

**POLITECHNIKA  
BYDGOSKA**  
Wydział Telekomunikacji,  
Informatyki i Elektrotechniki

**PRACA DYPLOMOWA  
INŻYNIERSKA**  
na kierunku Informatyka Stosowana

**Opracowanie zestawu ćwiczeń laboratoryjnych wyko-  
rzystujących Raspberry Pi**

**Development of a set of laboratory exercises using Raspberry Pi**

Remigiusz Dończyk  
121 179

dr. inż. Beata Marciniak

Bydgoszcz, 13 stycznia 2026

# Metryka pracy dyplomowej

## Dane ogólne

Nazwa Uczelni	Politechnika Bydgoska im. Jana i Jędrzeja Śniadeckich
Wydział	Telekomunikacji, Informatyki i Elektrotechniki
Kierunek	Informatyka Stosowana
Tryb studiów	niestacjonarne
Dane autora	Remigiusz Dończyk, 121 179
Dane promotora	dr. inż. Beata Marciniak

## Dane dotyczące pracy dyplomowej

Język pracy	język polski [PL]
Tytuł pracy	Opracowanie zestawu ćwiczeń laboratoryjnych wykorzystujących <i>Raspberry Pi</i> .
Opis pracy	Celem pracy jest dobór zestawów laboratoryjnych i opracowanie instrukcji do maksymalnie 6 ćwiczeń z użyciem dostarczonych przez prowadzącego zestawów czujników oraz <i>Raspberry Pi</i> . Każde gotowe ćwiczenie powinno składać się z przykładowego programu w języku <i>Python</i> , pozwalającego na konfigurację urządzeń, oraz instrukcji przebiegu ćwiczenia.
	<ol style="list-style-type: none"><li>1. Dobór zestawów czujników, różnych dla każdego ćwiczenia;</li><li>2. przygotowanie programów w języku <i>Python</i> wykorzystujących zestawy czujników;</li><li>3. opracowanie instrukcji ćwiczeń do każdego zestawu, zawierających przykładowe zadanie do rozwiązania;</li><li>4. wykonanie rozwiązań zadań zdefiniowanych w instrukcjach ćwiczeń;</li><li>5. przetestowanie działania rozwiązań i bezpieczeństwa użytkowania.</li></ol>
Typ pracy	inżynierska
Streszczenie	Praca dyplomowa jest dokumentacją rozwiązania problemu inżynierskiego na temat wykorzystania mikrokomputera <i>Raspberry Pi</i> wraz z zestawem modułów fizycznych do opracowania serii zadań laboratoryjnych dla przyszłych roczników studentów na wczesne semestry edukacji na kierunkach związanych z telekomunikacją, informatyką, i elektrotechniką. Część przedująca prezentuje informacje wspólne dla laboratoriów: dostępne prowadzącemu moduły i wybór spośród nich zestawów laboratoryjnych; istniejące alternatywy <i>Raspberry Pi</i> ; połączenia fizyczne i logiczne wykorzystywane przez użyte moduły; oprogramowanie składające się na wygodny w użytkowaniu dla prowadzącego i studentów system; i rozwiązania mające wpływ na bezpieczeństwo urządzeń. Dalsza część pracy jest podzielona na sekcje poświęcone każdemu z laboratoriów z osobna i prezentuje zestaw laboratoryjny wraz z rozwiązaniem zadania laboratoryjnego w celu weryfikacji poprawności przygotowania.
Słowa kluczowe	komputer, <i>Python</i> , <i>Raspberry Pi</i> , czujniki, laboratoria

# Diploma thesis record

## General information

University name	Bydgoszcz University of Science and Technology
Faculty	Telecommunications, Computer Science and Electrical Engineering
Field of study	Applied Computer Science
Mode of study	part-time
Author's information	Remigiusz Dończyk, 121 179
Supervisor's information	dr. inż. Beata Marciniak

## Data regarding the diploma thesis

Language of thesis	Polish [PL]
Title of thesis	Development of a set of laboratory exercises using <i>Raspberry Pi</i> .
Description	The goal of the thesis is the selection of laboratory exercise kits and the development of up to 6 experiments using hardware sensors and <i>Raspberry Pi</i> boards provided by the instructor. Each laboratory should consist of an example <i>Python</i> program, allowing for the configuration of hardware, as well as an exercise guide document.
	<ol style="list-style-type: none"><li>1. Selection of sensors, unique to each exercise;</li><li>2. preparation of <i>Python</i> programs utilizing the selected hardware;</li><li>3. development of instructions for each exercise, containing tasks to be completed;</li><li>4. execution of solutions for tasks contained in the instructions;</li><li>5. test of the solutions' functionality and safety of usage.</li></ol>
Type of thesis	Engineer's
Abstract	This thesis documents the solution to an engineering problem involving the use of a <i>Raspberry Pi</i> microcomputer and a set of physical modules to develop a series of laboratory exercises for future generations of students in early semesters of telecommunications, computer science, and electrical engineering courses. The main section presents information common to the laboratories: modules available to the instructor and the selection of laboratory sets from them; existing <i>Raspberry Pi</i> alternatives; physical and logical connections used by the modules selected; software making up a user-friendly system for the instructor and students; and solutions affecting device security. The remainder of the thesis is divided into sections dedicated to each laboratory and presents the laboratory set and the solution to the laboratory exercise to verify its correctness.
Keywords	computer, <i>Python</i> , <i>Raspberry Pi</i> , sensors, laboratories

---

# Spis treści

---

<b>Spis treści</b>	<b>4</b>
<b>1 Informacje wstępne</b>	<b>6</b>
1.1 Przegląd rynku . . . . .	6
1.2 Dostępny sprzęt . . . . .	7
1.2.1 Raspberry Pi . . . . .	7
1.2.2 Moduły . . . . .	7
1.3 Uzupełnienie: Alternatywy Raspberry Pi . . . . .	8
1.4 Połączenia fizyczne . . . . .	9
1.4.1 GPIO . . . . .	9
1.4.2 Grove . . . . .	9
1.5 Połączenia logiczne . . . . .	11
1.5.1 PWM . . . . .	11
1.5.2 I <sup>2</sup> C . . . . .	12
1.5.3 I <sup>2</sup> S . . . . .	13
1.5.4 SPI . . . . .	13
1.5.5 UART . . . . .	15
1.6 Oprogramowanie . . . . .	16
1.6.1 Raspberry Pi OS . . . . .	16
1.6.2 Bash . . . . .	16
1.6.3 Python . . . . .	17
1.6.4 rpi-lgpio . . . . .	17
1.6.5 marimo . . . . .	18
1.7 Wdrożenie rozwiązania . . . . .	19
1.7.1 Udostępnianie . . . . .	19
1.7.2 Pliki pomocnicze . . . . .	19

1.7.3	Testy rozwiązań	20
1.7.4	Bezpieczeństwo	20
<b>2</b>	<b>Laboratoria</b>	<b>22</b>
2.1	Laboratorium I	22
2.1.1	Wybrany sprzęt	22
2.1.2	Dodatkowe oprogramowanie	23
2.1.3	Rozwiązywanie	23
2.2	Laboratorium II	25
2.2.1	Wybrany sprzęt	25
2.2.2	Dodatkowe oprogramowanie	26
2.2.3	Rozwiązywanie	26
2.3	Laboratorium III	28
2.3.1	Wybrany sprzęt	28
2.3.2	Rozwiązywanie	28
2.4	Laboratorium IV	30
2.4.1	Wybrany sprzęt	30
2.4.2	Dodatkowe oprogramowanie	30
2.4.3	Rozwiązywanie	31
2.5	Laboratorium V	33
2.5.1	Wybrany sprzęt	33
2.5.2	Rozwiązywanie	33
<b>3</b>	<b>Podsumowanie</b>	<b>36</b>
<b>Bibliografia</b>		<b>38</b>
<b>Spis rysunków</b>		<b>40</b>
<b>Spis tabel</b>		<b>41</b>
<b>Spis załączników</b>		<b>42</b>

# Rozdział 1

---

## Informacje wstępne

---

Rozdział ten skupia się na informacjach mających znaczenie dla więcej niż jednego z laboratoriów. W związku z tym, w celu niepowtarzania tych informacji w każdym istotnym miejscu, umieszczone są one w przodujączej sekcji pracy.

### 1.1 Przegląd rynku

Na rynku istnieją zestawy mogące stanowić poszczególne laboratoria, które mogłyby być wykorzystane do przygotowania laboratoriów na potrzeby uczelni. Przystosowanie jednak niezależnych produktów wiąże się z komplikacjami wynikającymi z różnic w wykorzystywanych rozwiązań technicznych oraz programowych.

Ta praca jest kolekcją kilku zestawów wykorzystujących sprzęt dostępny już uczelni, a także zrealizowanych w jednolity i przystępny sposób. Wygodna metoda instalacji, przygotowania, a także wyczyszczenia środowiska laboratoryjnego oraz wykonania zadań zgodna dla wszystkich realizowanych zagadnień niweluje te komplikacje i pozwala na poświęcenie większej ilości czasu treści zadań niżli zapoznania z kolejnymi metodami interakcji ze sprzętem. Wykorzystanie opisanych poniżej rozwiązań pozwala również m.in. na automatyczne sprawdzanie poprawności rozwiązań.

## 1.2 Dostępny sprzęt

### 1.2.1 Raspberry Pi

*Raspberry Pi* [1] to seria małych komputerów jednoprzepłytkowych stworzonych przez Fundację Raspberry Pi w Wielkiej Brytanii. Celem Fundacji jest rozpowszechnienie systemów o możliwościach obliczeniowych, co wpływa na niewielki koszt płytki, i prowadzi do wykorzystania standardowej architektury procesora i modułów komunikacyjnych (*WiFi*, *Bluetooth*). Takie urządzenia można zastosować w wielu różnych projektach m.in. do połączenia urządzeń Internetu Rzeczy.

Koszt najtańszego modelu wynosi ok. 20 zł, a najtańszy pełnoprawny komputer w serii kosztuje ok. 50 zł.

Urządzenia *Raspberry Pi* wyposażone są w zestaw czterdziestu pinów *GPIO* pozwalających na podłączenie dowolnych układów wykorzystujących zasilanie 3,3 V lub 5 V. Stanowią dzięki temu przystępna opcję zarówno do nauki, użytkowania systemów komputerowych, jak i eksperymentów polegających na budowaniu własnych urządzeń.

### 1.2.2 Moduły

Moduły udostępnione przez promotora na potrzeby niniejszej pracy dyplomowej:

- Sense HAT [2];
- Flick HAT [3];
- Grove Base Kit [4];
- Automation HAT Mini [5];
- Enviro HAT [6];
- Unicorn HAT [7];
- Traffic pHAT [8].

Niektóre moduły nie zostały uwzględnione w procesie selekcji – tab. 1.1.

Tabela 1.1: Spis niewykorzystanych modułów

Moduł	Powód
Flick HAT	Firma producenta została rozwiązana [9], oficjalna dokumentacja ich produktów nie jest dostępna, a konfiguracja nakładki nie jest oczywista.
Automation HAT Mini	Zaprojektowany do pracy z urządzeniami o napięciu 12 V lub 24 V, co przy nieprawidłowym użytkowaniu może uszkodzić płytę.
Unicorn HAT	Wyposażony w macierz diod $8 \times 8$ , podobnie do <i>Sense HAT</i> , który ma szersze możliwości.

