МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

С.Б. Могильний

ВБУДОВАНІ СИСТЕМИ ПРОГРАМНО-АПАРАТНИХ КОМПЛЕКСІВ ОБРОБКИ ІНФОРМАЦІЇ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра за освітніми програмами
«Радіотехнічні комп'ютеризовані системи», «Інформаційна та комунікаційна
радіоінженерія», «Інтелектуальні технології радіоелектронної техніки»
спеціальності 172 «Електронні комунікації та радіотехніка»*

Київ
КПІ ім. Ігоря Сікорського
2023

Рецензент — *Мосійчук Віталій Сергійович*, канд. техн. наук, доц. кафедри прикладної радіоелектроніки радіотехнічного факультету, Національний технічний університет КПІ ім. Ігоря Сікорського

Відповідальний редактор — *Жук Сергій Якович*, д-р техн. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 5 від 23.02.2023 р.) за поданням Вченої ради радіотехнічного факультету (протокол № 02/2023 від 10.02.2023 р.)

Електронне мережне навчальне видання

Автор: *Могильний Сергій Борисович*, канд. техн. наук, доц.

ВБУДОВАНІ СИСТЕМИ ПРОГРАМНО-АПАРАТНИХ КОМПЛЕКСІВ ОБРОБКИ ІНФОРМАЦІЇ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Вбудовані системи програмно-апаратних комплексів обробки інформації: Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 172 «Електронні комунікації та радіотехніка» / КПІ ім. Ігоря Сікорського; автор: С.Б.Могильний. – Електронні текстові дані (1 файл: 3,74 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2023. – 121 с.

В навчальному посібнику наводяться рекомендації до виконання лабораторних робіт з кредитного модуля «Вбудовані системи програмно-апаратних комплексів обробки інформації», який викладається студентам радіотехнічного факультету, що навчаються на спеціальності 172 «Електронні комунікації та радіотехніка».

Реєстр. № НП 22/23-474. Обсяг 4,8 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

Перелік умовних позначень та скорочень

1-Wire	One-Wire- Interface
ЄСКД	єдина система конструкторської документації
ІС	інтегральна схема
ЛР	лабораторна робота
ПОП	подійно орієнтоване програмування
ШІМ	широтно-імпульсна модуляція
DMA	direct memory access
I2C	Inter Integrated Circuit
IDE	Integrated Development Environment
NMEA	National Marine Electronics Association
PWM	програмована логічна інтегральна схема
RPi	Raspberry Pi
SPI	Serial Peripheral Interface Bus
UART	Universal Asynchronous Receiver/Transmitter
GPIO	General Purpose Input Output
GPS	Global Positioning System

ВСТУП

Даний навчальний посібник призначений для підготовки бакалаврів за освітніми програмами «Радіотехнічні комп'ютеризовані системи», «Інформаційна та комунікаційна радіоінженерія», «Інтелектуальні технології радіоелектронної техніки» спеціальності 172 «Електронні комунікації та радіотехніка» і може бути використаний для інших освітніх програм.

Підготовка спеціалістів за дисципліною «Вбудовані системи програмно-апаратних комплексів обробки інформації» передбачає 18 годин лекцій, 36 годин лабораторних занять, модульну контрольну роботу та домашню контрольну роботу, які орієнтовані на отримання знань і навичок роботи з вбудованими мікрокомп'ютерними системами, їх програмуванню на мові Python та обміну даними за різними протоколами з сенсорами та виконавчими механізмами.

Основна мета посібника – сформувати у студентів в процесі виконання лабораторних робіт навички реалізації мікрокомп'ютерних систем керування програмно-апаратними. Такі задачі вимагають знань системного проєктування, програмування, протоколів обміну даними, алгоритмів роботи різних виконавчих механізмів. Виконання лабораторних робіт дозволяє закріпити теоретичні знання, отримані на лекціях.

Навчальний посібник підготовлено відповідно до робочої навчальної програми (силабусу) дисципліни «Вбудовані системи програмно-апаратних комплексів обробки інформації». Він містить необхідний теоретичний матеріал, який дозволяє застосовувати набуті знання для реалізації взаємодії мікрокомп'ютера з різними складовими радіотехнічного комплексу для оброблення інформації.

