

WYŻSZA SZKOŁA BANKOWA W POZNANIU

Wydział Finansów i Bankowości

Łukasz Piasecki

**Projekt systemu nadzoru i sterowania dla wybranych systemów
bytowych przedsiębiorstwa.**

Praca magisterska

Promotor

Dr hab. Wojciech Rudziński

Poznań 2021

Spis treści

Rozdział 1 Wstęp	3
Rozdział 2 Założenia teoretyczne i wyjaśnienie pojęć.	4
2.1 Nadzór i sterownie	4
2.2 Systemy bytowe w ogólności	4
2.3 Kontrola dostępu.....	4
2.4 Zawiadamianie o pożarze	5
2.5 System antywłamaniowy.....	5
2.6 Czujniki zalania i zadymienia	5
Rozdział 3 Zastosowane technologie.....	7
3.1 Komputer Raspberry Pi 4.....	7
3.2 System Raspbian	8
3.3 GPIO	8
3.4 Przekaznik 5V	9
3.5 Zasilacz buforowy 12V.....	10
3.6 Czytnik RFID RC522.....	11
3.7 Kontaktron CMD14.....	12
3.8 Zwora otwarcia drzwi	12
3.9 Buzzer.....	13
3.10 Czujnik dymu i gazów łatwopalnych	14
3.11 Detektor ruchu PIR HC-SR501	16
3.12 Czujnik opadów	17
3.13 Wyświetlacz LCD	17
3.14 Czujnik temperatury i wilgotności DHT11	19
Rozdział 4 Praktyczna realizacja projektu.....	21
4.1 Montaż i łączenie elementów projektu	21
4.2 Python	22
4.3 Biblioteka GPIO	23
4.4 Biblioteka I2C_LCD_driver	25
4.5 Biblioteka adafruit_dht	25
4.6 Biblioteka datetime	26
4.7 Biblioteka mfrc522	27
4.8 Implementacja	27
Rozdział 5 Testowanie rozwiązania oraz spostrzeżenia	36
5.1 Wyzwalanie alarmów pojedynczo	36

5.2 Wyzwalanie więcej niż jednego alarmu jednocześnie	37
5.3 Kasowanie alarmów	Błąd! Nie zdefiniowano zakładki.
5.4 Odporność na sabotaż	37
Rozdział 6 Zakończenie.....	40
Bibliografia	40

Rozdział 1

Wstęp

Tematyka inteligentnych budynków w ostatnich latach budzi spore zainteresowanie. Trend ten widoczny jest zarówno w budownictwie mieszkaniowym jak i w przemyśle. Wpływ na to mają rosnąca ilość dostępnej w Internecie wiedzy na ten temat oraz coraz bardziej dostępna cena podzespołów niezbędnych do realizacji różnych projektów Internetu Rzeczy.

Rozdział 2

Założenia teoretyczne i wyjaśnienie pojęć.

2.1 Nadzór i sterownie

Systemy nadzoru i sterowania komunikują się z czujnikami i urządzeniami wykonawczymi, aby dostarczyć informacji o bieżącym stanie obiektu (nadzór) lub dokonać zmiany tego stanu (sterowanie). Czynności te mogą dokonywać się w czasie rzeczywistym, w określonych chwilach lub tylko w reakcji na określone zdarzenie. Obie te operacje można wykonywać lokalnie, przy bezpośrednim dostępie do podzespołów spełniających te funkcje lub zdalnie, nawet na innym kontynencie.

2.2 Systemy bytowe w ogólności

System bytowy to taki, który jest związany z działaniem lub funkcjonowaniem człowieka, przedsiębiorstwa lub procesu. Zaliczyć do nich można:

- Systemy kontroli dostępu
- Systemy powiadamiania o pożarze oraz wczesnej detekcji dymu
- Systemy antywłamaniowe oraz alarmowe
- Systemy ochrony przed zalaniem, zaccadzeniem lub zagazowaniem

Każdy z wyżej wymienionych zostanie krótko scharakteryzowany w dalszej części pracy.

2.3 Kontrola dostępu

Systemami kontroli dostępu nazywamy oprogramowanie oraz urządzenia pozwalające na ograniczenie (kontrolę) dostępu osób do kontrolowanego obiektu. Bardzo częstym jest tutaj podział obiektu na mniejsze części (strefy) i zróżnicowanie użytkowników systemu lub ich grup pod względem możliwości dostępu do tych stref. W rozwiązaniach bardziej zaawansowanych możliwe jest też ustalenie dni i godzin, w których wstęp jest dozwolony lub zabroniony poszczególnym grupom, zdefiniowanie dni świątecznych w danym roku kalendarzowym, kontrola ilości osób w danej strefie (np. za pomocą kołowrotu), tryby wejścia komisyjnego (konieczna autoryzacja dwóch lub więcej uprawnionych użytkowników) oraz specjalny rodzaj wejścia tzw. wymuszony lub pod przymusem. Polega on na tym na takim sposobie uzyskania dostępu (innym kodem, naciskając ukryty

przycisk, przykładając kartę na dłużej lub dwukrotnie) który jednocześnie wysyła do systemu informację, że użytkownik został zmuszony do tych działań i może być w niebezpieczeństwie oraz potrzebować pomocy. Tryb ten można spotkać nie tylko w sejfach i kasach pancernych, lecz także w przemysłowych systemach kontroli dostępu do budynków. Do uwierzytelnienia w tego typu systemach można użyć zarówno przedmiotów takich jak karty lub breloki w technologii RFID (rzadziej NFC), danych biometrycznych jak odciski linii papilarnych, skan siatkówki oka lub geometria twarzy, jak i pilotów radiowych lub aplikacji w telefonie.

2.4 Zawiadamianie o pożarze

Systemami zawiadamiania o pożarze nazywamy zespoły urządzeń oraz programów umożliwiających automatyczną detekcję pożaru (czujniki dymu i temperatury), proste i szybkie ręczne informowanie o pożarze wykrytym przez człowieka (przyciski ręcznego ostrzegania pożarowego tzw. ROP) oraz alarmowanie o wykrytym zagrożeniu (sygnalizatory akustyczne i wizualne, moduły komunikacji przewodowej i bezprzewodowej wysyłające informację o pożarze do wskazanych odbiorców). Urządzenia te powinny cechować się wysoką sprawnością i niezawodnością, min. spełniać normy określające, ile czasu dane urządzenie powinno wytrzymać przy ekspozycji na otwarty ogień.

2.5 System antywłamaniowy

System antywłamaniowy służy do ochrony mienia przed grabieżą, zniszczeniem oraz włamaniem. Podobnie jak opisany wcześniej system powiadamiania o pożarze podzielić go można na elementy służące do wykrycia zdarzenia, takie jak czujniki ruchu oparte o detekcję promieniowania podczerwonego lub ultradźwiękowe oraz urządzenia sygnalizacyjne zarówno bezpośrednio informujące o włamaniu światłem i dźwiękiem jak i wysyłające komunikaty do zdefiniowanych adresatów np. mailem lub sms-em. Opcjonalnym elementem są podobnie jak w systemach kontroli dostępu komponenty służące do uruchomienia tzw. „cichego alarmu” czyli powiadomienia o niebezpieczeństwie bez informowania o tym osób naruszających strefę, np. w formie przycisku na pilocie, połączenia na odpowiedni numer lub aplikacji w telefonie.

2.6 Czujniki zalania i zadymienia

Systemy ochrony przed zalaniem oraz umożliwiające detekcję gazów łatwopalnych lub niebezpiecznych jak metan lub tlenek węgla zwany potocznie czadem zazwyczaj są

zintegrowane w jedno urządzenie zawierające czujnik oraz sygnalizator, zwykle dźwiękowy. W tego typu systemach bardzo dużą rolę odgrywa czas reakcji zwykle liczony w minutach a działanie ogranicza się w pierwszym rzędzie do natychmiastowego opuszczenia obszaru objętego alarmem. Również ze względu na stosunkowo mały zasięg oddziaływania czynników takich jak zalanie wodą lub zagazowanie w zamkniętych pomieszczeniach lepszym rozwiązaniem wydaje się większa ilość pojedynczych czujników i sygnalizatorów, natomiast centralny system ostrzegania, choć bardzo ważny, to ze względu na wspomnianą konieczność bardzo szybkich działań w obszarze wystąpienia alarmu staje się kwestią drugorzędną.

Rozdział 3

Zastosowane technologie

3.1 Komputer Raspberry Pi 4

Platformą sprzętową na której uruchamiane będzie oprogramowanie sterujące podzespołami systemu jest mikrokomputer Raspberry Pi 4B wyposażony w 4GB pamięci RAM z systemem operacyjnym Raspbian. Projekt Raspberry Pi rozpoczął się w 2012 roku i od początku dedykowany jest dla automatyków, robotyków oraz programistów, zarówno doświadczonych jak i tych dopiero rozpoczynających naukę. Obecnie jest to rozwiązanie powszechnie znane i stosowane zarówno w hobbystycznie realizowanych projektach jak i rozwiązaniach przemysłowych. Znaczny wpływ na popularność rozwiązania ma relatywnie niska cena urządzenia oraz duża ilość ogólnodostępnych materiałów.



Rysunek 3. 1 Komputer Raspberry Pi 4B

Źródło: <https://botland.com.pl/>

3.2 System Raspbian

System do niedawna zwany Raspbianem, a od niedawna Raspberry Pi OS (The Raspberry Pi Foundation, 2021) to jeden z najczęściej używanych i najbardziej znanych systemów operacyjnych na platformę Raspberry Pi. Jest to system oficjalnie uznany przez The Raspberry Pi Foundation, oparty na Debianie, będącym jedną z dystrybucji Linuxa. Działa w architekturze ARM, wykorzystuje środowisko graficzne LXDE oraz PIXEL.

3.3 GPIO

Wejścia-wyjścia ogólnego przeznaczenia (ang. General Purpose Input Output, GPIO) to zestaw pinów stanowiący interfejs komunikacyjny między komputerem a podłączonymi do niego zewnętrznymi urządzeniami. Raspberry Pi posiada takich pinów 40 i umożliwia pracę z nimi w kilku układach, z których najpopularniejsze to GPIO oraz BOARD.