Pozostałe moduły zostały rozdzielone do wykorzystania w poszczególnych laboratoriach. W związku z ilością możliwych zastosowań zestawu *Grove* został on wykorzystany (używając różnych urządzeń zewnętrznych) dwukrotnie. Szczegółowe opisy połączeń i wykorzystania znajdują się w rozdziałach poświęconych każdemu z laboratoriów z osobna.

### 1.3 Uzupełnienie: Alternatywy Raspberry Pi

Wspominając o tym urządzeniu nie można zapomnieć o podobnych urządzeniach dostępnych na rynku. W ostatnich latach na rynku pojawiły się bowiem alternatywy o porównywalnej lub nawet korzystniejszej cenie. Niektórymi z popularnych rozwiązań w tej kategorii są *Banana Pi* [10], *Odroid* [11], *Libre Computer* [12], *Orange Pi* [13], lub poleasingowe komputery w małej obudowie.

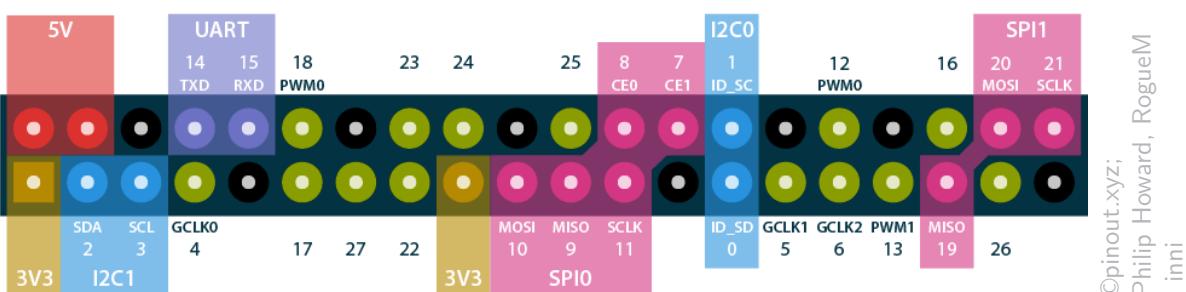
*Raspberry Pi* pełniło ważną funkcję w popularyzowaniu pojęcia komputerów jednopłytkowych jako niedrogiego narzędzia do nauki i prototypowania, i nadal jest najlepiej udokumentowanym z konkurentów, co jest ważnym elementem przystępności w środowisku naukowym. Alternatywy często skupiają się w pierwszej kolejności na udostępnieniu wyższej funkcjonalności, co sprawia, że są dobrą opcją dla bardziej zaawansowanych użytkowników.

Osobnym rodzajem popularnych urządzeń jest seria *Arduino*, która jest często przytaczana wraz z *Raspberry Pi*. Istnieją między nimi jednak istotne różnice, z powodu których urządzenia te nie odgrywają tej samej roli. Urządzenia *Arduino* nie są pełnoprawnymi komputerami (wymagają osobnego urządzenia do zaprogramowania) i konkurują jedynie z *Raspberry Pi Pico*, a nie z resztą płyt o możliwości obsługi pełnego systemu operacyjnego.

## 1.4 Połączenia fizyczne

### 1.4.1 GPIO

Złącze *General-Purpose Input/Output* jest stosowane w mikrokontrolerach, komputerach jednopłytkowych, i innych urządzeniach elektronicznych w celu umożliwienia bezpośredniej komunikacji z układami zewnętrznymi. W przypadku *Raspberry Pi* składa się z czterdziestu pinów [14]. Część możliwości wymaga konfiguracji jądra systemu, co jest przez producenta udostępnione w przystępny sposób. Rysunek 1.1 zaznacza te dodatkowe możliwości; piny wchodzące w ich skład można również użyć w standardowy sposób.



Rysunek 1.1: Układ pinów *Raspberry Pi* – adresacja *BCM*

Drugą dostępną metodą adresacji jest adresacja fizyczna, rozpoczynającą się od 1 i 2 odnoszących się odpowiednio do pinów 3,3 V i 5 V po lewej stronie powyższego rysunku, numerując kolejne kolumny pinów.

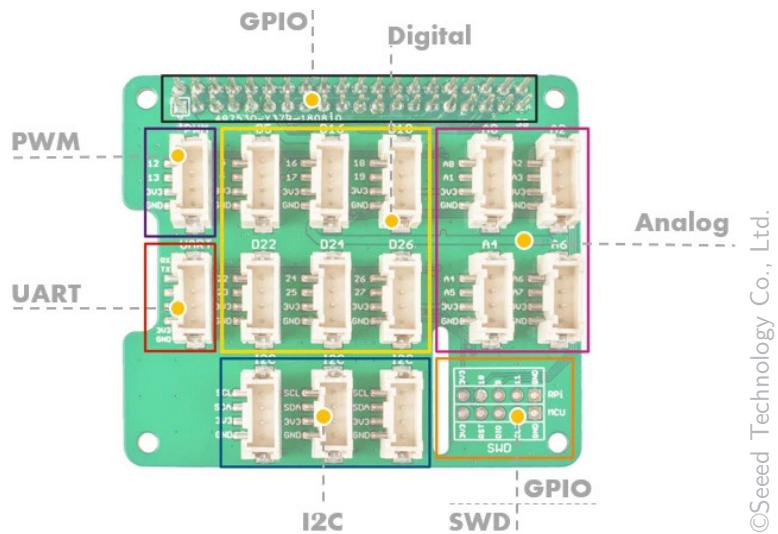
Piny oznaczone mianem *GCLK* udostępniają dostęp do wbudowanego zegara sprzętowego. *PWM*, *I<sup>2</sup>C*, *UART*, i *SPI* zostaną omówione w sekcji 1.5 opisującej połączenia logiczne.

Istnieje wiele bibliotek oferujących środowiska upraszczające interakcję z urządzeniami wykorzystującymi interfejs *GPIO*, w zależności od języka programowania. Praca ta używa języka programowania *Python*, więc przykłady wykorzystują wiązania *liblgpio* dla tego języka (opisane w sekcji 1.6.4).

### 1.4.2 Grove

Moduł *Grove Base HAT* [15] jest podłączony do płytki *Raspberry Pi* przy pomocy interfejsu *GPIO* i pełni funkcję adaptera tego interfejsu na własny system złączy. Moduł wyprowadza interfejs *GPIO*, jednak wiele połączeń pokrywa się z systemem *Grove*, w związku z czym wykorzystanie ich nie jest bezpieczne bez wcześniejszego zrozumienia tego systemu. Na module *Grove Base HAT* złącza podpisane są przy użyciu numeracji *BCM*.

Połączenia fizyczne wykorzystywane przez interfejs *Grove* przedstawione na rysunku 1.2 i opisane w tabeli 1.2 są podobne do standardowych połączeń  $I^2C$ . Połączenia *VCC* i *GND* zawsze dostarczają elementom zasilanie, jednak pozostałe dwa zachowują się w różny sposób w zależności od rodzaju złącza. Połączenia te wykorzystują interfejs *GPIO* do komunikacji z płytą, więc wykorzystanie danego portu *Grove* wraz z pinami *GPIO* o tej samej numeracji wiąże się z konfliktami.



Rysunek 1.2: Diagram złączy *Grove Base HAT*

Tabela 1.2: Opis złączy *Grove Base HAT*

Złącze	Opis
GPIO	Bezpośrednie połączenie do złącza <i>GPIO</i> płytki <i>Raspberry Pi</i> . Może powodować konflikty ze złączami <i>Grove</i> .
I <sup>2</sup> C	Zgodnie ze standardowymi łączami, zewnętrzny pin pełni funkcję zegara, a wewnętrzny przesyłu danych.
Cyfrowy	Zewnętrzny pin pełni funkcję pierwszego cyfrowego złącza wejścia/wyjścia, a wewnętrzny jest złączem dodatkowym dla modułów wykorzystujących dwie linie.
UART	Specjalizowana forma trybu cyfrowego, pierwszy pin odbiera dane od podłączonego urządzenia, a drugi wysyła.
Analogowy	Wejścia analogowe są podłączone do wbudowanego przetwornika analogowo-cyfrowego, co pozwala na ich użycie z płytą <i>Raspberry Pi</i> , nie posiadającej natywnie takiej możliwości. Podobnie do trybu cyfrowego, linia zewnętrzna jest wykorzystywana w pierwszej kolejności, a wewnętrzna w drugiej.
PWM	Port <i>PWM</i> podłączony jest do sprzętowego złącza <i>PWM Raspberry Pi</i> . Pozostałe złącza mogą być używane jako <i>PWM</i> na poziomie oprogramowania. Złącze sprzętowe pozwala na lepszą dokładność przy wysokich częstotliwościach, jednak wchodzi w konflikt z niektórymi innymi funkcjami płytki.
SWD	Dostęp do wbudowanego oprogramowania modułu <i>Grove Base HAT</i> .

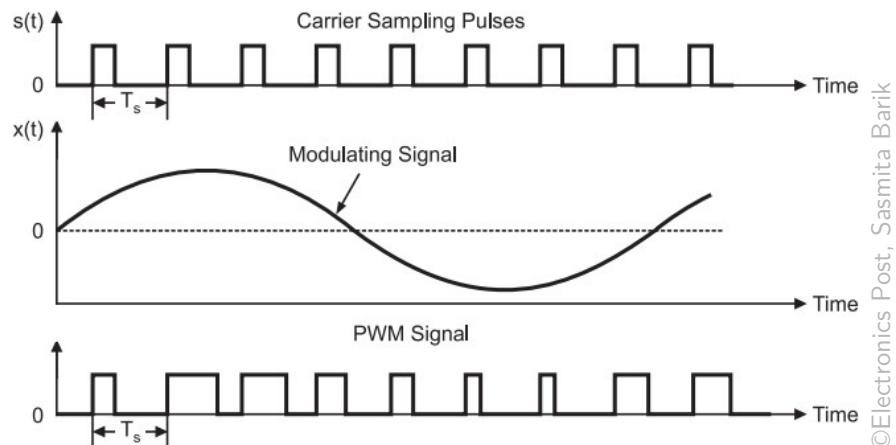
## 1.5 Połączenia logiczne

### 1.5.1 PWM

*Pulse Width Modulation* [16] to technika polegająca na kodowaniu wartości sygnału danych na sygnale nośnym prostokątnym o stałej częstotliwości i amplitudzie jako czasu trwania poszczególnych impulsów elektrycznych, przedstawiona na rysunku 1.3. Wartość przesyłaną ustalającą się regulując proporcję czasu, w której sygnał przyjmuje wartość wysoką. Zbocze rosnące sygnału jest wysyłane w stałych odstępach czasowych, co pozwala na wykorzystanie sygnału bez transmisji osobnego sygnału synchronizacyjnego. Dzięki zapisowi informacji poza amplitudą sygnału *PWM* jest odporne na typowe szумy transmisyjne.

Wadą *PWM* jest zmienność zawartości energetycznej pulsów, w związku z czym linia transmisyjna i urządzenie korzystające z tej modulacji muszą być odporne na maksymalną możliwą ilość energii (wypełnienie sygnału blisko 100 %), aby zapewnić odporność na uszkodzenia. Cechą ta znalazła jednak zastosowanie jako prosta metoda kontroli ilości energii dostarczanej do urządzeń o zmiennych zapotrzebowaniach, np. oświetlenia o regulowanej jasności, bez zmiany

rzeczywistej wartości napięcia.



