realizując architekturę tzw. Super Loop z wykorzystaniem programowych timerów opartych na funkcji systemowej millis(). Takie podejście pozwala na symulację wielozadaniowości kooperacyjnej (non-blocking), co jest kluczowe dla płynności działania interfejsu.

Struktura pętli dzieli zadania na dwie kategorie:

1. Zadania czasu rzeczywistego (wykonywane w każdym obiegu):
   - Obsługa wejść: Funkcje readCmd() (UART) oraz handleButtons() są wywoływane w każdej iteracji, aby zapewnić natychmiastową reakcję na komendy z PC lub naciśnięcia przycisków przez użytkownika (minimalizacja opóźnień input-lag).
   - Obsługa wyjść: Funkcje ledAlarmUpdate i buzzerPassive również działają w każdym obiegu. Jest to konieczne, ponieważ realizują one efekty dynamiczne (np. miganie diody, modulacja dźwięku) w oparciu o własne, wewnętrzne liczniki czasu. Gdyby umieszczono je w wolniejszym timerze, efekty alarmowe byłyby szarpane lub niesynchroniczne.

2. Zadania okresowe (harmonogramowane przez millis()):
   - Timer Transmisji (intervalMs): Co określony interwał (domyślnie 1000 ms, konfigurowalne) system wysyła paczkę danych telemetrycznych do komputera (sendUART). Oddzielenie tego procesu zapobiega zalewaniu łącza szeregowego zbyt dużą ilością danych.
   - Timer Główny (250 ms): Odpowiada za cykliczną akwizycję danych z czujników (analogRead), aktualizację zegara RTC, przeliczenie logiki sterowania

(updateLogic) oraz odświeżenie ekranu OLED. Częstotliwość 4 Hz (co 250 ms) jest kompromisem między płynnością odczytów na ekranie a stabilnością pomiarów.

## 4.4 Odbiór danych z oprogramowania PC

**Listing 4.4.** Kod źródłowy funkcji _pool_queue

```
1     def _poll_queue(self):
2         while not self.q.empty():
3             line = self.q.get()
4                 if  time.time()  -  self.connection_start_time  <
      self.STABILIZATION_DELAY: continue
5
6             if not line.startswith("DATA"):
7                     self._log(line  if  line.endswith('\n')  else
      line+'\n')
8
9             data = parse_data_line(line)
10            if data:
11                self._update_data_ui(data)
12                self._update_csv(data)
13                self._update_chart_data(data)
14
15        self.root.after(50, self._poll_queue)
```

Funkcja realizuje mechanizm asynchronicznego przetwarzania danych w architekturze Producent-Konsument:

1. Odbiór z kolejki: Funkcja sprawdza, czy wątek komunikacyjny (SerialReader) umieścił nowe pakiety danych w kolejce self.q. Dzięki zastosowaniu kolejki FIFO (First In, First Out), główny wątek interfejsu nie jest blokowany przez operacje wejścia/wyjścia (I/O).

2. Filtracja i Parsowanie: Surowe dane tekstowe są wstępnie filtrowane (ignorowanie śmieciowych danych tuż po nawiązaniu połączenia), a następnie przekazywane do parsera, który zamienia ciąg znaków (np. t=22.5) na słownik wartości zmiennoprzecinkowych.

3. Dystrybucja: Przetworzone dane są jednocześnie wysyłane do trzech niezależnych modułów:
   - Warstwa prezentacji: Aktualizacja etykiet tekstowych (_update_data_ui).
   - Warstwa archiwizacji: Zapis do pliku CSV, jeśli włączono nagrywanie (_update_csv).
   - Warstwa wizualizacji: Aktualizacja buforów danych dla wykresów (_update_chart_data).

4. Cykliczność: Funkcja wykorzystuje metodę root.after(50, ...), aby zaplanować swoje kolejne wywołanie za 50 ms. Pozwala to na płynne odświeżanie interfejsu bez zamrażania okna aplikacji, co miałoby miejsce w przypadku użycia standardowej pętli while True i funkcji sleep.

## 4.5 Obsługa trybu nocnego (Maszyna Stanów)

Ciekawym rozwiązaniem jest system edycji godzin nocy bezpośrednio z poziomu urządzenia. Zastosowano tu prostą maszynę stanów (NightEditState), która zmienia zachowanie przycisków w zależności od tego, czy użytkownik jest w trybie podglądu, czy edycji.