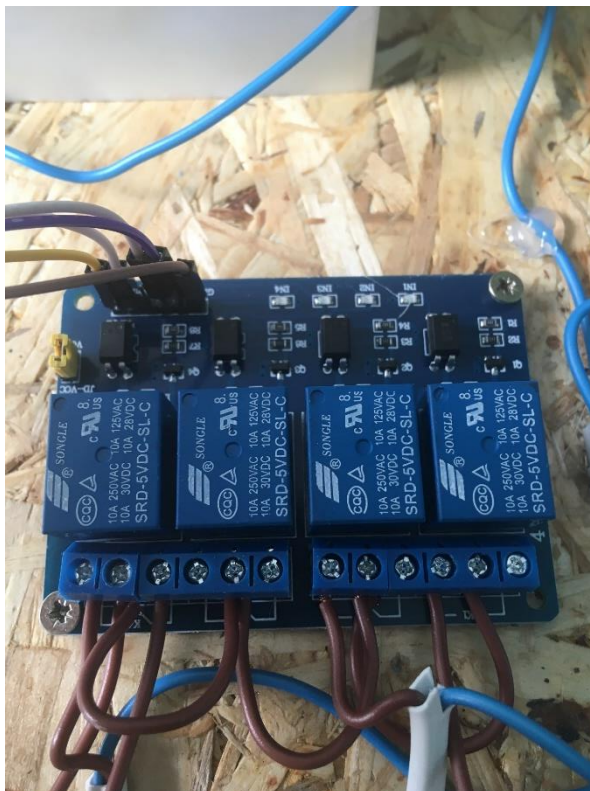
3v3 Power	1		2	5v Power
GPIO 2 (I2C1 SDA)	3		4	5v Power
GPIO 3 (I2C1 SCL)	5		6	Ground
GPIO 4 (GPCLK0)	7		8	GPIO 14 (UART TX)
Ground	9		10	GPIO 15 (UART RX)
GPIO 17	11		12	GPIO 18 (PCM CLK)
GPIO 27	13		14	Ground
GPIO 22	15		16	GPIO 23
3v3 Power	17		18	GPIO 24
GPIO 10 (SPI0 MOSI)	19		20	Ground
GPIO 9 (SPI0 MISO)	21		22	GPIO 25
GPIO 11 (SPI0 SCLK)	23		24	GPIO 8 (SPI0 CE0)
Ground	25		26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27		28	GPIO 1 (EEPROM SCL)
GPIO 5	29		30	Ground
GPIO 6	31		32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33		34	Ground
GPIO 19 (PCM FS)	35		36	GPIO 16
GPIO 26	37		38	GPIO 20 (PCM DIN)
Ground	39		40	GPIO 21 (PCM DOUT)

Rysunek 3. 2 Układ pinów GPIO w Raspberry Pi 4B

Źródło: <https://pinout.xyz/>

3.4 Przekaznik 5V

Przekaznik to proste urządzenie elektroniczne, którego zadaniem jest zmiana stanu po wystąpieniu określonych warunków. W projekcie zastosowano układ 4 przekazników firmy Songle zintegrowanych na jednej płytce sterowanych napięciem 5V DC, gdzie stan niski (logiczne 0) powoduje wyzwolenie przekaznika. Maksymalny prąd jaki może być podłączany do styków roboczych urządzenia to 10A, o napięciu do 250V dla prądu zmiennego i do 30V dla stałego. W stanie spoczynku styk NC (ang. Normal close) jest zwarty do styku ogólnego, natomiast styk NO (ang. Normal open) jest rozarty. W przypadku wysterowania przekaznika, które powodowane jest podaniem na pin sterujący sygnału niskiego następuje odwrócenie sytuacji, tj. styk NC rozwiera się, natomiast styk NO zostaje zwarty do styku ogólnego. Niezwykle prosta zasada działania czyni urządzenie niezwykle wszechstronnym włącznikiem zasilania i umożliwia sterowanie znacznie wyższymi napięciami aniżeli możliwe by to było przy zasilaniu urządzeń bezpośrednio z pinów GPIO Raspberry Pi (max. 5V na stałe, max. 3,3V sterowane).



Rysunek 3. 3 Przekaznik zastosowany w projekcie

3.5 Zasilacz buforowy 12V

Do zasilania systemu zastosowano zasilacz buforowy 12V firmy Pulsar. Zestaw składa się z transformatora obniżającego napięcie zmienne z sieci do wartości 17V oraz mostka prostującego, na wyjściu, którego pojawia się napięcie stałe o nominalnej wartości 12V. Dodatkowo jest on wyposażony w styki ładowania, umożliwiające podłączenie akumulatora, co pozwala na pracę układu nawet przy zaniku zasilania sieciowego oraz styk TAMPER typu NC, który rozwiera się w przypadku zwolnienia przycisku, który normalnie jest dociskany przez obudowę urządzenia. Pozwala to na wysłanie informacji o otwarciu obudowy, co może ułatwić zapobieganie sabotażowi.



Rysunek 3. 4 Zasilacz obniżający napięcie zmienne do 17V

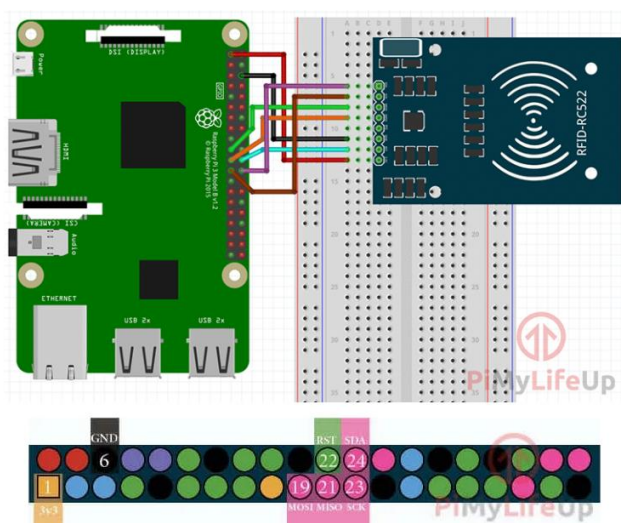


Rysunek 3. 5 Prostownik zmieniający prąd zmienny o obniżonym napięciu na prąd stały o napięciu nominalnym 12V

3.6 Czytnik RFID RC522

Do obsługi kart oraz breloków działających w technologii RFID zastosowany został czytnik RC522. Urządzenie podłączono do pinów GPIO przy użyciu interfejsu SPI (ang. Serial Peripheral Interface). Pewien problem techniczny stanowił fakt, że urządzenie fabrycznie nie ma przylutowanych kołków goldpin i konieczne jest wykonanie tego we własnym zakresie. Urządzenie działa za pomocą interfejsu SPI (ang. *Serial Peripheral Interface*). Łącznie czytnik ma 8 pinów do których podłączamy:

- SDA (ang. *Serial Data Signal*)
- SCK (ang. *Serial Clock*)
- MOSI (ang. *Master Out Slave In*)
- MISO (ang. *Master In Slave Out*)
- IRQ (ang. *Interrupt Request*) – nie podłączone
- GND (ang. *Ground*)
- RST (ang. *Reset*)
- 3,3V - zasilanie



Rysunek 3. 6 Sposób połączenia czytnika RFID

Komunikacja z kartami, brelokami i innymi urządzeniami RFID odbywa się za pomocą pola elektromagnetycznego 13,56 MHz. Dużą zaletą tego modelu jest jego wysoka dostępność i relatywnie niska cena, co w połączeniu z faktem, że w realizacji rzeczywistego systemu kontroli dostępu w budynku może znajdować się od kilku do kilkunastu chronionych zamkami magnetycznymi przejść czyni go niezwykle atrakcyjnym rozwiązaniem.

3.7 Kontaktron CMD14

Jako czujnik otwarcia drzwi zastosowano kontaktron CMD14. Jest to urządzenie o bardzo prostej zasadzie działania. Jeden z segmentów kontaktronu jest magnesem. Jak długo urządzenie jest w polu magnetycznym (zwarcie elementów) obwód pozostaje zwarty. Gdy drzwi zostają otwarte, rozwarcie elementów powoduje przerwanie pola magnetycznego, co skutkuje przerwaniem obwodu. Zwarcie bądź rozwarcie obwodu monitorowane jest na jednym z pinów GPIO i dostarcza do systemu informacji o tym czy drzwi są zamknięte.

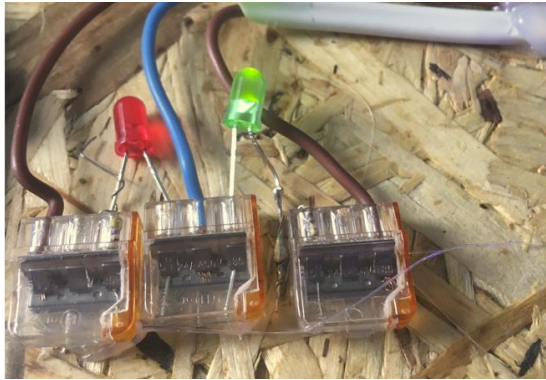


Rysunek 3. 7 Kontaktron zastosowany w projekcie

3.8 Zwora otwarcia drzwi

Ze względu na stosunkowo wysoką cenę elektrorygli oraz zwor elektromagnetycznych w projekcie badawczym zdecydowano się zastąpić ten element przez zestaw dwóch diod led koloru zielonego i czerwonego. Dioda czerwona podłączona jest do styku NC (ang. *Normal Close*) a zielona do styku NO (ang. *Normal Open*). Wysterowanie przekaźnika powoduje otwarcie styku normalnie zamkniętego i zwarcie styku normalnie otwartego, co powoduje chwilowe zgaszenie palącej się na stałe diody i zapalenie zgaszonej. Zestaw ten dla celów naukowych w pełni zastępuje prawdziwy zamek elektromagnetyczny. Kwestię doboru tego, czy zamek jest otwarty czy też zamknięty podczas zaniku napięcia należy rozważyć podczas wdrażania systemu kontroli dostępu w konkretnym budynku, z zachowaniem obowiązujących przepisów ewakuacyjnych, aby jednocześnie należyście zabezpieczyć system przez sabotażem poprzez odcięcie zasilania przez potencjalnych

włamywaczy a jednocześnie nie odciąć drogi ucieczki w przypadku ogłoszenia np. alarmu przeciwpożarowego.



Rysunek 3. 8 Zestaw dwóch żarówek imitujących działanie zwory elektromagnetycznej.