©Electronics Post, Sasmita Barik

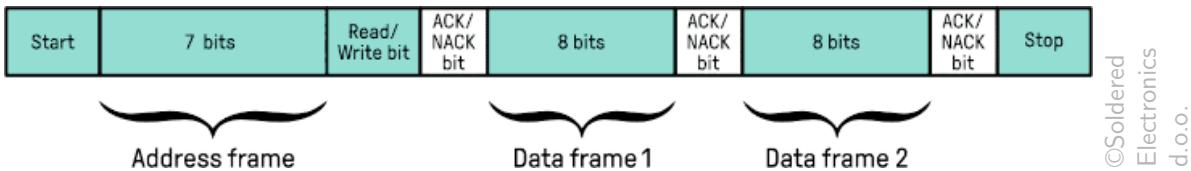
Rysunek 1.3: Modulacja szerokości impulsów

### 1.5.2 I<sup>2</sup>C

*Inter-Integrated Circuit* [17] to synchroniczna szeregowa magistrala komunikacyjna pozwalająca na komunikację wielu urządzeń za pomocą dwóch linii transmisyjnych. Nazwa wywołuje się z wczesnego użycia jako metoda komunikacyjna między układami scalonymi na jednej płytce drukowanej.

Oryginalna idea I<sup>2</sup>C zakłada istnienie tylko jednego urządzenia nadającego, jednak dzięki podłączeniu urządzeń za pomocą otwartego drenu lub kolektora oraz istnieniu mechanizmu wykrywającego kolizje możliwa jest praca w systemie zawierającym wiele urządzeń nadawczych. Standardowa 7-bitowa adresacja pozwala na podłączenie 127 urządzeń odbierających, jednak z powodu maksymalnej dopuszczalnej pojemności o wartości 400 pF w praktyce długość linii jest ograniczona do kilku metrów, co w większości przypadków ogranicza ilość podłączonych urządzeń. Dla sytuacji niestandardowych jest również dostępna adresacja 10-bitowa, zwiększająca limit do 1023 odbiorców.

Obie linie są w stanie spoczynku utrzymywane w stanie wysokim. Rozpoczęcie transmisji jest sygnalizowane poprzez zmianę stanu przez nadajnik na stan niski na obu liniach, a następnie uruchomienie zegara i przesłanie wiadomości o formacie przedstawionym na rysunku 1.4. Wiadomość może zawierać jedną lub więcej sekcji danych w zależności od rodzaju komunikacji. Transmisja jest zakończona poprzez powrót w stan spoczynku.



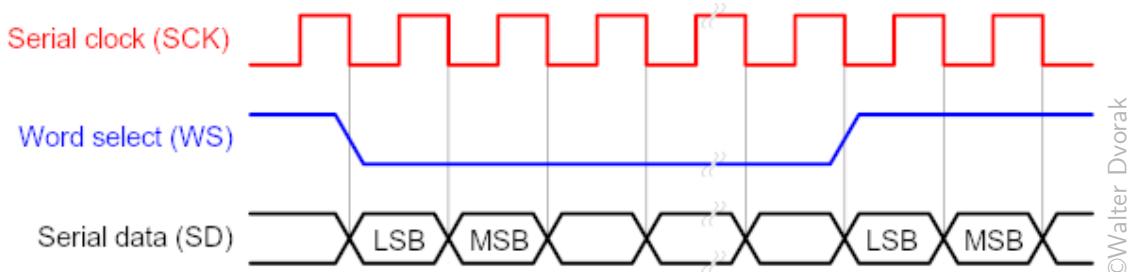
Rysunek 1.4: Przykład wiadomości na linii *SDA* interfejsu *I<sup>2</sup>C*

©Soldered  
Electronics  
d.o.o.

### 1.5.3 I<sup>2</sup>S

*Inter-Integrated Circuit Sound* [18] nie jest powiązane z *I<sup>2</sup>C*, jednak ta sama firma utworzyła ich specyfikacje. Protokół ten jest skupiony na wbudowanych urządzeniach audio, zapewniając prostą i ograniczoną funkcjonalność w celu zapewnienia niezawodności i prostoty wykorzystania.

Magistrala wykorzystuje trzy linie komunikacyjne, w tym dwa zegary. Pierwszy z zegarów ustanawia rytm przesyłania bitów informacyjnych. Drugi zegar jest sygnałem wybierającym jeden z dwóch dostępnych kanałów informacyjnych na linii danych. W teorii sygnał ten może być niepodłączony do zegara, jednak oba kanały przesyłają tą samą ilość danych, w związku z czym użycie odpowiednio dostosowanego zegara znaczaco upraszcza implementację i nie stosuje się w praktyce innych rozwiązań. Trzecia z linii przesyła dane dźwiękowe szeregowo, zakodowane kodem uzupełnień do dwóch, zaczynając od najbardziej znaczącego bitu danych jeden cykl po przełączeniu kanału. Rysunek 1.5 prezentuje te zależności.



Rysunek 1.5: Diagram czasowy *I<sup>2</sup>S*

### 1.5.4 SPI

*Serial Peripheral Interface* [19] to synchroniczny szeregowy interfejs komunikacyjny z jednym urządzeniem nadzorczym i wieloma podległymi wykorzystujący cztery linie. Połączone urządzenia tworzą ze swoich rejestrów wewnętrznych bufor kołowy, co zapewnia prostą implementację komunikacji w trybie pełny duplex.

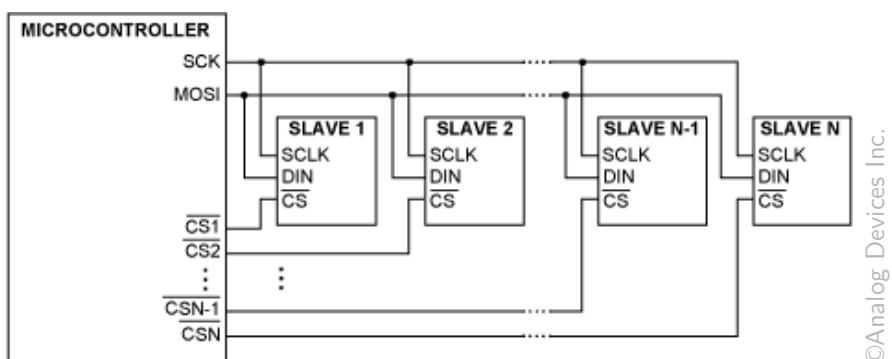
Wykorzystywane linie pełnią funkcje opisane w tabeli 1.3.

Tabela 1.3: Opis linii SPI

Linia	Opis
Chip Select	Sygnał aktywny w stanie niskim przesyłany przez urządzenie nadzorcze, uruchamiający obsługę komend przez podłączone urządzenia.
Serial Clock	Sygnał zegarowy nadawany przez urządzenie nadzorcze.
Master Out Slave In	Linia wysyłająca dane urządzenia nadzorczego.
Master In Slave Out	Linia odbierająca dane urządzenia nadzorczego.

Przesłane komendy są wykonywane w momencie wykrycia zbocza rosnącego sygnału *Chip Select*. Standardowe są dwa sposoby połączeń wykorzystujące linie w odmienny sposób.

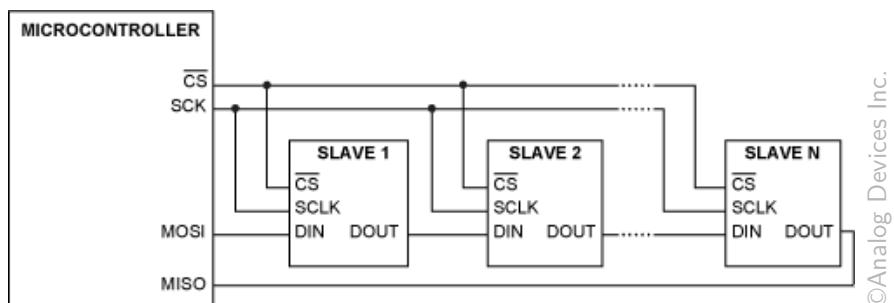
W pierwszej z nich dla  $N$  urządzeń podrzędnych wymagane jest  $N$  linii *Chip Select*. Na potrzeby tej pracy nadam tej metodzie nazwę "bezpośredniej", gdyż każde z urządzeń podrzędnych jest bezpośrednio wybrane do komunikacji przez urządzenie nadzorcze, które przełącza w tryb niski linię *Chip Select* danego urządzenia podrzecznego. Linia *MISO* jest wspólnie dzielona, więc w danym czasie wybrane może być dokładnie jedno z urządzeń. Wszystkie urządzenia podrzędne otrzymują w tym samym czasie te same dane, gdyż linia *MOSI* również jest wspólnie dzielona, ale na linii *MISO* odpowiada jedynie wybrane urządzenie. Rysunek 1.6 pomija linię *MISO*, gdyż częstym zastosowaniem architektury jest jednokierunkowe przesyłanie rozkazów do urządzeń podrzędnych.



Rysunek 1.6: Łącze bezpośrednie SPI

Druga z wykorzystywanych konfiguracji nie wymaga wykorzystania więcej niż jednej linii *Chip Select*, w zamian za nieco bardziej skomplikowany schemat komunikacji wywołany niejednolitą ilością cykli zegara obsługującą dane urządzenie podrzędne, w tej pracy otrzymuje miano metody "cyklicznej". W tym schemacie wejścia kolejnych urządzeń podrzędnych

są podłączone do wyjść poprzedzających urządzeń. Urządzenie nadzorcze wysyła serię komend dla wszystkich z urządzeń podrzędnych, wykorzystując cechę składania się przez nie w bufor kołowy, a następnie aktywuje je wszystkie jednocześnie zmieniając stan współdzielonej linii *Chip Select*. Urządzenia podrzędne bez zleconego zadania muszą otrzymać wartość zerową, która jest traktowana jako brak danych. Następnie urządzenie nadzorcze zbiera wszystkie odpowiedzi. Ten typ połączenia przedstawia rysunek 1.7.



©Analog Devices Inc.

Rysunek 1.7: Łącze cykliczne SPI

### 1.5.5 UART

*Universal Asynchronous Receiver-Transmitter* [20] to protokół komunikacyjny wykorzystujący dwie linie oraz wspólne uziemienie. Jest to protokół asynchroniczny, w związku z czym oba uczestniczące w transmisji urządzenia należy przygotować do komunikacji poprzez dobór parametrów komunikacyjnych na te same wartości, wspierane przez oba urządzenia, gdyż protokół nie wykorzystuje linii zegara synchronizującego. Linia *TX* jednego z urządzeń jest połączona do linii *RX* drugiego i vice versa.

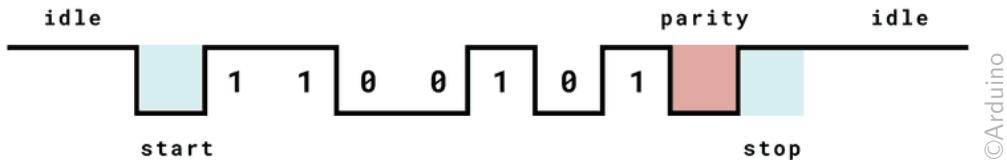
Ustawienia, które muszą mieć na urządzeniach zgodne wartości przedstawione zostały w tabeli 1.4.