Змістом даного видання є методичні рекомендації до виконання та опис лабораторних робіт кредитного модуля «Вбудовані системи програмно-апаратних комплексів обробки інформації». У вказівках до кожної лабораторної роботи наведені теоретичні питання в необхідному для виконання лабораторних робіт об'ємі.

Увага! Виконання кожної із лабораторних робіт, що наведені в даному посібнику, розраховане на 4 академічні години.

Виконання лабораторних робіт

Перед виконанням лабораторних робіт студенти повинні самостійно вивчити відповідні розділи дисципліни за рекомендованою літературою і конспектом лекцій, зміст лабораторної роботи, порядок її виконання. В режимі ofline перед початком виконання кожної роботи студенти проходять контроль знань, на якому повинні показати розуміння мети і змісту роботи. Студенти, які отримали незадовільну оцінку, до лабораторної роботи не допускаються. Приступати до виконання лабораторної роботи можна тільки з дозволу викладача. Результати виконання лабораторної роботи у вигляді працюючих схем, часових діаграм при симуляції реалізованого проєкта тощо надаються викладачу.

Зміст та оформлення звіту

У звіті необхідно відобразити:

- мету роботи;
- стислі теоретичні відомості;
- алгоритм та коди на мові сценаріїв Python;
- схеми підключення сенсорів та інших компонентів;
- результати у вигляді отриманих даних сенсорів тощо;
- аналіз отриманих результатів та висновки.

Звіти з лабораторних робіт виконуються кожним студентом індивідуально на окремих аркушах формату А4. На координатних осях графіків повинні бути позначенням масштабу та розмірності вимірюваних величин. У формулах та електричних схемах необхідно використовувати умовні позначення, які відповідають стандартам ЄСКД або використаної САПР. Особливу увагу необхідно приділити аналізу отриманих результатів та формулюванню висновків за результатами роботи.

Титульний лист звіту оформляється наступним чином:

Національній технічних університет України
Київський політехнічних інститут імені Ігоря Сікорського
Радіотехний факультет
Кафедра радіотехнічних систем

ЗВІТ

про лабораторну роботу ___
з дисципліни «Вбудовані системи програмно-апаратних комплексів обробки інформації»

_____

(назва роботи)

студента першого курсу, групи_____

_____

(прізвище, ім’я та по-батькові)

Викладач Могильний С.Б.:

Дата_____

202_ р.

Для забезпечення дистанційного (online) навчання використовуються платформи – Zoom та Moodle, які дозволяють проводити дистанційно лекційні та вступні заняття перед роботою (Zoom). Також при дистанційному навчанні студенти отримують цілодобовий доступ до мікрокомп’ютерів з Інтернету. Роз’яснення (консультації) зі складних та незрозумілих питань проводяться на онлайн зустрічах в Zoom.

Контроль виконання робіт та їх захист здійснюється на платформі дистанційного навчання Moodle. Протоколи виконання робіт завантажуються у

відповідний розділ дисципліни в Moodle. Треба звернути увагу при оформлені звітів робіт на дотримання вимог ДСТУ 3008:2015. Посилання на дистанційний курс - http://iot.kpi.ua/lms/course/view.php?id=5.

Зв'язок зі студентами підтримується через Телеграм–групу «Вбудовані системи» та електронну пошту старост груп, що забезпечує своєчасну видачу та контроль індивідуальних завдань у відповідності до графіку навчання.

Отримані бали відповідно до рейтингової системи оцінювання накопичуються в журналі оцінок Moodle та в розділі «Поточна інформація» Кампуса КПІ (https://ecampus.kpi.ua). В Кампусі також викладені всі навчальні матеріали курсу.

Лабораторна робота 1

ОРГАНІЗАЦІЯ ВІДДАЛЕНОГО ДОСТУПУ ТА ВИКОНАННЯ ОСНОВНИХ КОМАНД НА RASPBERRY PI

Мета роботи: Ознайомитися з функціональними можливостями мікрокомп’ютера Raspberry Pi (RPi) та основними командами для роботи з ним.

Зміст. В даній роботі вивчаються можливості віддаленого доступу до мікрокомп’ютера та вивчаються основні команди його операційної системи.