- Stan NIGHT_EDIT_OFF: Przyciski służą do zmiany trybu pracy i włączania światła.
- Stan NIGHT_EDIT_START / END: Te same przyciski służą do zwiększania/zmniejszania godziny, a wybrana wartość miga na ekranie OLED.

## 4.6 Opis działania najważniejszych algorytmów

1. Histereza i Debouncing: Przyciski są filtrowane programowo (funkcja checkPress), co zapobiega drganiom styków i błędnemu odczytowi wielokrotnych kliknięć.
2. Multitasking (Pseudo-wielozadaniowość): W Arduino nie użyto funkcji delay(). Zamiast tego system korzysta z funkcji millis(), co pozwala na jednoczesne miganie diodą alarmową, odtwarzanie dźwięku buzzera i komunikację z PC bez wzajemnego blokowania się procesów.
3. Dynamiczny wykres: Aplikacja PC wykorzystuje kolejkę deque o stałym rozmiarze (120 punktów). Gdy przychodzi nowa dana, najstarsza jest usuwana, co tworzy efekt "płynącego" wykresu w czasie rzeczywistym.

# 5. Testy i wyniki

## 5.1 Testy funkcjonalne części sprzętowej

Sprawdzono poprawność działania algorytmów sterujących oraz reakcję systemu na zmienne warunki środowiskowe. Symulację zmian temperatury i wilgotności zrealizowano poprzez manipulację napięciem na wejściach analogowych.

1. Scenariusz 1: System Alarmowy (Histereza)
   - Przebieg: Wymuszono wzrost odczytu temperatury powyżej 26.0°C (Warning), a następnie powyżej 30.0°C (Critical).
   - Wynik: Przy progu ostrzegawczym dioda LED zaczęła migać, a buzzer wydawał przerywany dźwięk. Po przekroczeniu stanu krytycznego oba sygnalizatory przeszły

w stan ciągły, a na ekranie OLED wyświetlił się komunikat „CRITICAL”. [Rys. 5.3]

- Wniosek: Logika alarmowa działa poprawnie i posiada priorytet nad innymi funkcjami.

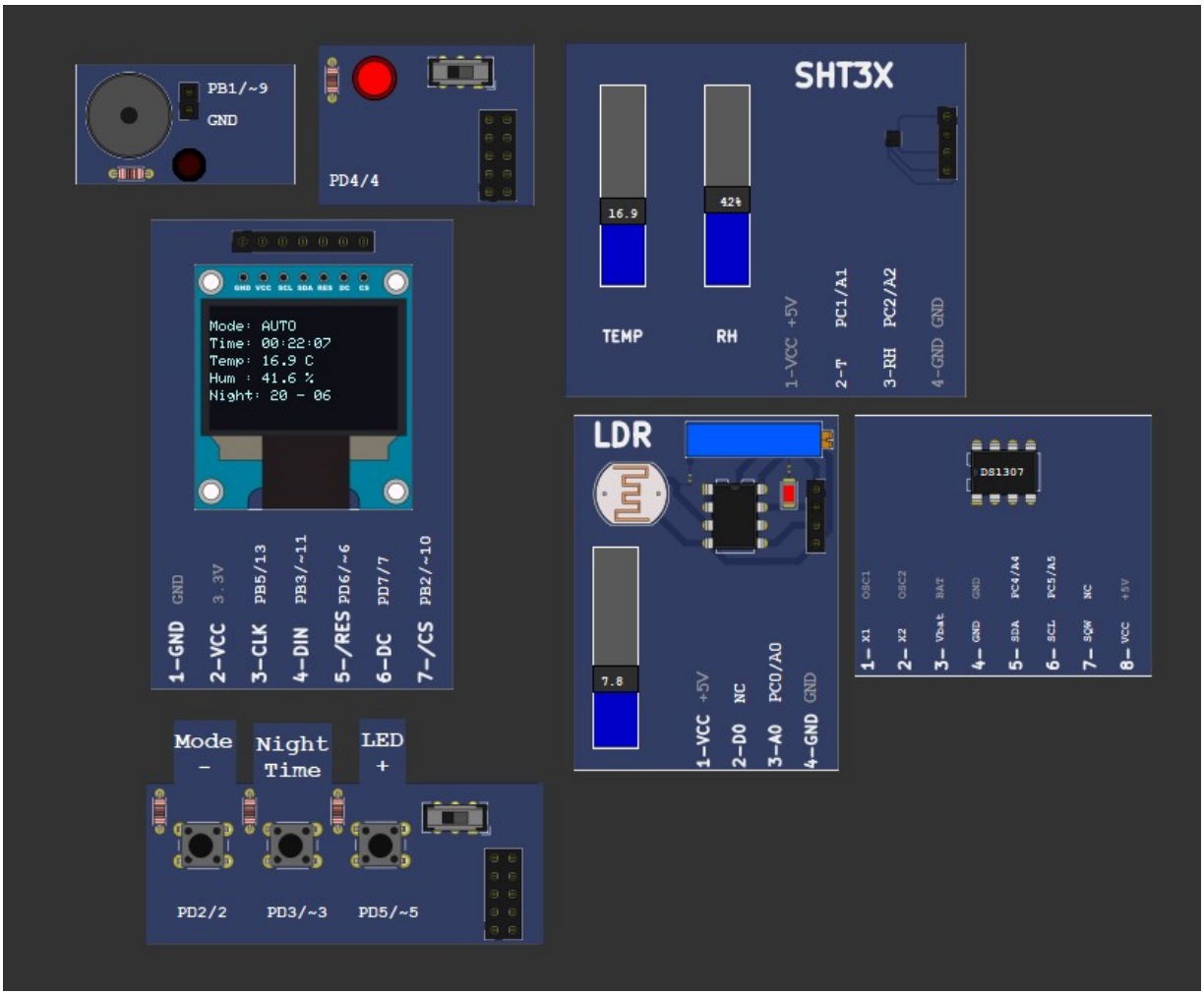
2. Scenariusz 2: Automatyka Oświetlenia (LDR)
   - Przebieg: W trybie AUTO zasłonięto fotorezystor, symulując zapadnięcie zmroku (wartość ADC < 50).
   - Wynik: Dioda LED (oświetlenie robocze) została włączona automatycznie. Odsłonięcie czujnika spowodowało jej wyłączenie. [Rys. 5.1]
   - Wniosek: Układ poprawnie interpretuje natężenie światła i steruje elementem wykonawczym.
3. Scenariusz 3: Interfejs Lokalny (OLED i Przyciski)
   - Przebieg: Przetestowano nawigację w menu edycji godzin nocnych ("Night Mode").
   - Wynik: Wciśnięcie przycisku BTN_NIGHT aktywuje tryb edycji (miganie cyfr na ekranie), a przyciski BTN_MODE / BTN_TOGGLE poprawnie inkrementują/dekrementują wartości godzin.