Tabela 1.4: Opis parametrów *UART*

Parametr	Opis
Baud Rate	Prędkość transmisji (ilość czasu przypisana jednemu bitowi).
Data	Ilość bitów w wiadomości.
Parity	Tryb bitu parzystości (brak, parzysty, nieparzysty).
Stop Bits	Długość przerwy między wiadomościami.
Flow Control	Tryb współpracy z wolniejszymi urządzeniami.

Wiadomość *UART* (rysunek 1.8) składa się z zakończenia domyślnego trybu wysokiego

nieczynnej linii (stanem niskim) o czasie trwania równym jednemu bitowi, a następnie przesłanie 5 do 9 bitów wiadomości i opcjonalnego bitu parzystości. Następna wiadomość może zostać przesłana minimalnie po czasie odpowiadającym 1 do 2 bitom w stanie spoczynku.



Rysunek 1.8: Schemat wiadomości *UART*

©Arduino

## 1.6 Oprogramowanie

### 1.6.1 Raspberry Pi OS

*Raspberry Pi OS* [21] (dawniej *Raspbian*) to dystrybucja *GNU/Linux* bazująca na dystrybucji *Debian*, utworzona i przystosowana do wykorzystania z mikrokomputerami *Raspberry Pi*. W momencie pisania pracy najnowsza wersja z 1 października 2025 opiera się na *Debianie 13 Trixie*, ta właśnie wersja jest wspierana przez skrypt instalacyjny i środowisko laboratoryjne.

Dystrybucja ma warianty przystosowane do poszczególnych kategorii produktów *Raspberry Pi*, optymalizujące system do współpracy na urządzeniach o niskiej mocy obliczeniowej, ograniczonej wielkości, i prędkości dysku (karty SD), a także zawierające pomocne skrypty konfiguracyjne do periferii płytka. Jest to oficjalny system operacyjny płytka *Raspberry Pi*, w związku z czym można spodziewać się możliwie najlepszej stabilności i wsparcia w przypadku wystąpienia problemów.

### 1.6.2 Bash

*Bash* [22] to jedna z najpopularniejszych powłok (interpretatorów poleceń) wykorzystywanych w systemach *Unix*, *Linux*, *MacOS*, oraz *Solaris*. Poza obsługą poleceń użytkownika zawiera zaawansowane możliwości skryptowe, dzięki czemu może być wykorzystywana do szeroko rozumianej automatyzacji administracji systemem.

W pracy wykorzystana właśnie w tym celu, co ułatwia osobie prowadzącej zajęcia wstępna konfigurację, a także czyszczenie środowiska między zajęciami – załączony skrypt instalacyjny *pi-labs.sh* [F1] wykorzystuje możliwości tego narzędzia do pełnej konfiguracji płytki do zajęć przy użyciu jednej komendy.

### 1.6.3 Python

*Python* [23] to interpretowany, wysokopoziomowy język programowania. Jest znany z przejrzystej składni, łatwości nauki i rozbudowanego zestawu bibliotek, zarówno wbudowanych jak i utworzonych przez społeczność. Dzięki temu również ilość materiałów na temat wykorzystania języka jest bardzo duża.

Ekspresywność języka pozwala na wyrażenie szerokiego zakresu wyrażeń w prosty sposób, a także wykorzystanie wszystkich popularnych paradygmatów programowania, co jest dużą zaletą w trakcie nauki innych narzędzi – dzięki tej prostocie nie trzeba “walczyć” ze składnią języka, a można skupić się na przyswojeniu nowych informacji na temat nowego narzędzia.

Największą wadą języka jest prędkość interpretera, która nie pozwala na wykorzystania w zastosowaniach o krytycznej wydajności. Część osób jest również przeciwna użyciu przez język białych znaków do definicji bloków semantycznych.

### 1.6.4 rpi-Lgpio

Wiązania *rpi-Lgpio* [24] języka C do użytkowania interfejsu *GPIO* zastępują bibliotekę *rpi-gpio*. Wykorzystują te same nazwy importowane i sygnatury w celu zapewnienia kompatybilności wstępnej. Istnieją z powodu usunięcia interfejsu *sysfs* do *GPIO* z nowszych wersji jądra systemowego. Zamiast przestarzałego interfejsu *sysfs* wykorzystują nowszy plik urządzenia *gpiochip*. Podstawowe wykorzystanie prezentuje rysunek 1.9.

```
import RPi.GPIO as GPIO # Taki sam import jak wcześniejsza biblioteka
                        # rpi-gpio.

# Wybór trybu adresacji BCM, zgodny z rysunkiem w sekcji GPIO.
GPIO.setmode(GPIO.BCM)

# Pin 23 w adresacji BCM (16 w adresacji fizycznej) uruchomiony jako
# wyjściowy.
GPIO.setup(23, direction=GPIO.OUT)

# Ustawiony wysoki stan wyjścia (= 3,3V).
GPIO.output(23, value=GPIO.HIGH)

# Uwolnienie blokady pinów przy zakończeniu programu.
GPIO.cleanup()
```

Rysunek 1.9: Podstawowe wykorzystanie *GPIO* w języku *Python*

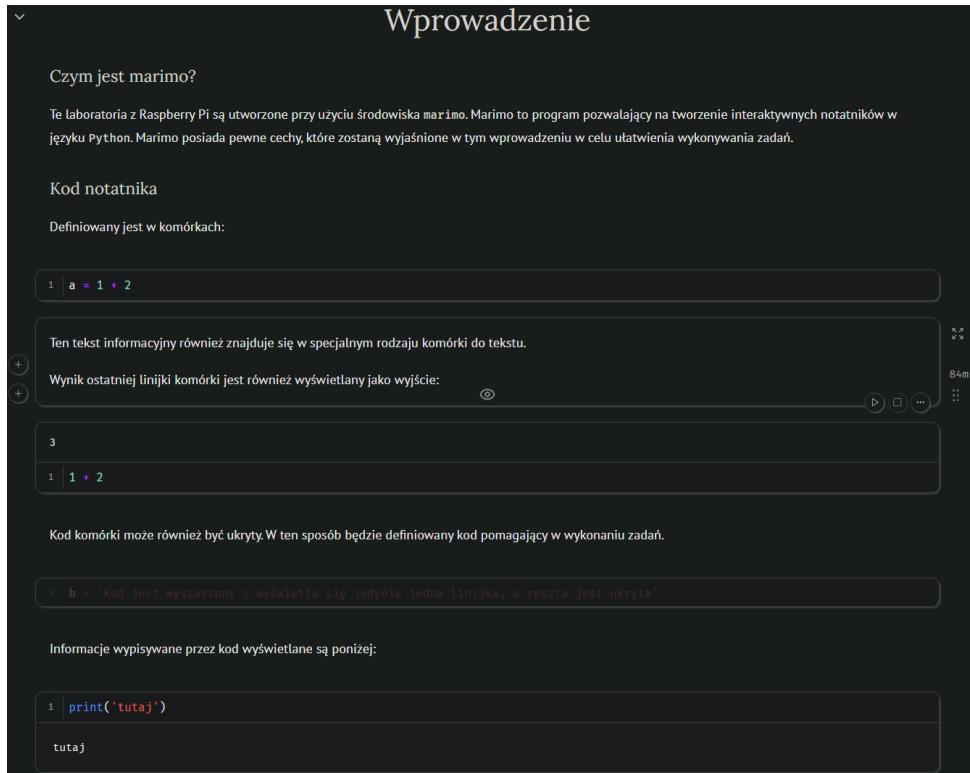
## 1.6.5 marimo

*Marimo* [25] jest środowiskiem notatnikowym *Python* mającym na celu usunięcie niektórych znaczących wad wcześniej istniejących rozwiązań [26] (tabela 1.5). Poza oferowaniem równie zaawansowanej funkcjonalności notatnika architektura projektu gwarantuje cechy wpływające na wygodę i stabilność użytkowania.

Tabela 1.5: Cechy *marimo* wynikające z braków w innych rozwiązańach

Cecha	Opis
Powtarzalność	Kod widoczny w notatniku <i>Jupyter</i> niekoniecznie odpowiada obecnemu stanowi projektu, co sprawia, że 36 % notatników nie jest powtarzalnych [27]. Wykorzystanie <i>marimo</i> rozwiązuje ten problem.
Łatwość utrzymania	Notatniki <i>marimo</i> są przechowywane jako standardowe programy <i>Python</i> , co pozwala m.in. na ich bezproblemowe wersjonowanie.
Interaktywność	<i>marimo</i> zawiera funkcjonalność tworzenia elementów interfejsu użytkownika, automatycznie synchronizowanych w kodzie przy zmianie wartości.
Wykorzystanie wtórne	Notatniki <i>marimo</i> mogą być wykorzystane z poziomu linii komend jak zwyczajne skrypty w języku <i>Python</i> , można również zimportować zdefiniowane w nich symbole do innych plików.
Udostępnianie	Każdy notatnik <i>marimo</i> może również być uruchomiony jako interaktywna aplikacja Internetowa wyświetlająca interfejs użytkownika bez możliwości edycji kodu.

Środowisko *marimo* wspiera zarówno tryb jasny, jak i ciemny interfejsu użytkownika; w tej sekcji znajduje się przykład trybu ciemnego (rys. 1.10), a w pozostałych przypadkach w celu lepszego dopasowania kolorystycznego rysunki będą wykorzystywać tryb jasny.



Rysunek 1.10: Tryb ciemny interfejsu graficznego *marimo*

## 1.7 Wdrożenie rozwiązania

### 1.7.1 Udostępnianie

Pierwotną metodą udostępnienia elementu technicznego tej pracy dyplomowej jest skrypt *Bash*, zawarty zarówno w tym pliku PDF, w Archiwum Prac Dyplomowych, jak i na mojej stronie Internetowej [28]. Zawiera on wszystkie informacje niezbędne do przygotowania i uruchomienia środowiska laboratoryjnego na płytce *Raspberry Pi* ze świeżo zainstalowanym systemem opartym na *Debianie 13 Trixie*. Dzięki użyciu stałych wersji programów ilość zmian niezbędnych do przystosowania skryptu do istniejących w przyszłości wersji systemu powinna być minimalna. Skrypt ten może być również użyty po zakończeniu laboratoriów do wyczyszczenia środowiska od zmian wprowadzonych przez studenta.

### 1.7.2 Pliki pomocnicze

Dla zapewnienia wstępnej wiedzy na temat współpracy ze środowiskiem *marimo*, poza laboratoriami część praktyczna pracy zawiera również plik wprowadzeniowy, którego fragment

znajduje się na powyższym rysunku 1.10. Opisuje on rozwiązania graficzne i interakcje możliwe do przeprowadzenia w środowisku.

Do pracy dołączony jest również instruktażowy plik łączenia modułów wraz z odpowiedziami na zadania laboratoryjne dla-prowadzącego.pdf [F2], pomagający osobie prowadzącej zajęcia połączenie elementów sprzętowych laboratoriów oraz pomoc w przypadku problemów z rozwiązyaniem zadania.

W celu pokazania pełnej treści laboratoriów dołączony jest również plik zawierający zdjęcia pełnej zawartości instrukcji studenckich instrukcje.pdf [F3].

### 1.7.3 Testy rozwiązań

Zadania pośrednie przygotowane dla studentów – wymagające napisania kodu – są automatycznie sprawdzane przy wykorzystaniu możliwości środowiska *marimo*. Pozwala to zaoszczędzić czas prowadzącego na pomoc indywidualną; sprawdzenie końcowego działania programu gwarantuje poprawne rozwiązywanie wcześniejszych zadań (lub indywidualną wiedzę studenta, jeśli rozwiązanie jest inne, ale również działa poprawnie).