1.1. Теоретичні відомості

Для виконання подальших робіт з підключення сенсорів та виконавчих механізмів нам необхідно забезпечити доступ до RPi з локальної мережі (виконання ЛР в лабораторії) або з Інтернету (для дистанційного навчання). В другому випадку скористаємось сервером VNC (рис.1.1).

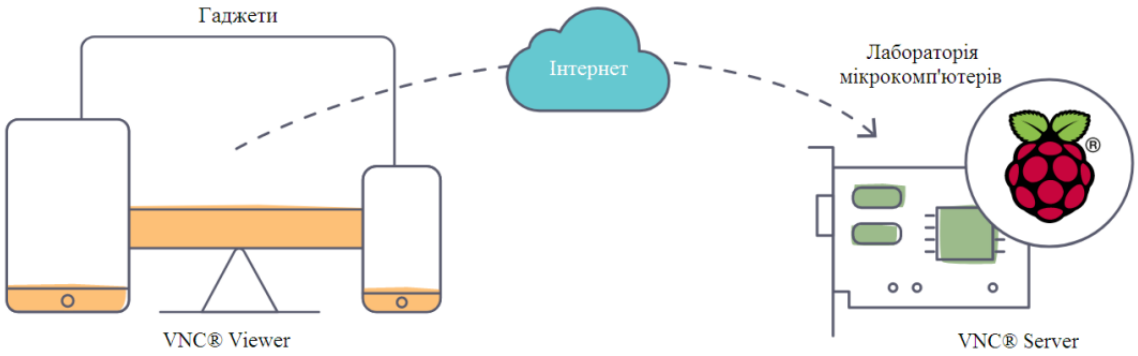


Рисунок 1.1 – Підключення до Raspberry Pi через Інтернет

Сервер VNC входить до складу операційної системи RPi і абсолютно безкоштовний для некомерційного використання; його просто потрібно включити.

Нам також знадобиться програма VNC Viewer для комп’ютера з ОС Windows, Mac, Linux, або мобільного пристрою iOS чи Android (рис.1.2), з якого ми будемо керувати RPi. На сайті realvnc.com можна завантажити VNC Viewer.

Рисунок 1.2 – VNC Viewer на гаджеті

Спочатку виконаємо команди, щоб переконатися, що у нас остання версія програмного забезпечення: `sudo apt update` і `sudo apt upgrade -y`

Якщо використовуємо стару версію VNC Server, перезапустимо його. Якщо ні, і вже завантажилися з використанням графічного робочого столу, виберемо **Menu > Preferences > Raspberry Pi Configuration > Interfaces** та переконаємося, що для VNC встановлено значення **Enabled**.

Або ж запустимо команду `sudo raspi-config`, перейдемо до **Interfacing Options > VNC** і виберемо **Yes**.

Відтепер сервер VNC буде запускатися автоматично при кожному завантаженні RPi.

Є два способи підключення і можемо використовувати один з них або обидва:

- Встановлення прямого зв'язку (підключаємо монітор, клавіатуру і мишу).
- Підключення через локальну мережу або Інтернет, яке розглядаємо в даній роботі.

Переконаймося, що ми завантажили додаток VNC Viewer на комп’ютери чи пристрої, з яких хочемо керувати мікрокомп’ютером.

Підключення через локальну мережу є швидким і простим, за умови, що ми приєднані до тієї самої приватної локальної мережі, що і наш RPi (наприклад, Ethernet чи Wi-Fi мережа вдома, в школі або в офісі) (рис.1.3).

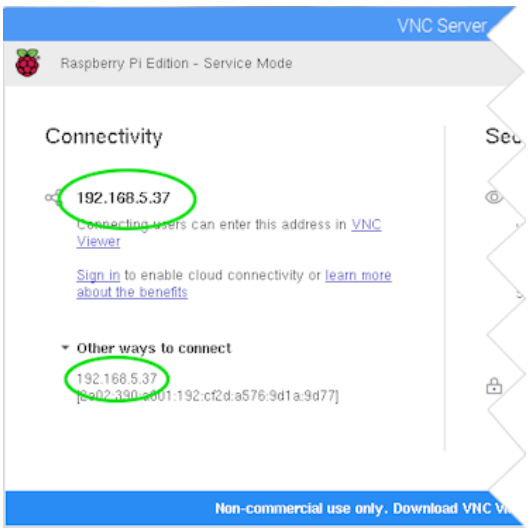


Рис.1.3 – Підключення в локальній мережі

1.2. Налаштування підключення до Raspberry Pi через хмару

Хмарні підключення зручні та зашифровані наскрізно, тому настійно рекомендуються для з'єднань через Інтернет. Немає потреби робити переконфігурацію брандмауера або маршрутизатора і не потрібно знати IP-адресу свого RPi чи надати йому статичну IP-адресу.