## 5.2 Testy aplikacji PC i komunikacji

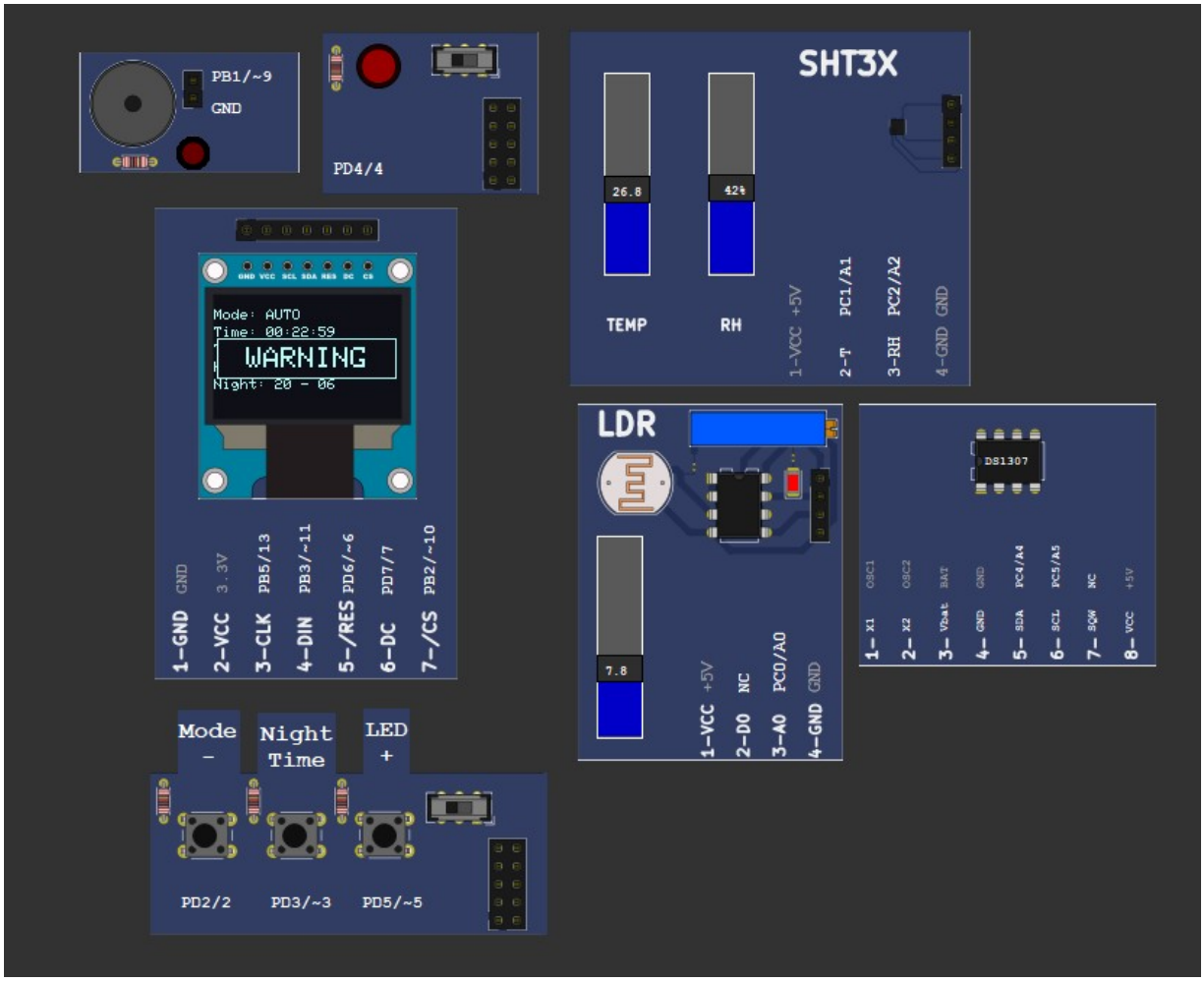
Testy te miały na celu weryfikację stabilności połączenia UART oraz poprawności wizualizacji danych.