Testy wykorzystują standardową bibliotekę języka, *unittest*, w celu największej przenosalności. Kod jest napisany w sposób niezawierający bezpośrednio prawidłowego rozwiązania, aby utrudnić możliwe oszukiwanie poprzez sprawdzenie rozwiązania w teście. Całkowite zlikwidowanie takich sytuacji nie jest możliwe, jednak również udowadnia wysoki poziom wiedzy studenta o języku *Python*.

### 1.7.4 Bezpieczeństwo

Przed pierwszym uruchomieniem skryptu z nieznanego źródła należy zapoznać się z jego treścią przy użyciu edytora tekstowego. Nie należy uruchamiać kodu, którego treść jest nam niezrozumiała. Dotyczy to załączonego do pracy skryptu instalacyjnego.

Skrypt zawiera środki testujące prawidłowość wykonania, jednak w każdym kodzie możliwe są błędy i niezaplanowane wykorzystania. Wykorzystane środki zapobiegawcze:

- Wykluczenie sprzętu mogącego uszkodzić płytę *Raspberry Pi* wyższym napięciem;
- zawarte w pracy, a także w nagłówku skryptu ostrzeżenie o nieuruchamianiu nieznanego kodu;
- test otrzymania uprawnień administratorskich;

- użycie folderu tymczasowego przez całą długość działania kodu;
- sprawdzenie istnienia zmian wykonywanych przez skrypt przed ich wykonaniem;
- wykorzystanie stałej wersji języka *Python* i bibliotek;
- wykorzystanie środowiska wirtualnego;
- załączona instrukcja połączeń dla prowadzącego zajęcia;
- załączony plik wprowadzający do wykorzystania środowisk notatnikowych;
- automatyczne testy prawidłowości wykonania zadań przez studenta;
- wyżej wymienione testy nie uruchamiają kodu studenta.

## Rozdział 2

---

# Laboratoria

---

## 2.1 Laboratorium I

### 2.1.1 Wybrany sprzęt

Oficjalny moduł Fundacji Raspberry Pi – *Sense HAT* – zawiera żyroskop, akcelerometr, magnetometr, barometr, termometr, higrometr, czujnik oświetlenia, joystick, oraz matrycę  $8 \times 8$  diod LED RGB [29].

Część dostępnej funkcjonalności pokrywa się z pozostałymi dostępnymi modułami, w związku z czym do tego laboratorium wykorzystana została funkcjonalność *Inertial Measurement Unit* – żyroskop, akcelerometr, magnetometr, a także matryca pikseli w celu symulacji działania kompasu.

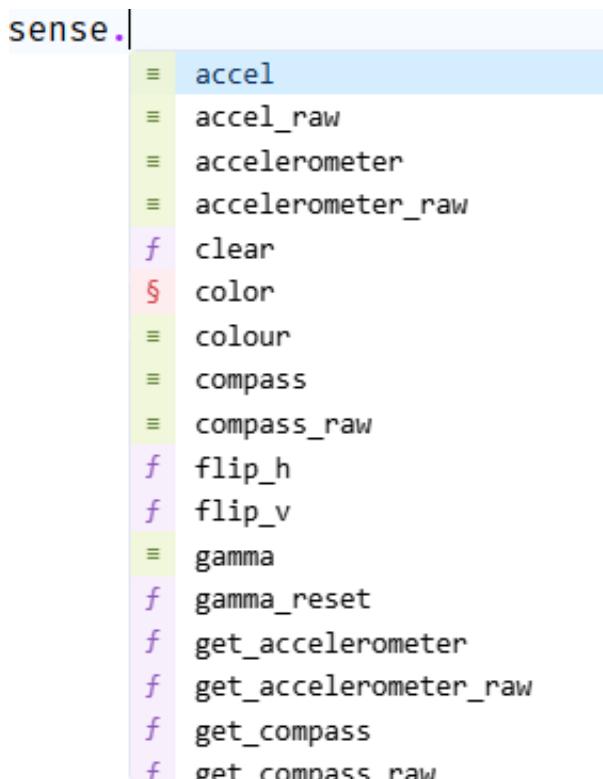
Z powodu dużych zakłóceń magnetycznych spowodowanych obecnością metalowych elementów w większości pomieszczeń uczelni, utworzony kompas nie będzie bardzo dokładny, jednak dostatecznie pokazuje działanie podobnych urządzeń np. w telefonach czy technologiiach *Full Body Tracking* popularyzowanych w kręgach fanów wirtualnej rzeczywistości. Rozwiązania te poza zdecydowanie bardziej wyspecjalizowanym doborze elementów sprzętowych i bardziej zaawansowanym oprogramowaniem kompensującym wpływ czynników zewnętrznych opierają się fundamentalnie na tej samej zasadzie działania.

W celu redukcji niedokładności instrukcja zawiera zadanie kalibracyjne, jego wynik zależy jednak od warunków środowiska i dokładności wykonania, więc jest jedynie krokiem referencyjnym pokazującym istnienie podobnego procesu po stronie producenta zintegrowanych urządzeń.

## 2.1.2 Dodatkowe oprogramowanie

W tym laboratorium wykorzystana jest oficjalna biblioteka *sense\_hat* [30]. Jest to zalecany i najprostszy sposób wykorzystania możliwości sensorów modułu *Sense HAT*, gdyż biblioteka zapewnia wysokopoziomowe interfejsy do obsługi zarówno czujników, jak i matrycy. Dzięki temu wykorzystanie sprzętu jest bliżej paradygmatowi deklaratywnemu, bez potrzeby zrozumienia niskopoziomowych detali komunikacji przeprowadzanej przez urządzenia.

Zastosowanie tej biblioteki stanowi łagodne wprowadzenie do dalszych zadań realizowanych w środowisku marimo. Głównym jej elementem jest klasa *SenseHat*, która zbiera zdefiniowane funkcje pod jedną nazwą, pozwalając środowisku marimo na wyświetlanie podpowiedzi w interfejsie graficznym (rys. 2.1).



Rysunek 2.1: Podpowiedzi w interfejsie graficznym marimo

## 2.1.3 Rozwiążanie

Rysunek 2.2 to fragment instrukcji laboratoryjnej opisujący pierwsze z zadań, jakim jest przeprowadzenie procesu kalibracji sensorów inercyjnych. Pokazuje on i wyjaśnia kolejne kroki niezbędne do wykonania podstawowego procesu kalibracji, co znacząco zwiększa dokładność

późniejszych pomiarów oraz stabilność danych. Instrukcja zwraca uwagę na prawidłowe ułożenie urządzenia w celu ograniczenia zakłóceń magnetycznych.

### Z1. Kalibracja IMU

Aby użyć czujników inercyjnych (IMU), do których zalicza się żyroskop, akcelerometr i magnetometr, należy najpierw je skalibrować. Można to zrobić przy użyciu (już zainstalowanej) biblioteki systemowej `octave`.

Otwórz terminal i skopiuj domyślne pliki konfiguracyjne sensora do folderu studenta: `cp -a /usr/share/librtimulib-utils/RTEllipsoidFit ~`

Przejdź do skopowanego folderu: `cd ~/RTEllipsoidFit`

W tym folderze znajdują się pliki niezbędne programowi kalibracyjnemu do przetworzenia danych otrzymanych podczas kalibracji na konfigurację prawidłowo dostosującą dane generowane w trakcie korzystania z czujników. Można dostosować te domyślne wartości dla specjalistycznych potrzeb, jednak w tym przypadku w zupełności wystarczą w niezmienionej formie.

Aby uruchomić program kalibracyjny, użyj komendy: `RTIMULibCal`

Program ten posiada prosty interfejs w języku angielskim. Wybierz opcję `m` i naciśnij dowolny przycisk, aby rozpocząć kalibrację, która odbywa się poprzez poruszanie sensorem (a zatem i płytka *Raspberry Pi*). Spróbuj obracać urządzenie w taki sposób, aby w trakcie kalibracji znalazło się w każdej możliwej orientacji. Kalibracja kończy się również poprzez naciśnięcie dowolnego przycisku.

Wybierz opcję `x` aby wyjść z programu kalibracyjnego.

Domyślnie plik konfiguracyjny jest zapisywany w lokalnym folderze użytkownika, jednak aby używać prawidłowej kalibracji we wszystkich programach, przenieś go w miejsce konfiguracji systemowej: `sudo mv ~/.config/sense_hat/RTIMULib.ini /etc`

Czujniki IMU potrzebują chwili, aby uruchomić się i skalibrować po uruchomieniu systemu. Po następnym kroku uruchamiającym system ponownie, upewnij się, że urządzenie przez 6-10 sekund po uruchomieniu (~30 sekund łącznie) nie poruszy się, aby uzyskać najdokładniejsze rezultaty.

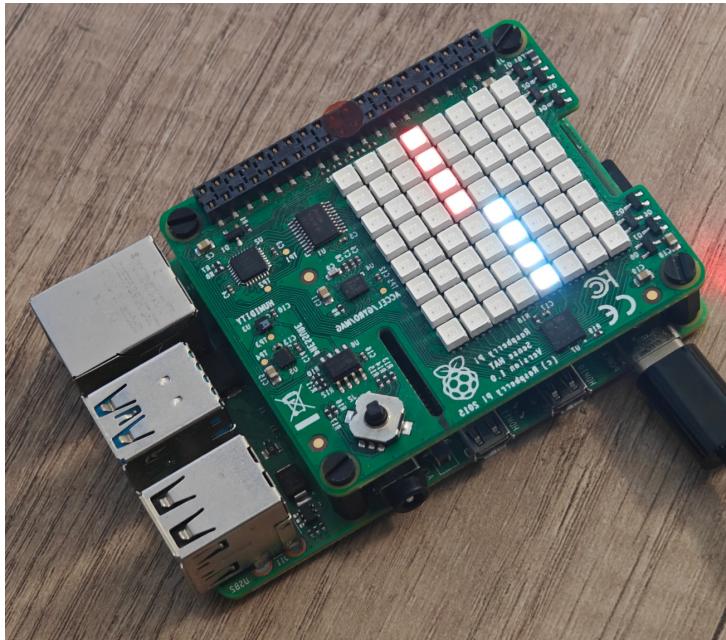
Po tym kroku należy zrestartować programy używające kalibracji, w tym również środowisko tego laboratorium, więc zapisz zmiany wykonane w tym pliku, a następnie użyj komendy do uruchomienia systemu ponownie: `sudo reboot`

---

Dzięki prawidłowo skalibrowanym sensorom odczyty będą bardziej odporne na wpływ magnetyzowanych materiałów w środowisku, choć mocne zakłócenia nadal mogą spowodować błędy w odczycie informacji. Staraj się trzymać sensor z dala od urządzeń elektronicznych i elementów ferromagnetycznych w trakcie kalibracji i obsługi czujników.

Rysunek 2.2: Fragment instrukcji na temat *Sense HAT*

Na rysunku 2.3 zaprezentowany jest wynik uruchomienia rozwiązanego laboratorium. Podobnie do standardowego kompasu kierunki północy i południa są reprezentowane w kolorach czerwonym i białym.