Нам знадобиться обліковий запис RealVNC – це абсолютно безкоштовно для налаштування. Нам надається спеціальна версія передплати на домашню сторінку, яка забезпечує як хмарне, так і пряме підключення, а також такі функції, як аутентифікація системи, передача файлів, друк та чат. Для цього необхідно:

1. Створити обліковий запис RealVNC, ввівши адресу своєї електронної пошти у відповідному полі на сторінці сайта, дотримуючись інструкцій.

2. На своєму RPi вибрати **Licensing** в меню стану VNC Server, далі вибрати **Sign in to your RealVNC account** і ввести електронну адресу та пароль нового облікового запису (рис.1.4).

Важливо! *Підключитися до облікового запису на сайті розробника з використанням віртуального робочого столу не можна. Необхідно підключити до RPi монітор, клавіатуру і мишу.*

3. На пристрої, з якого будемо керувати, запустимо VNC Viewer і ввійдемо, використовуючи ті самі дані облікового запису.

4. Підключення до RPi автоматично з’являється у програмі VNC Viewer під іменем вашого комп’ютера, наприклад, rpi5 (рис1.5). Просто натисніть або двічі клацніть для підключення:

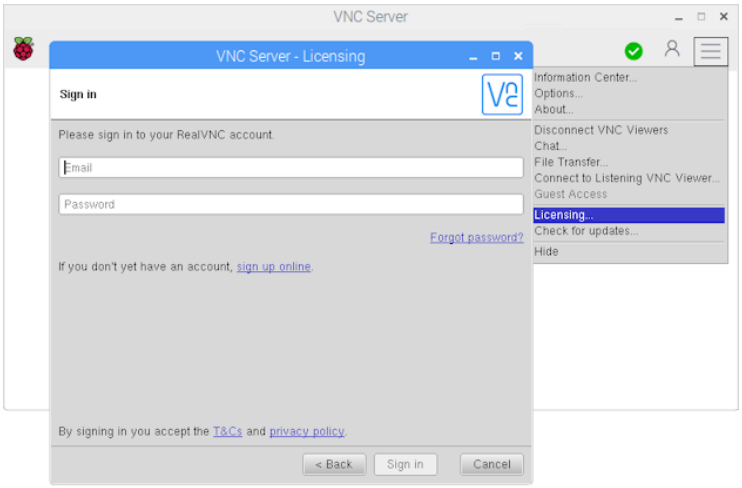


Рисунок 1.4 – Ліцензування Raspberry Pi на хмарному сервісі

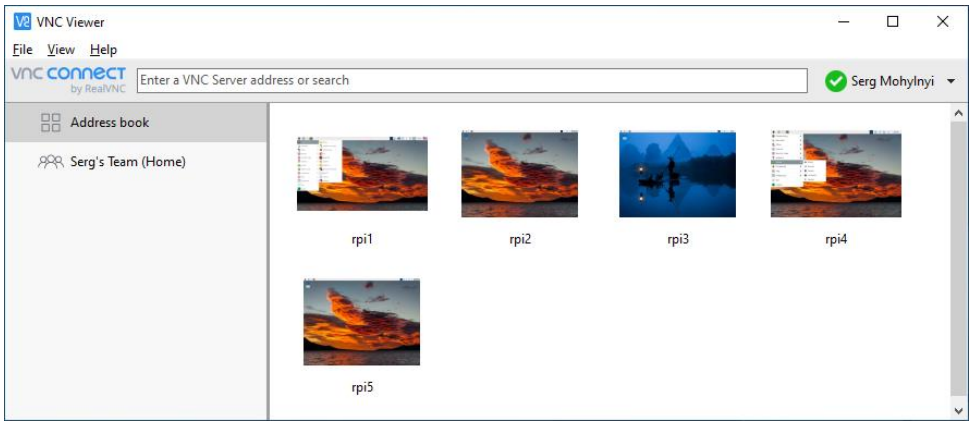


Рисунок 1.5 – Вибір підключення до Raspberry Pi

Аутентифікація на сервері VNC

Щоб встановити пряме або хмарне з’єднання, потрібно пройти аутентифікацію на сервері VNC. Вводимо ім’я користувача та пароль, які ми зазвичай використовуємо для *log on*, у свій обліковий запис користувача на RPi.