3.9 Buzzer

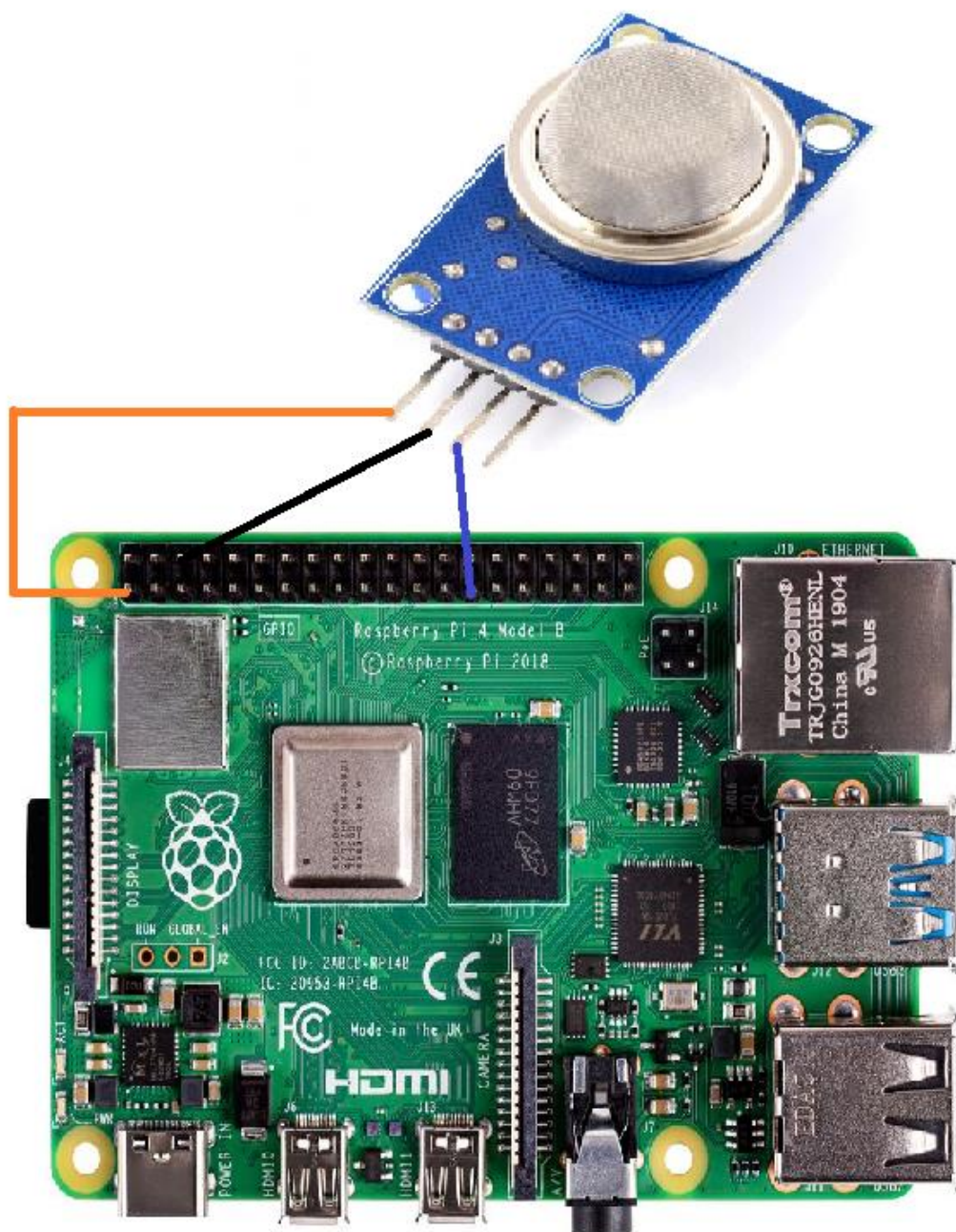
System sygnalizacji o uruchomieniu alarmu stanowi buzzer wydający sygnał modulowany oraz pulsująca dioda LED koloru żółtego. Efekt pulsowania i modulacji dźwięku uzyskano przez cykliczną zmianę stanu przekaźnika, do którego podłączono zarówno diodę jak i buzzer. Jak zostało wspomniane we wcześniejszych rozdziałach napięcie zasilające systemu wynosi 12V, w związku z czym zastosowany został buzzer przystosowany do takiego napięcia, choć na etapie testów stwierdzono, że również urządzenie o nominalnym napięciu 5V działa bezawaryjnie przy zasilaniu 12V.



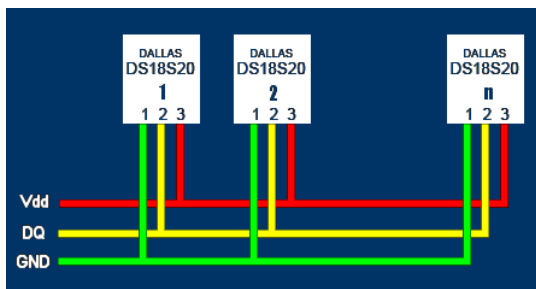
Rysunek 3. 9 Sygnalizator uruchomienia alarmu składający się z buzzera oraz diody LED

3.10 Czujnik dymu i gazów łatwopalnych

Do detekcji dymu oraz niebezpiecznych gazów zastosowany został czujnik MQ-2. Zgodnie z notą katalogową czujnik zasilany jest napięciem 5V DC i może wykryć stężenia LPG, dymu, alkoholu, propanu, wodoru, metanu i tlenku węgla w zakresie od 200 do 10000 ppm. Czujnik posiada zarówno wyjście cyfrowe jak i analogowe oraz pokrętko umożliwiające skalibrowanie jaka ilość cząsteczek gazu ma uruchomić alarm. Łączy się on za pomocą interfejsu 1-Wire, wykorzystującego tylko jedną linię danych.



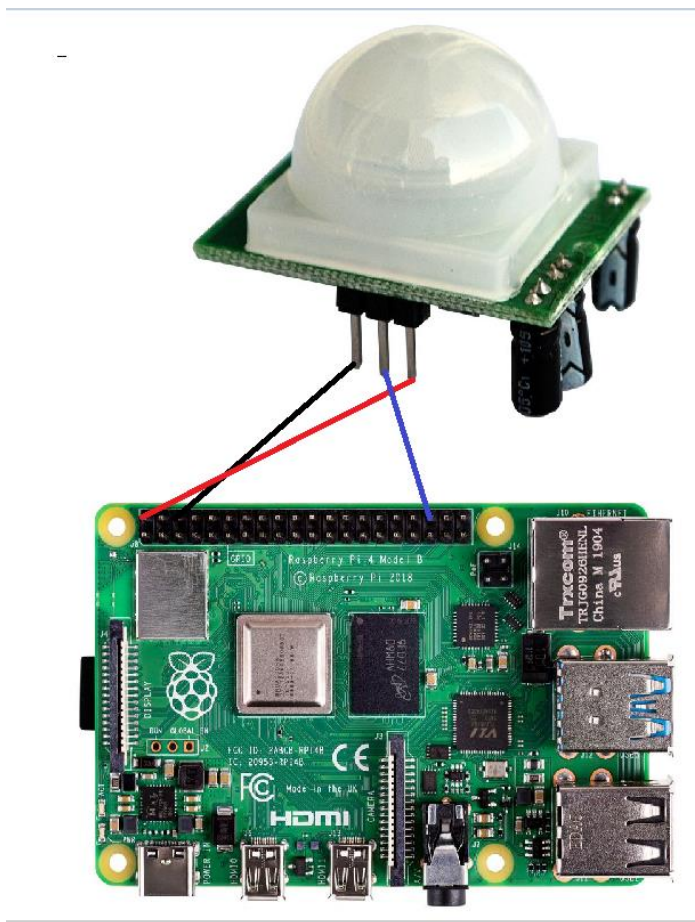
Rysunek 3. 10 Podłączenie czujnika MQ-2



Rysunek 3. 11 Schemat podłączenia urządzeń za pomocą interfejsu 1-Wire

3.11 Detektor ruchu PIR HC-SR501

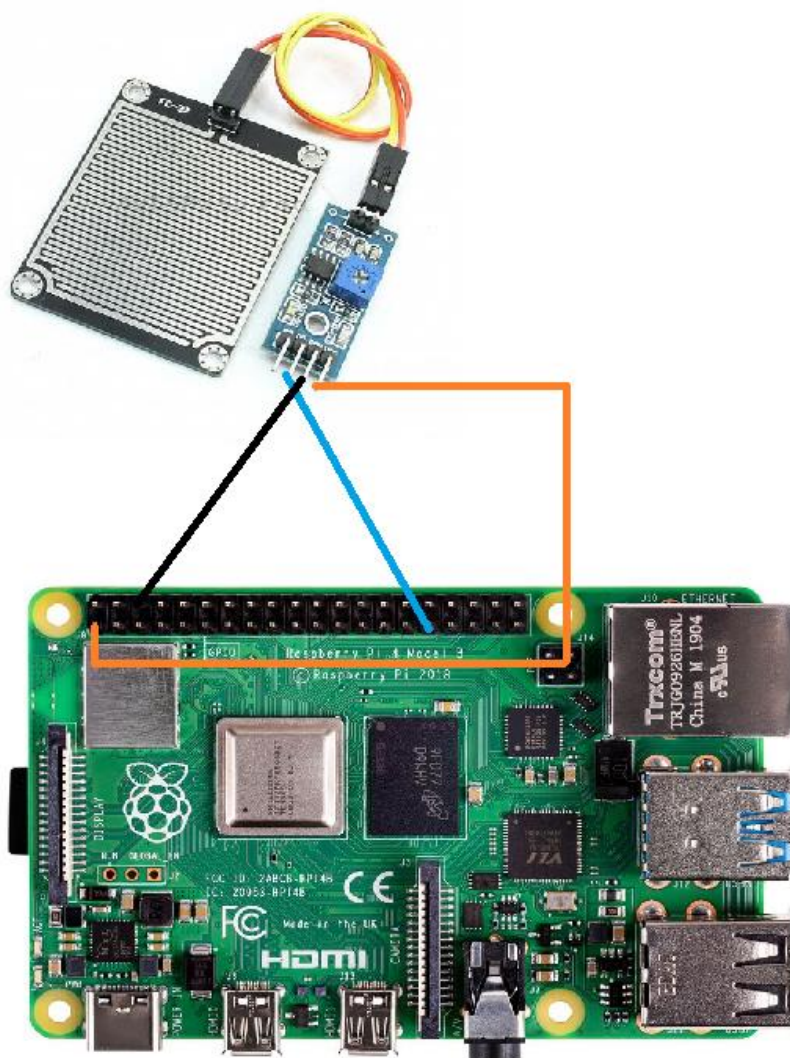
Jako czujnik ruchu wykorzystany został sensor promieniowania podczerwonego HC-SR501. Urządzenie posiada dwa pokrętki umożliwiające kalibrację czułości oraz czasu, przez który musi trwać ruch, aby wyzwolić sygnał na wyjściu czujnika.