Rysunek 2.3: Uruchomiony program pierwszego laboratorium

## 2.2 Laboratorium II

### 2.2.1 Wybrany sprzęt

System modułów *Grove* jest wykorzystany w dwóch laboratoriach, w związku z czym jego opis znajduje się w sekcji przodujączej (1.4.2). W niniejszym laboratorium zostały użyte dwa urządzenia współpracujące z modułem *Grove Base HAT*.

Głównym elementem zestawu laboratoryjnego jest czujnik wykorzystujący czas propagacji fal ultradźwiękowych do pomiaru odległości. Sensor emituje krótki sygnał ultradźwiękowy, a następnie nasłuchuje momentu powrotu echa fali od przeszkody, co pozwala na obliczenie dystansu przebytego przez falę, a zarazem odległości od przeszkody.

Do wyświetlenia wyników pomiaru wybrany został wyświetlacz LCD  $2 \times 16$  komórek o wielkości komórki  $8 \times 5$  pikseli. Długość 16 komórek pozwala na wyświetlenie mierzonej wartości wraz z jednostką i krótkim opisem, a druga linia może służyć do wyświetlenia danych pomocniczych lub drugiego rekordu. Wyświetlacz ten jest zaprojektowany do obsługi tekstu i nie można wykorzystać pikseli do dowolnego rysowania.

## 2.2.2 Dodatkowe oprogramowanie

*Seeed-grove.py* to oficjalna biblioteka programistyczna zestawu *Grove* w języku *Python*. Niestety nie jest ona dostępna przy wykorzystaniu standardowego repozytorium bibliotek języka. Kod biblioteki można zobaczyć jednak na platformie *GitHub* [31], dzięki czemu nadal możliwa jest prosta instalacja (rys. 2.4) przy użyciu stosunkowo nowych wersji oficjalnego menedżera bibliotek.

```
pip install git+https://github.com/Seeed-Studio/grove.py.git
```

Rysunek 2.4: Instalacja biblioteki *grove* z platformy *GitHub*

## 2.2.3 Rozwiązanie

Rysunek 2.5 to fragment instrukcji przedstawiający dwa zadań składających się na końcowy program mierzący odległość do obiektu przy wykorzystaniu sensora ultradźwiękowego. Pierwsze zadanie polega na uzupełnieniu ciała funkcji wykorzystując podaną zmienną i podstawowe prawa fizyki w celu otrzymania wartości odległości zmierzanej przez czujnik ultradźwiękowy. Drugie wyświetla obliczone wartości na wyświetlaczu LCD – z powodu szybkiego odświeżania wykorzystując wyrównanie danych bez czyszczenia ekranu funkcją *clear*.

#### Z4. Przetwarzanie danych

Sensor ultradźwiękowy zwraca jedynie czas od wysłania sygnału do otrzymania echa. Uzupełnij funkcję przyjmującą czas w mikrosekundach, tak, aby zwracała odległość w centymetrach. Wykorzystaj znajomość praw fizyki (prędkość dźwięku). Pamiętaj, że fala dźwiękowa dociera do celu, odbija się i wraca z powrotem do sensora.

```
1 v def time_to_dist(t: float) -> float:  
2     sekundy = t / 10**6  
3     metry = sekundy * 343 # m/s  
4     centymetry = metry * 100  
5     jedna_strona = centymetry / 2  
6     return jedna_strona
```

time\_to\_dist

Zadanie Z4. jest rozwiązane.

```
1 mo.md(f'Zadanie Z4. {"if test_z4(time_to_dist) else "nie"} jest rozwiązane.')
```

#### Z5. Wyświetlanie informacji

Napisz funkcję wyświetlającą wynik pomiaru odległości na panelu LCD. Wyświetl również poprzednią wartość w drugiej linijce.

Podpowiedź:

Funkcja `display.setCursor` wyznacza miejsce, gdzie umieszczona zostanie treść wiadomości.

Wartość zmiennych powinna zajmować zawsze tyle samo pól.

```
1 v def show_measurement(val: float, prev: float):  
2     display.setCursor(0, 0)  
3     display.write(f'cur: {val:.2f} cm')  
4     display.setCursor(1, 0)  
5     display.write(f'prv: {prev:.2f} cm')
```

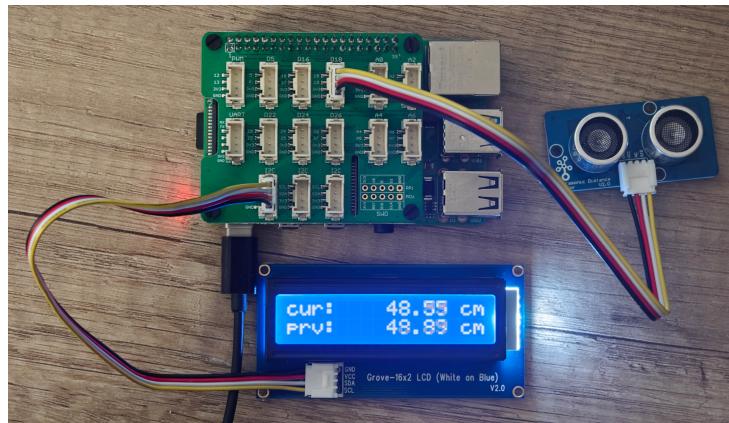
show\_measurement

Zadanie Z5. jest rozwiązane.

```
1 mo.md(f'Zadanie Z5. {"if test_z5(show_measurement) else "nie"} jest rozwiązane.')
```

Rysunek 2.5: Fragment instrukcji II

Rysunek 2.6 przedstawia rozwiązanie zadania. W tym przypadku podana odległość jest do telefonu, którym utworzone zostało zdjęcie, a wyświetlacz jest w trakcie zmiany wyświetlanej wartości.



Rysunek 2.6: Uruchomiony program drugiego laboratorium

## 2.3 Laboratorium III

### 2.3.1 Wybrany sprzęt

Podobnie do poprzedniego laboratorium, w tym przypadku również wykorzystany jest *Grove Base Kit*. Dzięki umieszczeniu tych laboratoriów sąsiednio w kolejności numerycznej oszczędzona jest niewielka ilość czasu na zmiany połączeń fizycznych.

Głównym elementem laboratorium jest czujnik ruchu *mini PIR S16-L221D* kompatybilny ze środowiskiem Grove. Czujnik ten umożliwia wykrywanie ruchu obiektów emitujących promieniowanie podczerwone. Podłączony jest do portu cyfrowego, jego użycie polega na analizie zboczy sygnału wyjściowego.

W laboratorium użyty został również przycisk z diodą LED. Dioda została wykorzystana do sygnalizacji wykrycia ruchu przez czujnik, przycisk resetuje jej stan. W ten sposób program wynikowy pełni rolę prostego systemu alarmowego, powiadamiającego o uprzednim wystąpieniu ruchu w obserwowanej strefie.

### 2.3.2 Rozwiążanie

Rysunek 2.7 to fragment instrukcji przedstawiający zadanie wykrycia wciśnięcia przycisku wyłączającego diodę. Oczywiste rozwiązanie zawiera pewną niedogodność, która ujawnia się po uruchomieniu pętli głównej z dużym opóźnieniem czasowym. Interakcja z czujnikiem ruchu, pomimo podobnego kodu, nie ma tego problemu dzięki wykorzystaniu mechanizmu flagi.

### Z3. Wyłączenie alarmu

Wciśnięcie przycisku przez użytkownika powinno wyłączyć alarm.

Przycisk nie wykorzystuje mechanizmu flagi, natomiast daje dostęp do obecnego stanu wciśnięcia.

Dane pomocnicze:

`btn.pressed` - stan wciśnięcia przycisku

`btn.led` - stan diody przycisku, można go zmieniać nadając wartość tej zmiennej

```
1 v def btn_pressed():
2 v     if btn.pressed:
3 v         btn.led = False
```

btn\_pressed

Zadanie Z3. jest rozwiązyane.

```
1 mo.md(f'Zadanie Z3. {"if test_z3(btn_pressed) else "nie"} jest rozwiązane.')
```

### Z4. Pętla główna

Napisz pętlę aktualizującą diodę przycisku co 0.1 sekundy. Przetestuj działanie alarmu, następnie zmień opóźnienie na 3 sekundy i przetestuj ponownie. Jakie różnice istnieją między mechanizmem flagi a bezpośrednim odczytem stanu?

To zadanie jest sprawdzane przez prowadzącego.

Dane pomocnicze:

`motion_detected()` - aktualizacja diody przy wykryciu ruchu

`btn_pressed()` - aktualizacja diody przy wciśnięciu przycisku

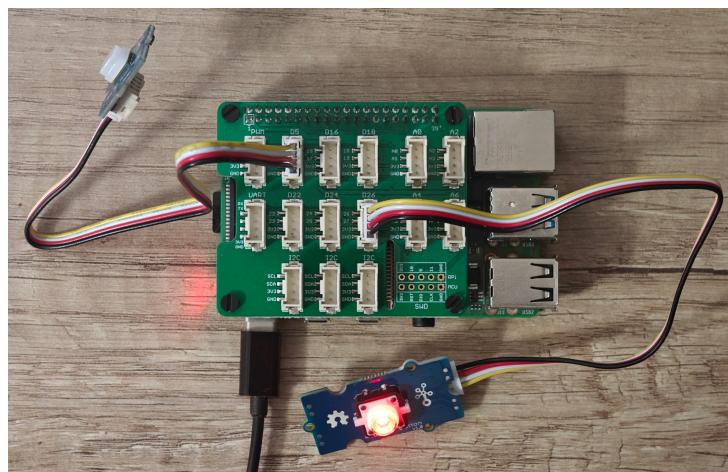
`await sleep(delay: float)` - oczekивание przez `delay` sekund

Uruchom

```
1 stp = mo.ui.run_button(label='Uruchom')
```

Rysunek 2.7: Fragment instrukcji III

Rysunek 2.8 przedstawia rozwiązanie instrukcji. Czujnik ruchu został wywołany i w wyniku tej akcji została uruchomiona dioda. Z powodu interaktywności zadania zdjęcie nie jest idealnym sposobem przedstawienia działania.



Rysunek 2.8: Uruchomiony program trzeciego laboratorium

## 2.4 Laboratorium IV

### 2.4.1 Wybrany sprzęt

*Enviro HAT* to moduł zawierający pełny zestaw sensorów niezbędnych do utworzenia małej stacji pogodowej. Wykorzystuje standardowe wielofunkcyjne elementy wymienione w tabeli 2.1.

Tabela 2.1: Układy scalone składające się na moduł *Enviro*

Układ	Opis
BME280	Zawierający czujniki temperatury, ciśnienia, i wilgotności.
LTR-559	Zawierający sensor jasności otoczenia i zbliżeniowy.
SPH0645LM4H-B	Mikrofon mikroukładowy $I^2S$ .
ST7735	Sterownik wyświetlacza LCD $160 \times 80$ pikseli.

### 2.4.2 Dodatkowe oprogramowanie

Wykorzystanie sensorów w kodzie jest wzorowane na bibliotece do nowszej płytki *Enviro+* [32]. W związku z aktualizacją kodu tej biblioteki z *Python 2* na *Python 3*, jej struktura nie jest zgodna z najnowszymi rekomendacjami stylistycznymi, biblioteka skupia się również w głównej części na sensorach niedostępnych na starszej wersji płytki w posiadaniu uczelni.

W trakcie pisania tej pracy została również znaleziona i zgłoszona niekompletność dokumentacji wymagań modułu [33].