1. Scenariusz 4: Wizualizacja danych w czasie rzeczywistym
   - Przebieg: Uruchomiono aplikację Python i nawiązano połączenie z portem COM. Obserwowano wykresy przy szybkich zmianach parametrów na czujnikach.
   - Wynik: Wykresy aktualizują się płynnie ("przesuwające się okno"), bez widocznych opóźnień. Interfejs aplikacji pozostaje responsywny dzięki zastosowaniu wielowątkowości (threading).
2. Scenariusz 5: Zdalne sterowanie
   - Przebieg: W aplikacji zmieniono tryb na manual oraz wymuszono wyłączenie LED przyciskiem "WYŁĄCZ".
   - Wynik: Dioda na płytce stykowej zgasła natychmiastowo, co potwierdza odebranie komend przez mikrokontroler. [Rys. 5.4]
3. Scenariusz 6: Rejestracja danych (Data Logging)
   - Przebieg: Uruchomiono funkcję nagrywania (przycisk "REC"), po 2 minutach zatrzymano zapis.
   - Wynik: Wygenerowano plik .csv, który poprawnie otworzono w arkuszu kalkulacyjnym. Wszystkie kolumny (czas, temperatura, wilgotność, stany flag) zawierały poprawne dane.

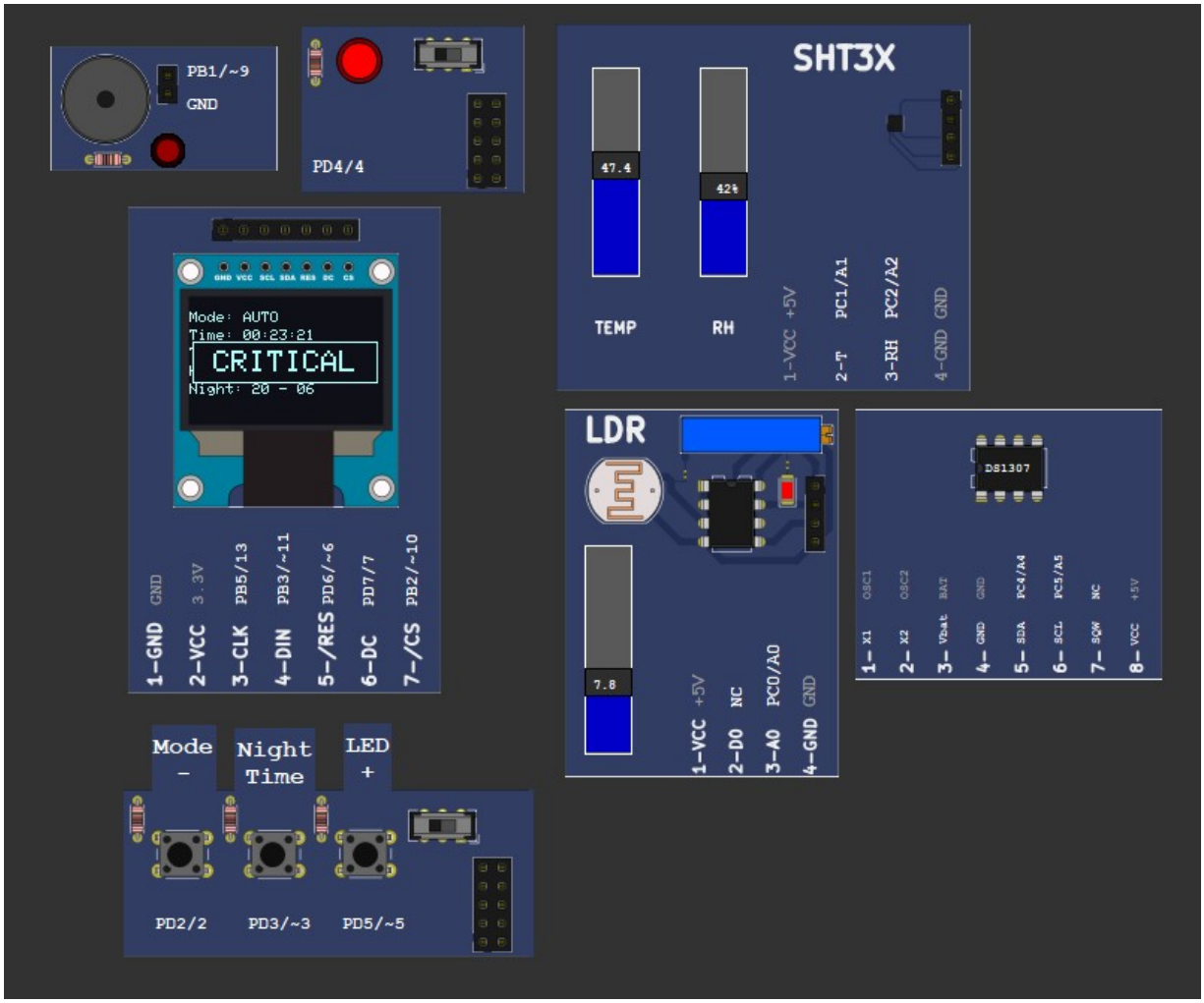
## 5.3 Wyniki przedstawione graficznie



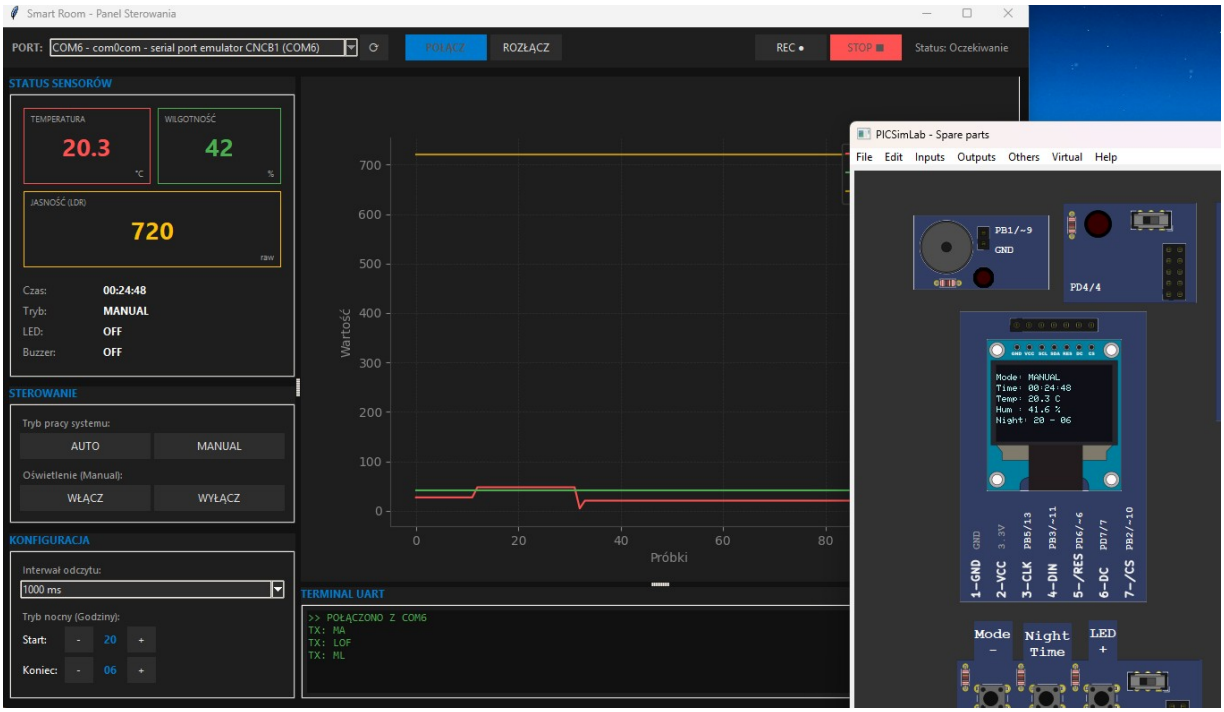