Rysunek 3. 12 Podłączenie czujnika PIR

3.12 Czujnik opadów

Do wykrycia zalania układu wodą lub inną cieczą zastosowany został czujnik o oznaczeniu HL-83, choć nie udało się odnaleźć jego fabrycznej nazwy ani nazwy firmy produkującej go. Urządzenie posiada pokrętko umożliwiające skalibrowanie jaka ilość cieczy na sondzie ma wyzwolić alarm. Podobnie jak czujnik dymu posiada zarówno wyjście cyfrowe jak i analogowe.



Rysunek 3. 13 Podłączenie czujnika zalania.

3.13 Wyświetlacz LCD

Do przekazywania informacji użytkownikowi końcowemu służyć będzie wyświetlacz LCD 2 x 16 podłączony do Raspberry interfejsem I²C.

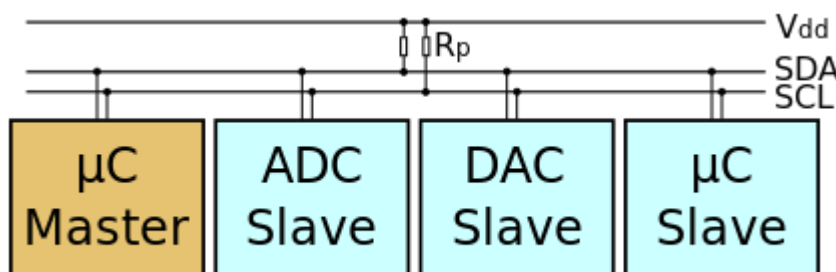


Rysunek 3. 14 Wyświetlacz LCD zastosowany w projekcie

Sama magistrala, opracowana przez przedsiębiorstwo Philips jest szeregową oraz dwukierunkową, znaną także pod drugą nazwą, będącą angielskim akronimem określenia „pośredniczenie pomiędzy układami scalonymi” (ang. *Inter-Integrated Circuit, IIC*). Komunikacja odbywa się za pomocą dwóch linii transmisyjnych:

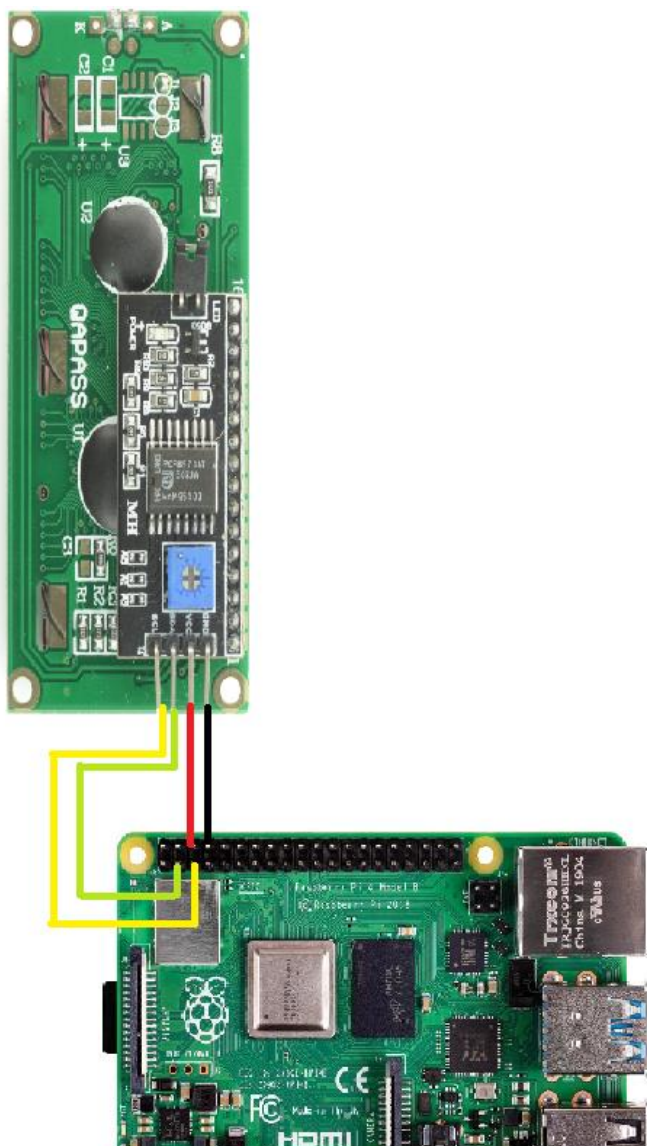
- Linia danych (ang. *Serial Data Line, SDA*)
- Linia zegara (ang. *Serial Data Clock, SCA*)

Standard pracuje w dwóch pierwszych warstwach modelu ISO/OSI, fizycznej i łącza danych. Długość połączenia uwarunkowana jest maksymalną pojemnością elektryczną linii i wynosi kilka metrów. Ilość możliwych do jednoczesnego podłączenia urządzeń pierwotnie wynosiła 112, obecnie jest to ponad tysiąc urządzeń.



Rysunek 3. 15 Schemat podłączenia do magistrali I2C

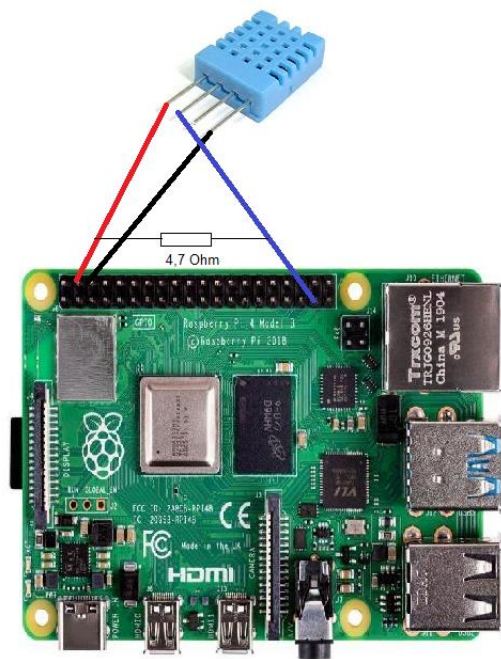
W praktyce połączenie z wyświetlaczem realizowane było za pomocą 4 przewodów, z których dwa to wspomniane tu już linie danych oraz zegarowa, a dwie kolejne stanowią zasilanie.



Rysunek 3. 16 Podłączenie wyświetlacza LCD za pomocą interfejsu I2C

3.14 Czujnik temperatury i wilgotności DHT11

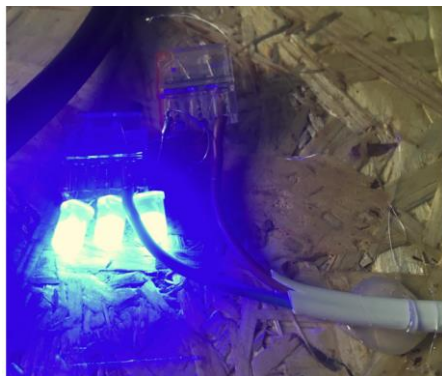
Jako termometr i czujnik wilgotności powietrza zastosowano czujnik DHT11 powszechnie używany w prostych projektach z zakresu Internetu rzeczy. W praktycznym zastosowaniu do 3 z 4 nóżek DHT11 podłączymy zasilanie, masę oraz linię danych. Dobrym zwyczajem jest połączenie linii danych z zasilającą (zazwyczaj +5V) rezystorem 4,7 Ohm.



Rysunek 3. 17 Podłączenie DHT11 do Raspberry Pi

3.15 Oświetlenie

Oświetlenie w budynku imitowane jest przez diody led koloru niebieskiego podłączone do styku normalnie otwartego przekaźnika, co umożliwia kontrolowanie ich uruchamiania. Mając możliwość sterowania oświetleniem można przyjąć wiele pojedynczych lub kombinowanych warunków uruchomienia i wyłączenia światła, jak np. wykrycie ruchu, określona godzina, obecność w budynku wg. danych systemu kontroli dostępu lub jawne polecenie operatora systemu poprzez wciśnięcie przycisku lub zdalną komendę.



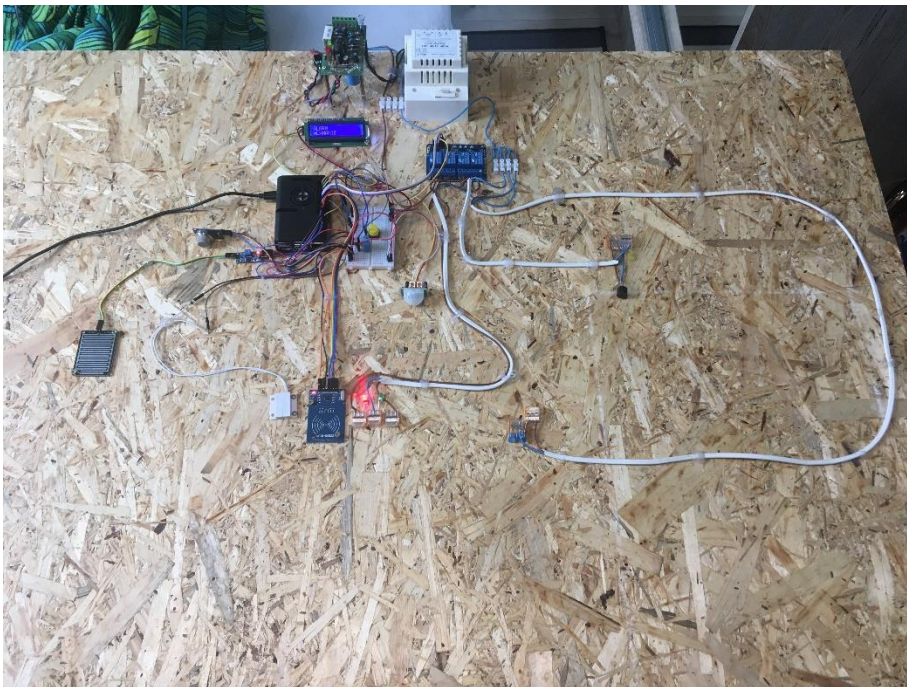
Rysunek 3. 18 Diody LED imitujące oświetlenie.