W związku z tymi trudnościami kod laboratoriów wykorzystuje bardziej niskopoziomowe biblioteki i reimplementuje funkcjonalność dostępną w oficjalnych źródłach przy wykorzystaniu bibliotek wymienionych w tabeli 2.2.

Tabela 2.2: Biblioteki języka *Python* wykorzystane w laboratorium IV

Biblioteka	Opis
bme280	Biblioteka wspierająca interakcje z sensorami udostępnianymi przez układ <i>BME280</i> .
ltr559	Biblioteka wspierająca wykorzystanie sensorów układu <i>LTR-559</i> .
sounddevice	Standardowa biblioteka wykorzystywana do interakcji z urządzeniami audio w języku Python, w tym przypadku z układem <i>SPH0645LM4H-B</i> .
st7735	Biblioteka obsługująca wyświetlacz ciekłokrystaliczny <i>ST7735</i> .
font-roboto	Biblioteka dostarczająca rodzinę czytelnych czcionek <i>Roboto</i> firmy Google do wykorzystania w języku <i>Python</i> .
pillow	Standardowa biblioteka <i>Python</i> zawierająca funkcjonalności przetwarzania obrazów.

### 2.4.3 Rozwiązanie

Rysunek 2.9 to fragment instrukcji zawierający dwa zadania. Zadanie Z3. wykorzystuje platformę *ntfy.sh* do wysłania powiadomienia w momencie przykrycia czujnika zbliżeniowego na płytce, informacje zawarte w tej wiadomości przedstawia rysunek 2.10. W zadaniu Z4. student wykorzystuje poprzednio uzupełnione funkcje do złożenia pętli głównej obsługującej program założony w temacie laboratorium.

### Z3. Publikowanie danych

Zgodnie z wcześniejszym opisem, dane będą również wysyłane przy użyciu serwisu `ntfy.sh` ([dokumentacja](#)), co pozwoli na otrzymywanie powiadomień. Uzupełnij funkcję, aby wysłać najnowsze dane.

Dane znajdują się w słowniku o nazwie `values`. Kluczami słownika są **nazwy metryk**. Wyslij najnowszą wartość każdej metryki jako typ danych `float`. Wykorzystaj parametr `json` funkcji `requests.post` (już zainportowanej pod nazwą `post`) i zmiennej `ntfy_topic`.

Wysyłaj wiadomości tylko jeśli użytkownik dotycza sensora zbliżeniowego.

Twoja nazwa tematu (zmienna `ntfy_topic`): lab-e45f01472278.

```
1 mo.md(f'Twoja nazwa tematu (zmienna `ntfy_topic`): `{ntfy_topic}`.')
```

```
1 v def send_message():
2 v     if values['prox'][-1] < 0.1:
3 v         post(f'https://ntfy.sh/{ntfy_topic}', json={
4 v             k: float(v[-1]) for k, v in values.items()
5 v         })
```

send\_message

Zadanie Z3 jest rozwiązane.

```
1 mo.md(f'Zadanie Z3 {"if test_z3(send_message) else "nie"} jest rozwiązane.')
```

### Z4. Pętla główna

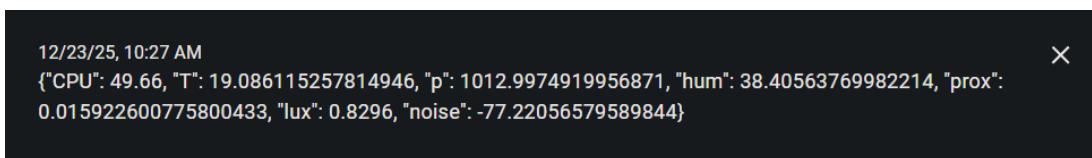
Zbiór, przetwarzanie, wysyłanie i wyświetlanie informacji powinno być stałym procesem, który nie odbywa się jedynie jeden raz.

Napisz pętlę główną programu, która odbywa się co sekundę. Jako opóźnienie wykorzystaj `await sleep(1)`.

Funkcje pomocnicze:

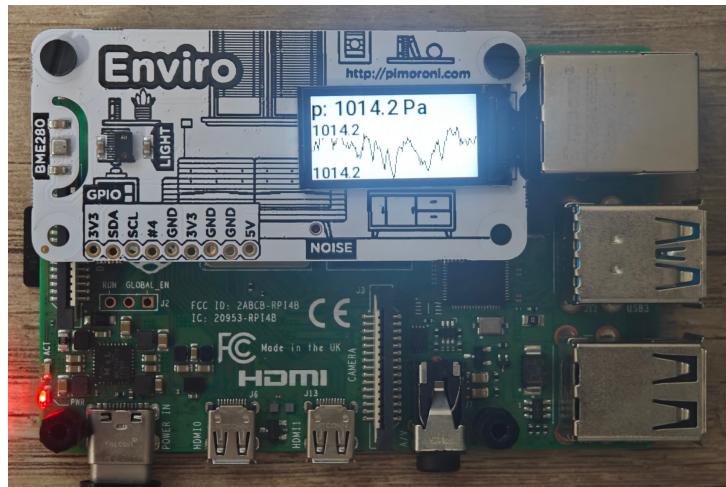
`await sleep(1)` - 1 sekunda opóźnienia (w trybie asynchronicznym, który jest używany w tym środowisku laboratoryjnym)  
`data_loop(list[Metric], times: int = 5) -> Metric` - jest to nieskończony iterator, która zwraca kolejne metryki po `times` razy, co pomaga w wygodnym wyświetleniu ich na ekranie  
`insert_vals(list[Metric])` - funkcja zbierająca i dopisująca wartości metryk do słownika `values`  
`display_text(Metric)` - funkcja wyświetlająca na ekranie dane metryki  
`update_sensors()` - wcześniej utworzona funkcja aktualizująca dane sensorów, pamiętaj, że pierwsze wywołanie zwraca nieprawidłowe dane  
`send_message()` - wcześniej utworzona funkcja wysyłająca dane

Rysunek 2.9: Fragment instrukcji IV



Rysunek 2.10: Powiadomienie pogodowe na platformie `ntfy`

Rysunek 2.11 przedstawia rozwiązanie zadania. Program wyświetla kolejno zbierane dane wraz z wykresami wcześniejszych wartości. Rysunek ten jest złożeniem dwóch zdjęć o różnych ekspozycjach w celu widoczności zarówno ekranu, jak i płytka,



Rysunek 2.11: Uruchomiony program czwartego laboratorium

## 2.5 Laboratorium V

### 2.5.1 Wybrany sprzęt

Do tego laboratorium wykorzystano moduł *Traffic pHAT*. Moduły *pHAT* różnią się od *HAT* w taki sposób, że nie posiadają czipa kontrolnego powiadamiającego płytę *Raspberry Pi* o swoich właściwościach. W związku z tą cechą, moduły *pHAT* należy traktować jako zbiór osobnych elementów elektronicznych w przenośnej formie, niżli bardziej zaawansowane urządzenie.

### 2.5.2 Rozwiązanie

Rysunek 2.12 to fragment instrukcji laboratoryjnej zawierający zadania związane z wykorzystaniem serwisu *ntfy* do przekazywania wiadomości i złożenia poprzednio uzupełnionych funkcji w program wypełniający rolę zawartą w nazwie laboratorium – system sygnalizacji świetlnej skrzyżowania.

## Z5. Wysyłanie wiadomości

Dana płytka ma zamocowany tylko jeden zestaw świateł sygnalizacyjnych W związku z tym do przetestowania systemu sygnalizacji będzie potrzebna praca w parach, jednak można najpierw skonfigurować i przetestować komunikację przy wykorzystaniu jednego światła. Do komunikacji podobnie jak w poprzednim laboratorium użyty zostanie serwis `ntfy.sh` ([dokumentacja](#)).

Nazwa twojego kanalu to: `lab-e45f01472278`. W kodzie dostępna jest zmienna `ntfy_topic`, która ją przechowuje.

W celu zaliczenia zadania usuń znak komentarza (#) z poniższej linijki i po uruchomieniu kodu odbierz wiadomość o treści 'zaliczono'. Wiadomość wyślij z terminala przy użyciu `curl`.

```
1 v mo.md(fnnn)
```

Zadanie Z5. jest rozwiązane.

```
1 receive_task(duration=30)
```

```
    wiadomość:  
    wiadomość: zaliczono
```

## Z6. Skrzyżowanie

To zadanie będzie wymagało testowania przy użyciu dwóch płyt. Jedna przyjmie rolę nadzędnej i będzie wysyłać wiadomości na swój kanał, a druga podrzędnej i będzie słuchać wiadomości **na kanale pierwszej**. Wiadomości powinny informować podrzędną płytę kiedy wywoływa funkcje `red_to_green` i `green_to_red`.

Domyślny stan diód to światło zielone dla płytki nadzędnej i czerwone dla płytki podrzędnej.

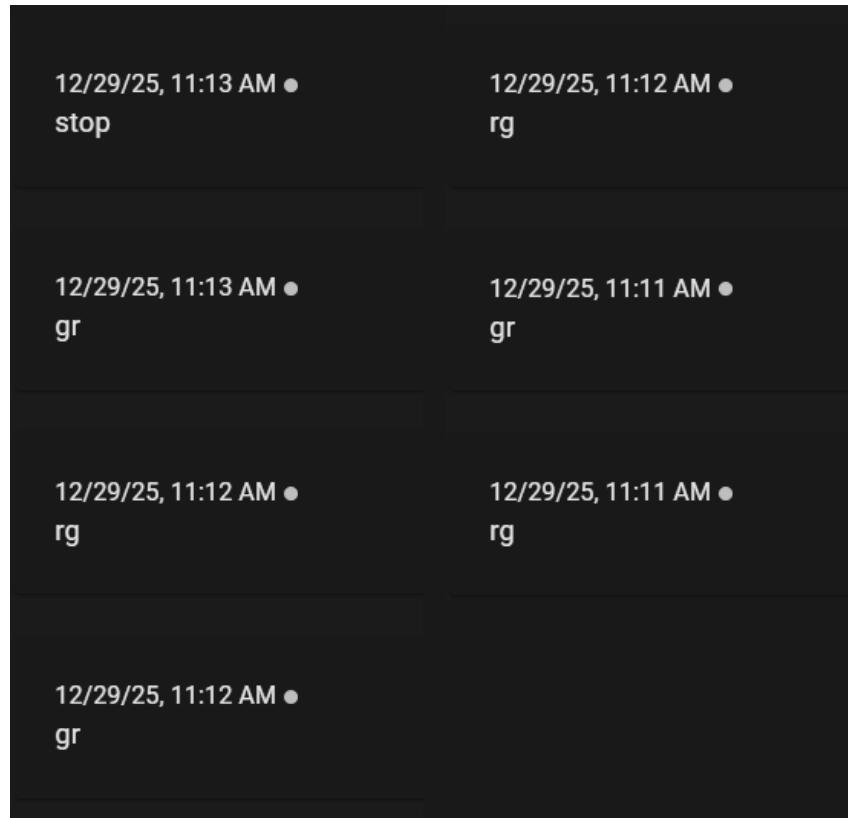
Funkcja odbierająca **musi** obsługiwać wiadomość **zakończającą pętlę**, a funkcja wysyłająca **nie może** być nieskończona, aby ułatwić sprawdzanie zadania.