За замовчуванням ці облікові дані – `pi` та `raspberry`, але, сподіваємось, викладач вже змінив пароль на щось більш безпечне!

Сервер VNC може віддалено показувати екран програм RPi, які використовують прямо нанесене накладання, наприклад, текстову консоль, модуль камери RPi тощо.

Щоб увімкнути цю функцію, відкриємо діалогове вікно VNC Server, перейдемо до **Menu > Options > Troubleshooting**, і виберемо **Enable direct capture mode**. На пристрої, який будемо використовувати для керування, запустимо VNC Viewer і підключимося (якщо вже підключені, то потрібно буде знову підключитися).

Якщо увімкнули режим прямої зйомки і рухи миші здаються нестабільними при віддаленому використанні, спробуємо натиснути F8, щоб відкрити контекстне меню VNC Viewer і вибрати **Relative Pointer Motion**.

Якщо продуктивність, здається, погіршується, спробуємо:

1. Запустимо на RPi `sudo raspi-config`, перейдемо до **Advanced options > Memory Split** та переконаємося, що графічний процесор має принаймні 128 МБ.

2. Зменшимо роздільну здатність екрану RPi.

Передача файлів на Raspberry Pi та з нього

Ми можемо передавати файли на RPi та з нього за умови, що ми підключаємося з VNC Viewer, який працює на настільному комп’ютері Windows, Mac або Linux.

• Щоб перенести файли на RPi, натискаємо кнопку панелі інструментів  VNC Viewer і дотримуємось інструкцій. Детальні кроки тут.

• Щоб передати файли *з* RPi, використовуючи VNC Viewer, віддалено відкриємо діалогове вікно VNC Server, виберемо **Menu > File transfer** і дотримуємось інструкцій. Детальні кроки тут.

Керування сервером VNC у командному рядку

Також можемо працювати з сервером VNC виключно в командному рядку або через SSH, якщо хочете.

Корисні команди для ОС RPi з використанням systemd:

• щоб запустити VNC Server зараз:

```
sudo systemctl start vncserver-x11-serviced.service
```

• щоб запустити сервер VNC під час наступного завантаження та кожного наступного завантаження:

```
sudo systemctl enable vncserver-x11-serviced.service
```

• щоб зупинити сервер VNC:

```
sudo systemctl stop vncserver-x11-serviced.service
```

• щоб запобігти запуску VNC-сервера під час завантаження:

```
sudo systemctl disable vncserver-x11-serviced.service
```

Вирішення проблем з роздільною здатністю екрану RPi

Це нам може знадобитися, якщо:

• погіршується працездатність – менша роздільна здатність екрану забезпечує більш швидку реакцію;

• наш RPi «безголовий» (тобто не підключений до монітора), а початкова роздільна здатність екрана за замовчуванням занадто мала.

Щоб змінити роздільну здатність, запустимо команду `sudo raspi-config`, перейдіть до **Advanced Options > Resolution**.

Якщо це меню недоступне або хочемо отримати більше контролю, вкажемо налаштування (табл.1.1) у файлі `/boot/config.txt`:

Таблиця 1.1 – Параметри для налаштування HDMI

Параметр	Значення	Пояснення
`hdmi_force_hotplug`	`1`	Вказує RPi, що підключений дисплей HDMI.
`hdmi_ignore_edid`	`0xa5000080`	Ігнорувати EDID/відображення даних.
`hdmi_group`	`2`	Визначає вихідну групу HDMI.
`hdmi_mode`	`16`	Режим (приклад) 1024x768 при 60Гц.

Звертаємо увагу, що параметри, зазначені в цьому файлі, змінюють параметри для моніторів, які згодом підключаємо (якщо не скасуємо `hdmi_force_hotplug`), тому вибираємо роздільну здатність „без голови”, сумісну з нашим звичайним монітором.