Rozdział 4

Praktyczna realizacja projektu

4.1 Montaż i łączenie elementów projektu

Układ został zmontowany na płycie wiórowej OSB, poszczególne elementy przytwierdzone zostały mechanicznie za pomocą wkrętów oraz kleju. Połączenia elektronicznie wykonano w kilku różnych technologiach. Były to połączenia lutowane, przy użyciu kostek wago oraz tradycyjnych zaciskowych a także (głównie do pinów GPIO w Raspberry) za pomocą kołków goldpin. Z uwagi na tymczasowy charakter instalacji, będącej projektem badawczym zastosowano powyższe rozwiązania, mając na uwadze niską cenę oraz ogólną dostępność komponentów potrzebnych do montażu. W przypadku zastosowania rozwiązania produkcyjnie konieczna byłaby analiza środowiska, w którym układ ma pracować. Należałoby wziąć pod uwagę takie czynniki jak np. materiał, z którego wykonano ściany lub podłogę, aby określić sposób montażu trasy kablowej lub w przypadku elementów będących na zewnątrz budynku, czynniki atmosferyczne przed którym należy chronić urządzenia, jak chociażby zastosowanie hermetycznych puszek w przypadku ryzyka wystąpienia opadów.



Rysunek 4. 1 Prototyp sprzętowej części systemu.

4.2 Python

Do implementacji projektu zastosowano wieloparadygmatowy język programowania Python. Powstał on w latach 90-tych ubiegłego stulecia, mając zastąpić język ABC, przede wszystkim wzbogacając go o rozszerzalność i obsługę wyjątków. Pierwsza wersja oznaczona numerem wersji 0.9 została zaprezentowana przez zespół pod przewodnictwem Guido van Rossuma w roku 1991. Sama nazwa języka wbrew stosowanemu powszechnie logo nie wzięła się od nazwy gatunku węża, lecz od emitowanego od lat 70-tych ubiegłego stulecia serialu pt. „Latający Cyrk Monty Pythona”. Wersja 1.0 udostępniona została w 1994, natomiast 2.0 w 2000 roku. Ważną datą dla historii rozwoju tego języka jest rok 2008, kiedy zdecydowano o rozdzieleniu projektu na dwie gałęzie-wersje programu: 2 oraz 3. Oficjalnie wersja 2 przestała być wspierana w roku 2020, tym samym wersja 3 jest jedyną oficjalnie rozwijaną wersją. Organizacją odpowiedzialną za rozwój języka jest niedochodowa fundacja Python Software Foundation (PSF). Sam proces rozwoju języka prowadzony jest przy zastosowaniu PEP (Python Enhancement Proposal). Jest to dokument zawierający propozycje zmian w języku, poddawany pod dyskusję społeczności programistów, która może zaopiniować wdrożenie lub odrzucenie propozycji. Jednym z najważniejszych dokumentów tego typu jest PEP8, zawierający propozycję organizacji składni, m.in. separację bloków kodu źródłowego za pomocą wcięć oraz kończenie instrukcji nową linią, co czasem powoduje nazywanie Pythona „językiem bez średników”, w odróżnieniu od składni popularnych języków programowania wysokiego poziomu jak C# czy Java, w których separacja linii kodu odbywa się za pomocą średnika, a grupowanie poleceń znakiem nawiasu klamrowego „{ }”. Inną ciekawą cechą odróżniającą Python od wspomnianych tu języków jest dynamiczne typowanie, czyli brak konieczności jawnego zadeklarowania typu zmiennej, błędnie określanego czasami przez początkujących programistów jako brak typów w Pythonie. W rzeczywistości typ zmiennej przydzielany jest w momencie inicjalizacji zmiennej wartością. Kolejną różnicą jest brak kompilacji, jest to bowiem język interpretowalny. Umożliwia to uruchomienie go w dowolnym środowisku obsługującym interpreter, ale w porównaniu z kompilowaną Javą czyni go nieco wolniejszym. Początkowo określany jako język skryptowy, obecnie przy zachowaniu tej pierwotnej funkcji ma bardzo wiele zastosowań, używany do programowania obiektowego, aplikacji Internetowych, jest także niezwykle popularny w Internecie Rzeczy na platformach sprzętowych Raspberry czy BeagleBoard.



Rysunek 4. 2 Logo języka Python

4.3 Biblioteka GPIO

Jedną z cech, które powodują popularność Pythona na Raspberry Pi jest duża ilość bibliotek wspierających podzespoły powszechnie stosowane w automatyce, robotyce czy Internecie Rzeczy. Podstawą i punktem wyjścia dla obsługi większości tych urządzeń jest obsługa opisanego już pokrótce interfejsu GPIO. Jest to biblioteka z otwartym kodem źródłowym, łatwo dostępna np. za pomocą managera pakietów *pip3*.

```
pip3 install RPi.GPIO
```

Rysunek 4. 3 Instalacja biblioteki GPIO w systemie Raspbian

Dzięki niej możliwe jest przyporządkowanie fizycznym pinom GPIO numerów wg. jednej z dwóch głównych konwencji, tj. BCM lub BOARD. W sposobie numeracji określanym słowem BOARD piny numerowane są wg. faktycznej kolejności na płycie stykowej, natomiast w domyślnym dla biblioteki standardzie BCM numeracja jest zgodna z ustaloną przez firmę Broadcom, będącą producentem procesora. Mimo wspomnianych tutaj ustawień domyślnych w trybie BCM dobrą praktyką jest jawne zadeklarowanie trybu celem uniknięcia pomyłki lub nieporozumienia. Samą numerację w systemie Raspbian można łatwo sprawdzić poleceniem *pinout*.

J8:

3V3	(1)	(2)	5V
GPIO2	(3)	(4)	5V
GPIO3	(5)	(6)	GND
GPIO4	(7)	(8)	GPIO14
GND	(9)	(10)	GPIO15
GPIO17	(11)	(12)	GPIO18
GPIO27	(13)	(14)	GND
GPIO22	(15)	(16)	GPIO23
3V3	(17)	(18)	GPIO24
GPIO10	(19)	(20)	GND
GPIO9	(21)	(22)	GPIO25
GPIO11	(23)	(24)	GPIO8
GND	(25)	(26)	GPIO7
GPIO0	(27)	(28)	GPIO1
GPIO5	(29)	(30)	GND
GPIO6	(31)	(32)	GPIO12
GPIO13	(33)	(34)	GND
GPIO19	(35)	(36)	GPIO16
GPIO26	(37)	(38)	GPIO20
GND	(39)	(40)	GPIO21

Rysunek 4. 4 Schemat będący wynikiem zastosowania polecenia *pinout*. Numeracja w konwencji BOARD w nawiasach, na zewnątrz BCM.

```
GPIO.setmode(GPIO.BCM)
```

Rysunek 4. 5 Deklaracja trybu numeracji BCM w projekcie.

Innymi funkcjami oferowanymi przez bibliotekę jest:

- ustawianie pinów jako wejścia lub wyjścia
- przypisywanie wejściom stanów (wysoki lub niski)
- odczytywanie stanów wyjść
- konfiguracja programowych rezystorów podciągających i ściąających (ang. pull up/down)
- detekcja zbocza
- reagowanie na wykryte zbocze.

```
GPIO.setup(DRZWI_PIN, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(ALARM_PIN, GPIO.OUT)
```

Rysunek 4. 6 Przykładowa konfiguracja wejścia oraz wyjścia. Widoczne przypisanie stanu IN oraz OUT pinowi o numerze przypisanym do zmiennej oraz konfiguracja programowego rezystora zwierającego do masy (pull down).

```
GPIO.output(ALARM_PIN, 1)
```

Rysunek 4. 7 Przypisanie stanu wysokiego do pinu wejściowego.

```
while GPIO.input(BUTTON_PIN) == 1:
```

Rysunek 4. 8 Wykorzystanie odczytu wartości pinu wejściowego w pętli while.

```
GPIO.add_event_detect(MQ2_PIN, GPIO.FALLING)
GPIO.add_event_callback(MQ2_PIN, zagazowanie)
```

Rysunek 4. 9 Przykład wykrycia wystąpienia zbocza na zdefiniowanym wejściu oraz reakcja na to zbocze poprzez wywołanie odpowiedniej funkcji.

4.4 Biblioteka I2C_LCD_driver

Obsługa i numeracja pinów na płytce to oczywiście podstawa, ale dla kompleksowej obsługi projektu programistycznego konieczne jest też zapewnienie dostępu do podłączonych do urządzenia peryferiów. Fizyczne połączenie realizowane jest za pomocą kilku interfejsów, z których pierwszym jest omawiany w poprzednich rozdziałach I²C z pomocą którego podłączony jest wyświetlacz LCD 2x16. Komunikacja z podłączonym w ten sposób urządzeniem umożliwia biblioteka języka Python o nazwie I2C_LCD_driver. Z jej pomocą w łatwy sposób możliwe jest wypisywanie komunikatów od wskazanego, określanego za pomocą dwóch współrzędnych miejsca na wyświetlaczu oraz czyszczenie wyświetlacza, celem zrobienia miejsca na następne komunikaty.

```
import I2C_LCD_driver
```

Rysunek 4. 10 Import biblioteki I2C_LCD_driver do projektu

```
mylcd.lcd_clear()
mylcd.lcd_display_string("ALARM", 1, 0)
mylcd.lcd_display_string("DETEKCJA GAZU", 2, 0)
```

Rysunek 4. 11 Obsługa wyświetlacza za pomocą zaimportowanej powyżej biblioteki. Wyczyszczenie z tekstu i wypisanie komunikatów od początku pierwszej oraz drugiej linii.

4.5 Biblioteka adafruit_dht

Dostęp do czujnika temperatury i wilgotności DHT11 podłączonego interfejsem 1-Wire uzyskano przy użyciu biblioteki adafruit_dht. Jest to niezwykle prosta biblioteka, w której po zdefiniowaniu odpowiedniego pinu GPIO, podłączonego do linii danych czujnika możliwe jest odczytanie temperatury oraz wilgotności.

```
DHT_SENSOR = DHT.DHT11(DHT_PIN)
```

Rysunek 4. 12 Zdefiniowanie pinu podłączonego do linii danych DHT11.

```
temp = DHT_SENSOR.temperature
hum = DHT_SENSOR.humidity
```

Rysunek 4. 13 Odczyt temperatury i wilgotności.