**Rys. 5.1.** Zrzut ekranu przedstawiający zapalony LED w trybie nocnym.




**Rys. 5.2.** Zrzut ekranu przedstawiający pracę układu w trybie „Warning”.




**Rys. 5.3.** Zrzut ekranu przedstawiający pracę układu w trybie „Critical”.




**Rys. 5.4.** Zrzut ekranu przedstawiający pracę układu w trybie manualnym przez oprogramowanie PC.

# 6. Podsumowanie

## 6.1 Ocena realizacji celu

Projekt Smart Room został zrealizowany zgodnie z pierwotnymi założeniami, osiągając pełną funkcjonalność w zakresie monitorowania środowiska i automatyzacji. Cel nadrzędny, jakim było stworzenie inteligentnego systemu reagującego na warunki zewnętrzne, został potwierdzony poprzez sprawne działanie algorytmu updateLogic, który bez opóźnień integruje dane z czujników temperatury, wilgotności oraz natężenia światła.

- **Niezawodność automatyzacji:** System poprawnie przełącza stan oświetlenia (LED) na podstawie progu fotorezystora oraz harmonogramu czasowego pobieranego z zegara RTC.
- **Bezpieczeństwo:** Zaimplementowanie dwustopniowego systemu ostrzegania (Warning/Critical) pozwala na skuteczną sygnalizację stanów niebezpiecznych, takich jak przegrzanie pomieszczenia, za pomocą zróżnicowanych sygnałów dźwiękowych i wizualnych.
- **Interdyscyplinarność:** Udało się stworzyć spójny ekosystem składający się z oprogramowania wbudowanego (C++) oraz aplikacji desktopowej (Python), co pozwala nie tylko na bieżącą obsługę, ale również na długofalową analizę danych dzięki modułowi zapisu do formatu CSV.