1.3. Найбільш корисні команди Raspberry Pi

Взагалі важко відстежити всі корисні команди RPi, які ми будемо часто використовувати, тому розглянемо список деяких з найбільш поширених і важливих, що зробить роботу в Linux на RPi набагато простішою.

Примітка: існує два режими (modes), в яких можемо працювати в Linux. Одним з них є режим з основними правами доступу, а інший режим з правами доступу адміністратора (типу супер користувача, або root). Деякі завдання не можуть бути виконані з основними правами і вам треба буде перейти в root-режим для їх виконання. Часто бачитимемо префікс `sudo` перед командами, що означає, що ми пропонуємо комп'ютеру виконати дану команду з супер повноваженнями. Іншим варіантом є доступ до командного рядка root, який запускає всі команди з супер повноваженням. Можемо перейти в режим `root`, ввівши `sudo su` у командному рядку. Після введення `sudo su`, ми побачимо у командному рядку `root@raspberrypi:/home/pi#`, і всі наступні команди можуть бути введені без префікса `sudo`, але матимуть повноваження суперкористувача.

Більшість команд нижче, мають багато інших корисних опцій, які в цьому занятті не пояснюються. Щоб побачити список всіх інших доступних варіантів команди, введемо команду, а потім `-help`.

Загальні команди

`sudo apt update` – оновлює версію Raspbian.

`sudo apt upgrade` – оновлює всі пакети програмного забезпечення, які ми встановили.

`apt` – відносно новий менеджер пакетів високого рівня для систем Debian/Ubuntu. Приклад використання: `sudo apt update`

`clear` – очищає екран терміналу раніше запуску команд і тексту.

`date` – виведення поточної дати.

`find / -name example.txt` – пошук у всій системі файлу `example.txt` і виведення списку всіх каталогів, які містять файл.

`history` – дозволяє побачити раніше використані команди або отримати інформацію про команду, яка виконується користувачем. Досвідчені користувачі знають, що історія запущених команд зберігається в файлі `~/.bash_history`. Щоб команда не записувалась в історію, наберемо перед командою пробіл: `[space] [command]`

`killall` – завершує процес з використанням його імені, наприклад, `killall firefox`

`man` – показує сторінку керівництва для команди чи файлу (від слова manual). Щоб дізнатися більше запустимо `man man`, щоб переглянути сторінку керівництва команди `man`. Наприклад, щоб прочитати ASCII-таблицю, набираємо: `man ascii`

`nano example.txt` – відкриває файл `example.txt` в «Nano» - текстовому редакторі Linux. Детальніше, як працювати в редакторі nano.

`pipes` – дозволяє виведення однієї команди використати як вхідні дані для іншої команди. Символом `pipes` є вертикальна лінія `|`. Наприклад, щоб

показувати тільки перші 10 записів з командою `ls`, це можна зробити через команду `head` таким чином (`head` за замовчуванням показує лише 10 записів): `ls | head`

`pkill` – завершує запущений процес. Ця команда особливо корисна, коли додаток не відповідає: `pkill [application_name]`

`poweroff` – негайне вимкнення.

`pwd` – відображає ім'я поточного робочого каталогу, тобто, `pwd` виведе щось на зразок `/home/pi`.

`raspi-config` – відкриває меню налаштування конфігурації.

`reboot` – негайне перезавантаження.

`sudo` – дає право дозволеному користувачу системи запустити команду як root користувач. Якщо набрали команду без `sudo`, а потім виявилось, що вона необхідна, набираємо: `sudo !!`

`shutdown -h now` – негайне вимикання.

`shutdown -h 01:22` – зупинка о 1:22.

`startx` – відкриває GUI (графічний інтерфейс користувача).

Команди для файлів/каталогів

`adduser` і `addgroup` – використовуються для додавання користувача і групи в систему у відповідності з конфігурацією за замовчуванням, указаній в файлі `/etc/adduser.conf`, наприклад, `sudo adduser pi`

`cat example.txt` – відображає вміст файлу `example.txt`.