4.6 Biblioteka datetime

Do obsługi daty i czasu w projektach Internetu Przedmiotów istnieje szereg gotowych pakietów dla różnych języków programowania. Fundamentalną jednak rzeczą jest sposób dostarczania informacji o aktualnym czasie, który następnie przetwarzany jest programowo w sposób zadany przez programistę. Możemy tutaj wyróżnić dostarczanie daty i godziny sprzętowo, przez zewnętrzne lub wbudowane moduły zegarów czasu rzeczywistego lub programowo, pobierając te wartości bezpośrednio z systemu operacyjnego, lub z jednego z dostępnych w Internecie serwerów NTP (ang. *Network Time Protocol*). W opisywanym w niniejszej pracy projekcie zdecydowano się zastosować czas systemowy dostępny w systemie Raspbian. Decyzja ta uwarunkowana była zarówno prostotą tego rozwiązania jak i łatwością zmiany tego parametru w przypadku wciąż obecnej w Polsce letniej zmiany czasu lub konieczności pracy systemu w innej strefie czasowej. W przypadku zdecydowanej większości fizycznych zegarów czasu rzeczywistego opisane wyżej sytuacje stawiają osobę odpowiedzialną za system przez koniecznością fizycznego odwiedzenia miejsca, gdzie znajduje się chronometr, podłączenia się do niego i ręcznego przeprogramowania, natomiast w przypadku czasu systemowego jedynym ewentualnym działaniem jest połączenie się z urządzeniem, np. poprzez SSH i zmiana strefy czasowej z poziomu systemu operacyjnego. Nie jest to oczywiście rozwiązanie doskonałe i pozbawione wad i konieczna jest każdorazowa analiza potencjalnych ryzyk podczas doboru rozwiązania dla konkretnej potrzeby. Może się bowiem okazać, że bardziej od elastyczności zmian istotne jest np. zabezpieczenie przed sabotażem poprzez atak z zewnątrz, wtedy bezpieczniejszym rozwiązaniem będzie sprzętowy zegar. Na potrzeby projektu z uzyskanego w opisany wyżej sposób „surowego” czasu konieczne było wyekstrahowanie daty w formacie RRRR-MM-DD oraz godziny w formacie GG:mm. Efekt ten został uzyskany za pomocą jednej z kilku bibliotek czasu obecnych w języku Python o nazwie datetime. Moduł ten w prosty sposób pozwala na pobranie aktualnej daty a następnie wycięcie z niej interesujących nas wartości i zapisanie w żądanym formacie jako łańcuch znaków.

```
now = datetime.datetime.now()
d = now.strftime("%Y-%m-%d")
t = now.strftime("%H:%M")
```

Rysunek 4. 14 Pobranie aktualnej daty i godziny oraz przypisanie ich do zmiennych w ustalonym formacie.

4.7 Biblioteka mfrc522

W realizacji czytników RFID, będących kluczowym elementem systemu kontroli dostępu wykorzystano opisany już wcześniej czytnik z interfejsem SPI, obsługiwany w programie przy zastosowaniu biblioteki mfrc522. W projekcie zastosowano jedną z klas obecnych w tej bibliotece o nazwie SimpleMFRC522. Po wywołaniu konstruktora tej klasy na rzecz utworzonego obiektu można korzystać z metod read() oraz write(), które upraszczają zapis lub odczyt danych z tagu RFID do jednej linii kodu. Zastosowane w projekcie breloki poza przypisanym tylko do odczytu unikatowym numerem mają też dostępne programowalne pole „text”, które umożliwia na każdym tagu zapis jakiejś wartości jak chociażby imię i nazwisko lub innego rodzaju służbowy identyfikator. Daje to twórcom systemu możliwość zapisu dodatkowej informacji mogącej służyć np. ułatwieniu identyfikacji w przypadku zgubienia breloka lub zwiększeniu bezpieczeństwa poprzez chociażby wprowadzenie w tym polu dodatkowego numeru kontrolnego lub innego narzędzia przypominającego weryfikację dwustopniową, co zwiększy odporność systemu na sabotaż.

```
from mfrc522 import SimpleMFRC522
```

Rysunek 4. 15 Import modułu SimpleMFRC522

```
id, text = reader.read()
```

Rysunek 4. 16 Odczyt numeru oraz programowalnej wartości tekstowej z tagu RFID.

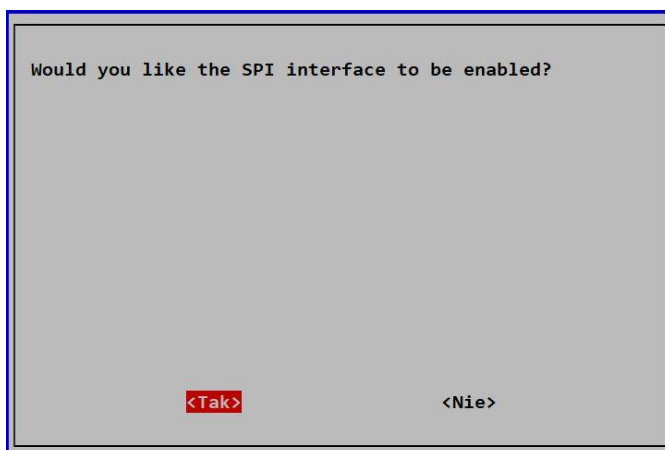
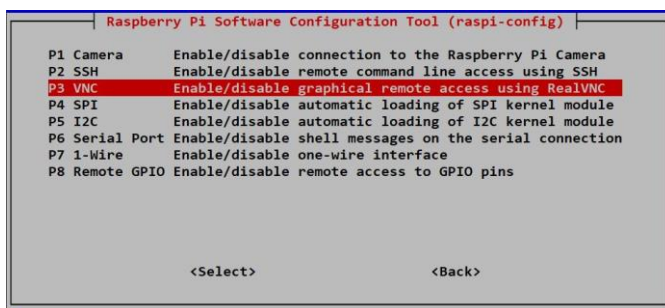
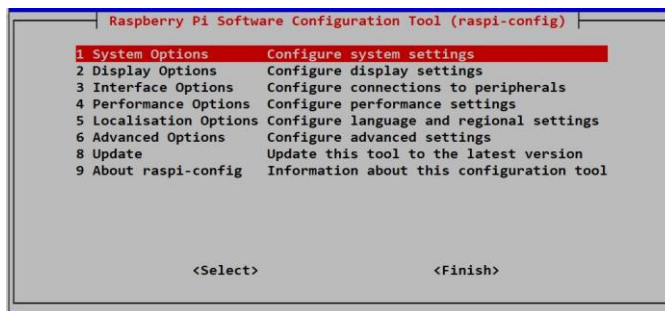
4.8 Implementacja

Proces implementowania zaprojektowanego systemu rozpoczęto od uruchomienia na platformie sprzętowej obsługi używanych w projekcie interfejsów 1-Wire, SPI oraz I²C. Domyślnie w systemie Raspbian interfejsy te są wyłączone, jednak można je w prosty sposób uruchomić w programie raspi-config, wywołanym z przełącznikiem sudo.

```
sudo raspi-config
```

Rysunek 4. 17 Wywołanie programu raspi-config

Następnie wybieramy menu zarządzania interfejsami i kolejno uruchamiamy każdy niezbędny do pracy. Po ich uruchomieniu konieczne jest zrestartowanie urządzenia.



Rysunek 4. 18 Uruchomienie przykładowego interfejsu.

Po wykonaniu poniższych kroków dla każdej wykorzystywanej magistrali oraz fizycznym podłączeniu urządzeń do pinów Raspberry zdecydowano, że za obsługę będą

odpowiedzialne dwa skrypty. Pierwszy z nich o nazwie sterownik.py odpowiada za systemy:

- alarmowy i antywłamaniowy
- detekcji dymu
- detekcji zalania
- kontroli temperatury i wilgotności
- obsługi wyświetlacza oraz daty i godziny

Drugi z programów o nazwie rfid.py odpowiada za realizację działania systemu kontroli dostępu do budynku. Podział na dwa skrypty zdeterminowany był koniecznością działania dwóch stale nasłuchujących pętli, jednej wykrywającej zbocza sensorów i drugiej odczytującej dane z breloków RFID. Możliwe było wykonanie tego w jednym skrypcie np. dzieląc go na wątki, jednak ze względu na konieczność pracy w czasie zbliżonym do rzeczywistego oraz chęć odseparowania od siebie systemów alarmowania od kontroli wejść zdecydowano się na powyższe rozwiązanie. W obu skryptach po zaimportowaniu opisanych wcześniej bibliotek koniecznych do obsługi sprzętu należało zadeklarować numery fizycznych pinów złącza do których podłączono konkretne podzespoły, a następnie ustalić ich tryby pracy oraz stany początkowe.

```
#PINY GPIO

DHT_PIN = 26
PIR_PIN = 16
HL83_PIN = 6
MQ2_PIN = 5
KONTAKTRON_PIN = 24
DRZWI_PIN = 17
ALARM_PIN = 27
LAMPY_PIN = 22
BUTTON_PIN = 14

GPIO.setmode(GPIO.BCM)
GPIO.setup(PIR_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(HL83_PIN, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(MQ2_PIN, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(DRZWI_PIN, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(ALARM_PIN, GPIO.OUT)
GPIO.output(ALARM_PIN, 1)
GPIO.setup(KONTAKTRON_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down = GPIO.PUD_UP)
```

Rysunek 4. 19 Deklaracja numerów pinów i ustalanie ich trybów oraz wartości początkowych.

W skrypcie odpowiedzialnym za kontrolę dostępu dodatkowo zadeklarowano wcześniej odczytane numery tagów RFID. Osoby uprawnione do wejścia wpisano jako kolejne elementy słownika „uprawnieni”, zadeklarowano też pusty słownik o nazwie „obecni”, do którego dodawani będą użytkownicy systemu po wejściu do obiektu i usuwani z niego po wyjściu. Dla uproszczenia założono, że użytkownik wchodząc i wychodząc odbija się na tym samym czytniku. W praktycznej realizacji prawdopodobnie konieczne byłoby umieszczenie dwóch czytników po dwóch stronach głównych drzwi, a w sytuacji, gdy zliczanie i rejestrowanie osób miało być istotne np. na wypadek ewakuacji budynku zaleca się także zastosowanie kołowrotu, aby uniknąć przejścia kilku osób przy jednym otwarciu przejścia. W opisanym tutaj przypadku w sytuacji konieczności przeprowadzenia akcji ratunkowej można skorzystać z zawartości słownika „obecni”, chociażby drukując ją i przekazując służbom ratunkowym. W wersji produkcyjnej tego typu systemu konieczne byłoby przechowywanie danych takich jak lista kart i breloków z przypisanymi osobami, uprawnienia do wejść (zakładając więcej niż jedno), czy listę osób aktualnie obecnych w budynku poza kodem źródłowym programu, najlepiej w formie bazy danych oraz utworzenie osobnego podprogramu umożliwiającego dodawanie nowych użytkowników. Na potrzeby prototypu zastosowane rozwiązanie, czyli umieszczenie numerów identyfikatorów bezpośrednio w kodzie w postaci stałych, a listy uprawnionych i obecnych w słownikach jest wystarczające do przeprowadzenia eksperymentu.

```
DRZWI_PIN = 17
LAMPY_PIN = 22
GPIO.setmode(GPIO.BCM)
GPIO.setup(DRZWI_PIN, GPIO.OUT)
GPIO.output(DRZWI_PIN, 1)
GPIO.setup(LAMPY_PIN, GPIO.OUT)
GPIO.output(LAMPY_PIN, 1)
#Numery tagow RFID
BRELOK_1 = 85733152491
KARTA_1 = 219546064077
KARTA_2 = 704240555340
uprawnieni = {85733152491, 219546064077, 704240555340}
obecni = {}
```

Rysunek 4. 20 Deklaracja numerów pinów GPIO wraz z ich konfiguracją oraz zadeklarowanie danych potrzebnych do obsługi kontroli dostępu w skrypcie rfid.py.