Wybierz drugą osobę, z którą przetestujesz odpowiedź. Uzupełnijcie funkcje wykonujące obie z ról wspomagając się dokumentacją `ntfy.sh`. Po przetestowaniu działania jednej strony zamieńcie role płyt, przetestujcie drugą część zadania, i zgłoście zakończenie ćwiczenia.

**Podpowiedź:** Obie części wymagają uruchomienia w pętli, która w przypadku płytki podrzędnej będzie stale nasłuchiwać i obsługiwać otrzymane wiadomości, a w przypadku płytki nadzędnej w odpowiednich momentach wysyłać komunikaty i zmieniać stan własny, aby sygnalizacja była zsynchronizowana. Pamiętaj, że światło zielone ani żółte nie może się pojawić na obu światłach jednocześnie.

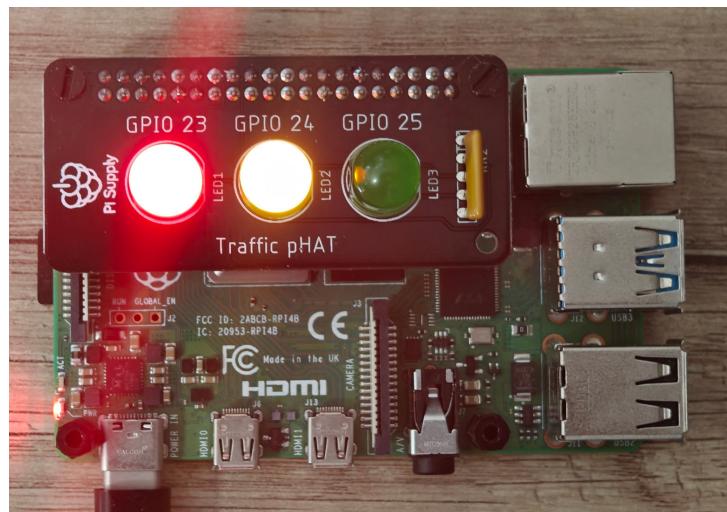
Rysunek 2.12: Fragment instrukcji V

Na rysunku 2.13 przedstawione są wiadomości, które płytka nadzorcza wysyła w celu kontroli stanu oświetlenia na płytce podrzędnej. W przykładowym rozwiązaniu wiadomości te zawierają jedną ze zdefiniowanych możliwych wartości uruchamiających poszczególne przejścia stanu – czerwony-zielony, zielony-czerwony, zakończenie działania programu.



Rysunek 2.13: Powiadomienia kontrolne na platformie *ntfy*

Faza druga czterofazowego systemu świetlnego – moment zmiany na światło zielone – jest przedstawiona na rysunku 2.14.



Rysunek 2.14: Uruchomiony program piątego laboratorium

## Rozdział 3

---

### Podsumowanie

---

Niniejsza praca inżynierska implementuje zestawy laboratoryjne wprowadzające studentów do wykorzystania peryferiów *GPIO* z komputerem jednopłytkowym *Raspberry Pi*. W tym celu dobrane moduły sprzętowe, a także rozwiązania architektury oprogramowania są dopasowane do warunków uczelni, pozwalając zarówno na szybkie i nieskomplikowane przygotowanie urządzeń do zajęć – co ogranicza nakład powtarzalnych czynności dla osoby prowadzącej zajęcia – jak i dokładnie udokumentowane i spójne środowisko udostępnione studentom. Laboratoria wykorzystują różne moduły fizyczne i w różny sposób wchodzą z nimi w interakcje z poziomu kodu, co pozwala studentom na nabycie unikalnych umiejętności w każdym z zestawów.

Poprawne rozwiązania zadań zostały przetestowane i umieszczone w załączniku dla prowadzącego, co w razie wątpliwości pozwoli na uzupełnienie wszystkich przygotowanych zadań. Skrypt instalacyjny zawiera bezpośrednio kod notatnika, dzięki czemu jakiekolwiek zmiany lub uzupełnienia treści laboratoriów mogą być zintegrowane do dalszych zajęć kopiując nową treść pomiędzy znaczniki graniczące zawartość pliku – rysunek 3.1. W ten sam sposób można rozszerzyć zakres pracy o nowe laboratoria, zwiększając zakres dostępnego materiału.

```
cat >laboratorium.txt <<'END'

Treść poniżej komendy cat znajdzie się w pliku.
Podana w linii komendy nazwa stanowi nazwę utworzonego pliku.
Znacznik END zakończa treść pliku i pozwala na uruchomienie dalej
→ zawartych komend.

END
```

Rysunek 3.1: Populacja treści pliku

W celu zapewnienia względów bezpieczeństwa środowisko laboratoryjne jest dostępne jedynie z lokalnej maszyny, jednak oprogramowanie *marimo* wspiera uruchamianie za wstecznym proxy, co może pozwolić na rozszerzenie projektu o prowadzenie zajęć zdalnych.

Obecnie sprawozdania pełnią funkcję archiwizacji pracy studentów, ale ciekawą opcją do rozważenia mogłoby być wykorzystanie uzupełnionych notatników laboratoryjnych w tym celu, gdyż wielkość tych plików jest niewielka, a zawierają już informacje powielane przez studentów do edytora dokumentów *Word* za pomocą zrzutów ekranu.

---

# Bibliografia

---

- [1] [raspberrypi.com/products](http://raspberrypi.com/products). [str. 7]
- [2] [raspberrypi.com/products/sense-hat](http://raspberrypi.com/products/sense-hat). [str. 7]
- [3] [seeedstudio.com/Flick-HAT-3D-Tracking-Gesture-HAT-for-Raspberry-Pi.html](http://seeedstudio.com/Flick-HAT-3D-Tracking-Gesture-HAT-for-Raspberry-Pi.html). Sklep nieoficjalny. [str. 7]
- [4] [seeedstudio.com/Grove-Base-Kit-for-Raspberry-Pi-p-2945.html](http://seeedstudio.com/Grove-Base-Kit-for-Raspberry-Pi-p-2945.html). [str. 7]
- [5] [shop.pimoroni.com/products/automation-hat-mini](http://shop.pimoroni.com/products/automation-hat-mini). [str. 7]
- [6] [shop.pimoroni.com/products/enviro?variant=31155658489939](http://shop.pimoroni.com/products/enviro?variant=31155658489939). [str. 7]
- [7] [shop.pimoroni.com/products/unicorn-hat](http://shop.pimoroni.com/products/unicorn-hat). [str. 7]
- [8] [anodas.lt/en/traffic-phat-led-overlay-for-raspberry-pi-zero-pi-supply-pis-1778](http://anodas.lt/en/traffic-phat-led-overlay-for-raspberry-pi-zero-pi-supply-pis-1778). Sklep nieoficjalny. [str. 7]
- [9] [find-and-update.company-information.service.gov.uk/company/08518826](http://find-and-update.company-information.service.gov.uk/company/08518826).  
[str. 8]
- [10] [banana-pi.org/banana-pi-sbcs](http://banana-pi.org/banana-pi-sbcs). [str. 8]
- [11] [hardkernel.com/product-category/odroid-board](http://hardkernel.com/product-category/odroid-board). [str. 8]
- [12] [libre.computer/products](http://libre.computer/products). [str. 8]
- [13] [orangepi.org/html/hardWare/computerAndMicrocontrollers/index.html](http://orangepi.org/html/hardWare/computerAndMicrocontrollers/index.html).  
[str. 8]
- [14] [pinout.xyz](http://pinout.xyz). [str. 9]
- [15] [wiki.seeedstudio.com/Grove\\_Base\\_Hat\\_for\\_Raspberry\\_Pi](http://wiki.seeedstudio.com/Grove_Base_Hat_for_Raspberry_Pi). [str. 9]

- [16] electronicspost.com/generation-and-detection-of-a-pwm-signal. [str. 11]
- [17] soldered.com/learn/what-is-the-i2c-communication-protocol. [str. 12]
- [18] wikipedia.org/wiki/I2S. [str. 13]
- [19] analog.com/technical-articles/daisychaining-spi-devices.html. [str. 13]
- [20] docs.arduino.cc/learn/communication/uart. [str. 15]
- [21] raspberrypi.com/software/operating-systems. [str. 16]
- [22] gnu.org/software/bash. [str. 16]
- [23] python.org. [str. 17]
- [24] rpi-lgpio.readthedocs.io. [str. 17]
- [25] marimo.io. [str. 18]
- [26] docs.marimo.io/faq#choosing-marimo. [str. 18]
- [27] blog.jetbrains.com/we-downloaded-10-000-000. [str. 18]
- [28] dot.aonodensetsu.me/pi-labs.sh. [str. 19]
- [29] raspberrypi.com/documentation/accessories/sense-hat.html. [str. 22]
- [30] sense-hat.readthedocs.io. [str. 23]
- [31] github.com/Seeed-Studio/grove.py. [str. 26]
- [32] github.com/pimoroni/enviroplus-python. [str. 30]
- [33] github.com/pimoroni/enviroplus-python/issues/157. [str. 30]

---

# Spis rysunków

---

1.1	Układ pinów <i>Raspberry Pi</i> – adresacja <i>BCM</i>	9
1.2	Diagram złączy <i>Grove Base HAT</i>	10
1.3	Modulacja szerokości impulsów	12
1.4	Przykład wiadomości na linii <i>SDA</i> interfejsu <i>I<sup>2</sup>C</i>	13
1.5	Diagram czasowy <i>I<sup>2</sup>S</i>	13
1.6	Łącze bezpośrednie <i>SPI</i>	14
1.7	Łącze cykliczne <i>SPI</i>	15
1.8	Schemat wiadomości <i>UART</i>	16
1.9	Podstawowe wykorzystanie <i>GPIO</i> w języku <i>Python</i>	17
1.10	Tryb ciemny interfejsu graficznego <i>marimo</i>	19
2.1	Podpowiedzi w interfejsie graficznym <i>marimo</i>	23
2.2	Fragment instrukcji na temat <i>Sense HAT</i>	24
2.3	Uruchomiony program pierwszego laboratorium	25
2.4	Instalacja biblioteki <i>grove</i> z platformy <i>GitHub</i>	26
2.5	Fragment instrukcji II	27
2.6	Uruchomiony program drugiego laboratorium	27
2.7	Fragment instrukcji III	29
2.8	Uruchomiony program trzeciego laboratorium	29
2.9	Fragment instrukcji IV	32
2.10	Powiadomienie pogodowe na platformie <i>ntfy</i>	32
2.11	Uruchomiony program czwartego laboratorium	33
2.12	Fragment instrukcji V	34
2.13	Powiadomienia kontrolne na platformie <i>ntfy</i>	35
2.14	Uruchomiony program piątego laboratorium	35
3.1	Populacja treści pliku	36

---

# Spis tabel

---

1.1	Spis niewykorzystanych modułów . . . . .	8
1.2	Opis złączy <i>Grove Base HAT</i> . . . . .	11
1.3	Opis linii <i>SPI</i> . . . . .	14
1.4	Opis parametrów <i>UART</i> . . . . .	15
1.5	Cechy <i>marimo</i> wynikające z braków w innych rozwiązańach . . . . .	18
2.1	Układy scalone składające się na moduł <i>Enviro</i> . . . . .	30
2.2	Biblioteki języka <i>Python</i> wykorzystane w laboratorium IV . . . . .	31

---

## **Spis załączników**

---

F1	pi-labs.sh . . . . .	16
F2	dla-prowadzacego.pdf . . . . .	20
F3	instrukcje.pdf . . . . .	20