## 6.2 Możliwe usprawnienia

- Mimo pełnej funkcjonalności, system posiada potencjał do dalszego rozwoju w następujących obszarach:
- **Rozbudowa łączności bezprzewodowej:** Zastąpienie komunikacji przewodowej UART modułem Wi-Fi (np. ESP8266/ESP32). Pozwoliłoby to na przesyłanie danych do chmury (IoT) i sterowanie pokojem z poziomu aplikacji mobilnej z dowolnego miejsca na świecie.
- **Zaawansowana analityka danych:** Wprowadzenie bazy danych SQL (np. SQLite) zamiast plików CSV w aplikacji Python. Umożliwiłoby to generowanie bardziej złożonych raportów okresowych (np. średnie zużycie energii lub statystyki temperatury z całego miesiąca).
- **Inteligentne sterowanie klimatyzacją:** Dodanie przekaźników sterujących wentylatorem lub grzejnikiem. System mógłby wtedy nie tylko sygnalizować przekroczenie temperatur, ale aktywnie dążyć do utrzymania zadanych parametrów (tzw. termostat).
- **Personalizacja interfejsu:** Rozbudowa menu OLED o możliwość konfigurowania progów alarmowych (np. temperatury krytycznej) bezpośrednio z poziomu przycisków na urządzeniu, bez konieczności modyfikacji kodu źródłowego.
- **Optymalizacja energetyczna:** Wprowadzenie trybów uśpienia mikrokontrolera (Sleep Mode), co byłoby kluczowe w przypadku chęci zasilania układu bateryjnie.