Kolejnym wspólnym dla obu skryptów krokiem jest zdefiniowanie wykorzystywanych później w głównej pętli programu funkcji. W krótszym ze skryptów realizującym kontrolę dostępu są to dwie funkcje odpowiedzialne odpowiednio za zapis oraz odczyt wartości z tagów RFID. Pierwsza z nich, właściwie nieużywana w głównej pętli, została zastosowana na etapie testowania do dopisywania do pola „text” posiadanych przez autora pracy

breloków i kart dodatkowych informacji. W ostatecznej wersji programu zdecydowano się nie stosować tego pola. Sama funkcja zawiera wywołanie konstruktora obiektu klasy SimpleMFRC522 oraz zastosowanie metody obiektu tej klasy nazwie „write()” celem dopisania ciągu znaków przekazanego jako argument funkcji. Po wykonaniu przypisania na standardowym wyjściu wyświetlany jest komunikat o sukcesie przypisania. Obie te komendy znajdują się w bloku obsługi błędów, aby niezależnie od sukcesu wykonania operacji zapisu na koniec dla bezpieczeństwa zastosować komendę przywracającą piny do ich domyślnej konfiguracji. Konieczność zastosowania tego polecenia zostało zauważone w toku testów, ponieważ późniejsze operacje odczytu bez zastosowania w tym miejscu GPIO.cleanup() nie zawsze dawały prawidłowy wynik.

```
def rfid_zapis(nazwa):  
    reader = SimpleMFRC522()  
    try:  
        reader.write(nazwa)  
        print("Przypisano")  
    finally:  
        GPIO.cleanup()
```

Rysunek 4. 21 Funkcja zapisu danych na breloku zbliżeniowym.

Druga z funkcji odpowiedzialna za odczyt danych co do zasady działa identycznie. Jediną różnicą jest kierunek działania, tj. przypisanie pobranej wartości do zmiennych i zwrócenie tej wartości jako wynik dziania podprogramu. Tak jak uprzednio napisano, zdecydowano się korzystać jedynie z pola z numerem id, zatem wyłącznie on jest zwracany.

```
def rfid_odczyt():  
    reader = SimpleMFRC522()  
    try:  
        id, text = reader.read()  
    finally:  
        GPIO.cleanup()  
    return id
```

Rysunek 4. 22 Funkcja odczytu danych z breloku zbliżeniowego.

Skrypt sterownik.py zawiera funkcje, obsługujące zdarzenia:

- Detekcji zalania
- Detekcji zagazowania
- Naruszenia kontaktronu przy drzwiach
- Wykrycia ruchu przez czujnik podczerwieni

Pierwsza z czterech funkcji o nazwie „zalanie()” przyjmuje jeden argument będący numerem pinu czujnika cieczy. W przypadku jej wywołania wyświetla ona komunikat o

wykryciu zalania na ekranie LCD oraz przesyła sygnał modulowany co sekundę na przekaźnik obsługujący sygnalizator alarmowy, co skutkuje uruchomieniem sygnalizacji świetlno-dźwiękowej. Dwoma niezbędnymi warunkami przerwania pętli, co równoznaczne jest z zakończeniem alarmu jest zanik źródła alarmu, czyli zaprzestanie wysyłania sygnału przez czujnik oraz ręczne skasowanie alarmu przez naciśnięcie przycisku przez operatora systemu. Uniemożliwia to celowe skasowanie alarmu bez wykrycia i usunięcia jego przyczyny (warunek ustąpienia źródła), a jednocześnie nawet podczas zaistnienia problemu przez krótki czas zapewnia, że zostanie to przez osobę odpowiedzialną za nadzór zauważone (warunek wciśnięcia przycisku).

```
def zalanie(pin):
    while GPIO.input(pin) == 0:
        while GPIO.input(BUTTON_PIN) == 1:
            mylcd.lcd_clear()
            mylcd.lcd_display_string("ALARM ZALANIE", 1, 0)
            GPIO.output(ALARM_PIN, 0)
            time.sleep(1)
            GPIO.output(ALARM_PIN, 1)
            time.sleep(1)
```

Rysunek 4. 23 Funkcja wywoływana w przypadku wykrycia zalania budynku.

Kolejna funkcja o nazwie „zagazowanie()” poza innym warunkiem jej wywołania oraz komunikatem nie różni się zupełnie niczym od poprzedniej.

```
def zagazowanie(pin):
    while GPIO.input(pin) == 0:
        while GPIO.input(BUTTON_PIN) == 1:
            mylcd.lcd_clear()
            mylcd.lcd_display_string("ALARM", 1, 0)
            mylcd.lcd_display_string("DETEKCJA GAZU", 2, 0)
            GPIO.output(ALARM_PIN, 0)
            time.sleep(1)
            GPIO.output(ALARM_PIN, 1)
            time.sleep(1)
```

Rysunek 4. 24 Funkcja wywoływana w przypadku detekcji gazu w budynku.

Następny podprogram odpowiedzialny za wywołanie alarmu w przypadku sforsowania drzwi przypomina dwa poprzednie sposobem działania, ale ma nieco bardziej skomplikowany warunek wyjścia z pętli. Poza skasowaniem alarmu przyciskiem oraz zanikiem przerwy w obwodzie kontaktronu konieczna jest także informacja o stanie przekaźnika odblokowującego drzwi (zmienna DRZWI_PIN). Bez tej informacji alarm mógłby być wyzwalany za każdym razem przy naruszeniu czujnika, również wtedy, gdy drzwi są otwierane przez uprawnioną kartę. Czyniłoby to ten alarm bezużytecznym. Posiadając tę dodatkową informację możliwe jest wyzwolenie działania tylko w

przypadku, gdy przekaźnik nie jestysterowany, a zatem wejście następuje przy użyciu siły.

```
def drzwi(pin):  
    if GPIO.input(pin) == 1:  
        if GPIO.input(DRZWI_PIN) == 1:  
            while GPIO.input(BUTTON_PIN) == 1:  
                mylcd.lcd_clear()  
                mylcd.lcd_display_string("ALARM", 1, 0)  
                mylcd.lcd_display_string("DRZWI SFORSOWANE", 2, 0)  
                GPIO.output(ALARM_PIN, 0)  
                time.sleep(1)  
                GPIO.output(ALARM_PIN, 1)  
                time.sleep(1)
```

Rysunek 4. 25 Funkcja wywoływana w przypadku sforsowania drzwi.

Ostatnia funkcja również reaguje tak samo jak poprzednie, ale ma nieco inny warunek wyjścia z pętli, polegający na wystąpieniu zdarzenia w godzinach 21.00-6.00. spowodowane jest to założeniem, że alarm uzbrojony jest właśnie w tych godzinach.

```
def wlamanie(pin):  
    h = datetime.datetime.now().strftime("%H")  
    if h < 6 or h > 21:  
        while GPIO.input(BUTTON_PIN) == 1:  
            mylcd.lcd_clear()  
            mylcd.lcd_display_string("ALARM", 1, 0)  
            mylcd.lcd_display_string("WLAMANIE", 2, 0)  
            GPIO.output(ALARM_PIN, 0)  
            time.sleep(1)  
            GPIO.output(ALARM_PIN, 1)  
            time.sleep(1)
```

Rysunek 4. 26 Funkcja wywoływana w przypadku wykrycia ruchu w zadanych godzinach.

Jedyną różnicą w strukturze obu skryptów jest obecna w sterownik.py detekcja wystąpienia zbocza, będącego informacją o wystąpieniu zdarzenia oraz reakcja na to zbocze w postaci wywołania konkretnej z 4 wcześniej zdefiniowanych funkcji. W przypadku wystąpienia wspomnianego zbocza instrukcja zostaje uruchomiona w oddzielnym wątku nie naruszając działania głównego programu, co pozwala na bezkolizyjny powrót do niego po ustąpieniu alarmu.

```

GPIO.add_event_detect(HL83_PIN, GPIO.FALLING)
GPIO.add_event_callback(HL83_PIN, zalanie)
GPIO.add_event_detect(MQ2_PIN, GPIO.FALLING)
GPIO.add_event_callback(MQ2_PIN, zagazowanie)
GPIO.add_event_detect(KONTAKTRON_PIN, GPIO.RISING)
GPIO.add_event_callback(KONTAKTRON_PIN, drzwi)
GPIO.add_event_detect(PIR_PIN, GPIO.RISING)
GPIO.add_event_callback(PIR_PIN, wlamanie)

```

Rysunek 4. 27 Wykrywanie zbocza sensorów i zdefiniowanie reakcji na ich wystąpienia.

Ostatnim elementem obu programów są pętle zawierające główny kod skryptu. Instrukcje w nich zawarte są nieco inne i zostaną poniżej pokrótce omówione.

W sterownik.py program w czasie rzeczywistym pobiera temperaturę z DHT11. W czasie testowania aplikacji wykryto występujący okresowo błąd, który zatrzymywał program, a który ze względu na brak wpływu na poprawność odczytów można zignorować, zatem procedurę odczytu parametrów z czujnika umieszczono w bloku obsługi wyjątków, nakazując interpreterowi kontynuować przetwarzanie kodu programu w przypadku wystąpienia wspomnianego błędu. Następnie obie te wartości zostają wyświetlone na ekranie lcd z odpowiednimi etykietami i oznaczeniem jednostek. Po upływie 10 sekund na wyświetlaczu następuje zmiana i pokazywane są pobrane kilka linii wcześniej z systemu data i godzina w odpowiednim formacie. Ta informacja również pokazywana jest przez 10 sekund, po czym wykonanie pętli rozpoczyna się od początku. Sytuacja trwa do chwili ręcznego przerwania działania programu lub wykrycia zbocza któregoś z czujników i przekazanie sterowania do wątku, w którym wywołano procedurę odpowiedzialną za obsługę alarmu.

```

def main_loop():
    while True:
        mylcd lcd_clear()
        try:
            temp = DHT_SENSOR.temperature
            hum = DHT_SENSOR.humidity
        except RuntimeError:
            continue
        mylcd lcd_clear()
        mylcd lcd_display_string("TEMPERATURA %d°C" % (temp, chr(223)),
1, 0)
        mylcd lcd_display_string("WILGOTNOSC %d%" % (hum), 2, 0)
        time.sleep(10)
        mylcd lcd_clear()
        now = datetime.datetime.now()
        d = now.strftime("%Y-%m-%d")
        t = now.strftime("%H:%M")
        mylcd lcd_display_string(d, 1, 0)
        mylcd lcd_display_string(t, 2, 0)
        time.sleep(10)

```

```
mylcd.lcd_clear()

main_loop()
```

Rysunek 4. 28 Główna pętla skryptu sterownik.py.

Skrypt rfid.py w swojej pętli głównej oczekuje na odczyt breloka bądź karty i następnie podejmuje działania w zależności od przynależności do grup. W przypadku nieuprawnionej karty następuje natychmiastowy powrót po początku instrukcji. Jeżeli właściciel karty jest jednak uprawniony do wejścia, to sprawdzany jest kierunek jego podróży na podstawie obecności lub braku wpisu w słowniku obecni. Jeżeli dana osoba nie jest jeszcze wpisana, oznacza to, że wchodzi i należy ją dodać do słownika. W przeciwnym wypadku, gdy numer karty już jest wpisany, odbicie karty oznacza wyjście i konieczne jest usunięcie numeru z listy obecnych w budynku. W kolejnym kroku sprawdzana jest ilość osób obecnych w obiekcie i jeżeli jest ona większa od zera, zapalane jest światło (przełącznik sterowany sanem niskim, stąd logiczne zero), a jeżeli ostatnia osoba wyszła z budynku, światło jest gaszone do momentu wejścia pierwszego pracownika. Po wykonaniu tego sprawdzenia następuje wysterowanie przełącznika i w konsekwencji otwarcie drzwi na 10 sekund umożliwiając wejście do budynku, po czym są one ponownie zamykane.

```
while True:
    if rfid_odczyt() in uprawnieni:
        if rfid_odczyt() in obecni:
            del obecni[rfid_odczyt()]
        else:
            obecni[rfid_odczyt()] = [rfid_odczyt()]
    if len(obecni) > 0:
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(LAMPY_PIN, GPIO.OUT)
        GPIO.output(LAMPY_PIN, 0)
    else:
        GPIO.setmode(GPIO.BCM)
        GPIO.setup(LAMPY_PIN, GPIO.OUT)
        GPIO.output(LAMPY_PIN, 1)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(DRZWI_PIN, GPIO.OUT)
    GPIO.output(DRZWI_PIN, 0)
    time.sleep(10)
    GPIO.output(DRZWI_PIN, 1)
```

Rysunek 4. 29 Główna pętla skryptu rfid.py.

Rozdział 5

Testowanie rozwiązania oraz spostrzeżenia

Po zakończeniu fazy implementacyjnej projekt poddano szczegółowym testom, podczas których wyzwalano poszczególne alarmy pojedynczo i w większej liczbie, w różnej kolejności i kombinacjach. Dokonano również prób sabotażu, aby sprawdzić odporność systemu na przypadkowe oraz celowo wywołane awarie.

5.1 Wyzwalanie alarmów pojedynczo

W przypadku każdego z możliwych alarmów, czyli:

- Sforsowania drzwi
- Zalania
- Wykrycia gazu
- Naruszenia strefy czujnika podczerwieni (włamanie)

Wywołanie jednego z tych zdarzeń naraz daje zamierzony efekt, czyli wyświetlenie na ekranie LCD informacji o rodzaju zagrożenia oraz uruchomienie sygnalizatora dźwiękowo-światlnego. Samo usunięcie źródła problemu, czyli np. osuszenie czujnika cieczy lub ponowne zwarcie kontaktronu nie jest wystarczające do zakończenia alarmu. Zgodnie z założeniami konieczne jest dodatkowe ręczne skasowanie alarmu przez operatora poprzez wciśnięcie przycisku. W sytuacji odwrotnej, tj. wciśnięcia przycisku skasowania alarmu bez usunięcia jego przyczyny ekran LCD oraz sygnalizator wracają do stanu normalnego tylko tak długo jak długo przycisk jest w stanie wciśniętym, zwolnienie go powoduje ponowne uruchomienie alarmu. Działanie to jest efektem wrażliwości na zbocze narastające pinu podłączonego do przycisku i można to traktować jako możliwość chwilowego „wyciszenia” alarmu przez operatora świadomego zagrożenia, celem usunięcia przyczyny. Można również w wersji produkcyjnej systemu dobrać przycisk niemożliwy do przytrzymania lub zablokowania w pozycji wciśniętej np. poprzez zaklejenie taśmą lub przyciśnięcie ciężkim przedmiotem.





Rysunek 5. 1 Komunikaty wyświetlane na ekranie LCD podczas wyzwolenia poszczególnych alarmów.

5.2 Wyzwalanie więcej niż jednego alarmu jednocześnie

Podczas wyzwolenia więcej niż jednego alarmu jednocześnie niezależnie od ich rodzaju oraz kolejności najnowszy z wywołanych jest ogłaszany na ekranie LCD, natomiast sygnał dźwiękowo-światlny przez cały czas jest aktywny. Podczas poprawnego kasowania alarmu (usunięcie przyczyny i wciśnięcie przycisku) w kolejności odwrotnej od wywołanej na ekranie pojawia się kolejny najnowszy z wywołanych alarmów, w przypadku usunięcia przyczyny alarmu innego niż najnowszy i naciśnięcia przycisku system nadal informuje o najnowszym wyzwolonym alarmie. Całkowite wyłączenie ostrzegania następuje dopiero po usunięciu wszystkich problemów i wciśnięciu przycisku kasowania. Jest to zachowanie akceptowalne, nie powoduje ono sytuacji fałszywego braku informacji w sytuacji, gdy powinna ona zostać przekazana.

5.3 Odporność na sabotaż

Na potrzeby eksperymentu opracowano następujące scenariusze potencjalnych awarii:

- Odłączenie zasilania systemu (12V DC) – układ zasilania prądem stałym posiada dodatkowe zasilanie bateryjne umożliwiające pracę podzespołów po zaniku zasilania w sieci. Obecność zasilania sieciowego jest sygnalizowana przez świecącą się diodę LED na obudowie, jednak nie ma żadnego systemu powiadamiania na

odległość np. w postaci styku informującego o utracie jednego źródła zasilania. Zasilacz posiada za to styk TAMPER będący w układzie normalnie zamkniętym, który uaktywnia się po otwarciu obudowy, co stanowi zabezpieczenie przed ingerencją w urządzenie. W projekcie styku nie zastosowano, przy wdrożeniu na skalę przemysłową postulowane wykorzystanie tej informacji i wyzwolenie alarmu np. „SABOTAŻ ZASILANIA”.

- Odłączenie zasilania jednostki sterującej (230V AC) – Obwód prądu zmiennego, którym zasilany jest komputer Raspberry Pi nie jest w żaden sposób chroniony ani monitorowany. Odłączenie zasilania powoduje wyłączenie urządzenia i zakończenie działania skryptów odpowiedzialnych za działanie systemu. Konieczne monitorowanie obecności zasilania oraz jego modyfikacja poprzez wprowadzenie do zasilania 12V falownika, zmianę zasilacza Raspberry na DC lub dołączenie do zestawu zasilacza awaryjnego UPS.
- Odłączenie jednostki sterującej od Internetu – obecnie system przystosowany do działania offline, w przypadku zmiany w sposobie przechowywania danych systemu kontroli dostępu konieczne zagwarantowanie dostępu do sieci LAN w przypadku lokalnego serwera bazodanowego lub Internet w przypadku bazy danych z zdalnej lokalizacji lub chmurze
- Odłączenie sensorów od jednostki sterującej w normalnym trybie pracy – podczas odłączenia czujników w normalnym trybie pracy system nie reaguje w żaden sposób, co stwarza możliwość obejścia zabezpieczeń. Wyjątek stanowi kontaktron, w logice NC, gdzie przerwa w obwodzie stanowi wyzwolenie alarmu. W przypadku pozostałych złącz konieczne odpowiednie zabezpieczenie przewodów, aby uniemożliwić ich przecięcie oraz zastosowanie styków sabotażowych lub złącza z parametryzowanymi rezystorami 2EOL/NO. Niezależnie od rozwiązań systemowych niezbędne jest zapewnienie okresowego przeglądu sprawności działania.
- Odłączenie sensorów od jednostki sterującej podczas trwania alarmu – odłączenie sensora podczas trwania alarmu nie rozłącza go, nadal konieczne jest wciśnięcie przycisku kasowania alarmu. Powoduje to jednak usunięcie źródła alarmu, zatem po skasowaniu go nie będzie już ponownie wywołany. Wyjątek również stanowi działający w obwodzie NO kontaktron, pozostałe sugestie jak w punkcie powyżej.

- Odłączenie ekranu LCD – odłączenie ekranu nie powoduje wyzwolenia alarmu. Podczas trwania alarmu odłączenie nie powoduje jego wyłączenia, sygnalizator świetlno-dźwiękowy nadal działa. Konieczne wprowadzenie złącza sabotażowego oraz okresowy przegląd techniczny.
- Odłączenie sygnalizatora - odłączenie sygnalizatora nie powoduje wyzwolenia alarmu. Podczas trwania alarmu odłączenie nie powoduje jego wyłączenia, ekran LCD z komunikatem o alarmie nadal działa. Konieczne wprowadzenie złącza sabotażowego oraz okresowy przegląd techniczny.
- Uzyskanie nieautoryzowanego zdalnego dostępu do jednostki sterującej – nieautoryzowany dostęp za pomocą protokołów sieciowych SSH lub Telnet może spowodować wyłączenie urządzenia, wyłączenie programu, jego modyfikację lub uzyskanie dostępu do wrażliwych danych. Konieczne zastosowanie zabezpieczeń w postaci silnych haseł i odpowiednio skomplikowanych kluczy szyfrujących, wyłączenia na zaporze dostępu SSH lub Telnet z Internetu i w przypadku konfiguracji zdalnej stosowanie VPN, lub całkowite wyłączenie dostępu zdalnego w urządzeniu jeżeli nie jest ono wymagane.

Rozdział 6

Zakończenie

Bibliografia

The Raspberry Pi Foundation. (2021, 05 09). *www.raspberrypi.org*. Pobrano z lokalizacji
<https://www.raspberrypi.org/blog/8gb-raspberry-pi-4-on-sale-now-at-75/>