`cd /abc/xyz` – змінює поточний каталог на каталог `/abc/xyz`. Якщо випадково змінили директорію, то повернутися можна командою: `cd -`

`chmod` – зазвичай використовується для зміни прав доступу до файла. Команда `chmod` може використовувати символи `u` (користувач, який є власником файлу), `g` (група файлів), `o` (інші користувачі) і дозволи `r` (читання), `w` (запис) і `x` (виконання), + додає права, - забирає права. Наприклад, `chmod u +x *filename*` надає дозвіл на виконання власнику

файла. Можливе використання восьмеричного формату, наприклад, якщо дозволити власнику читати-писати-виконувати (rwx або 111 – число 7 в десятковому форматі), групі – лише читати та писати (rw- або 110 – число 6 в десятковому форматі), а іншим – лише читати (r-- або 100 – число 4 в десятковому форматі), то команда для дозволів file1 буде виглядати так: chmod 764 file1.

chown – змінює користувача та/або групу, якій належить файл. Як правило, вона повинна бути запущена як root, використовуючи sudo, наприклад, sudo chown pi:root *filename* змінює власника на pi і групу на root.

cp XXX – копіює файл або каталог XXX, і вставляє його у вказане місце; тобто, cp examplefile.txt /home/pi/office/ копіює examplefile.txt в поточному каталозі і вставляє його в каталог /home/pi/office/. Якщо файл не знаходиться в поточному каталозі, додайте шлях розташування файлу (тобто, cp /home/pi/documents/examplefile.txt /home/pi/office/ копіює файл з каталогу documents в каталог office).

dd – копіює файл з заданим перетворенням файлу. Часто використовується, щоб копіювати весь диск в один файл або навпаки, наприклад, dd if=/dev/sdd of=backup.img створить резервну копію з SD-карти або USB-диска в /dev/sdd. Переконаємося, що використовується правильний диск при копіюванні образу на карту пам'яті, бо ця команда може перезаписати весь диск.

head filename – відображає початок файлу і може бути використано з -n, щоб вказати кількість рядків, які показати (за замовчуванням 10), або з -c, щоб задати кількість байт.

ls -l – отримуємо список файлів в поточному каталозі, разом з розміром файлу, датою зміни і дозволами. Використання прапора -a замість -l дає змогу переглядати файли, які починаються з . (крапки, тобто, точкові файли).

`mkdir example_directory` – створює новий каталог з ім'ям `example_directory` всередині поточного каталогу.

`mv XXX` – переміщає файл або каталог з ім'ям `XXX` у вказане місце. Наприклад, `mv examplefile.txt /home/pi/office/` переміщує `examplefile.txt` з поточного каталога в каталог `/home/pi/office/`. Якщо файл не знаходиться в поточному каталозі, додаємо шлях розташування файлу (тобто, `mv /home/pi/documents/examplefile.txt /home/pi/office/` переміщує файл з каталогу `documents` в каталог `office`). Ця команда також може бути використана для перейменування файлів (але тільки в тій же директорії). Наприклад, `mv examplefile.txt newfile.txt` перейменовує `examplefile.txt` в `newfile.txt`, і зберігає його в тій же директорії.

`rm example.txt` – видаляє файл `example.txt`.

`rmdir example_directory` – видаляє каталог `example_directory` (тільки якщо він порожній).

`scp user@10.0.0.32:/some/path/file.txt` – копіює файл через SSH. Може бути використано для завантаження файлу з десктопа/ноутбука на RPi. `user@10.0.0.32` – це ім'я користувача та IP-адреса дектопа/ноутбука, а `/some/path/file.txt` – шлях та ім'я файлу файла на десктопі/ноутбуці.

`tar` – створення архіву або витягувати файлів з архівного файлу. Може також зменшити необхідний простір, шляхом стиснення файлу, подібно до архіву. Щоб створити стиснений файл використайте `tar -cvzf *filename.tar.gz* *directory/*` Щоб витягнути вміст файлу з архіву скористаємося `tar -xvzf *filename.tar.gz*`

`touch` – створює новий порожній файл в поточному каталозі.

`tree` – показати каталог і всі підкаталоги та файли з відступом у вигляді деревовидної структури.

`users` – виводить імена всіх користувачів, які увійшли в систему.

Команди для мережі/Інтернету

`ifconfig` — використовується для перевірки стану мережевого з'єднання (щоб побачити, наприклад, чи отримав `wlan0` IP-адресу).

`ip` – утиліта командного рядка із пакета iproute2. Дозволяє виконати налаштування мережевої підсистеми і замінює такі утиліти, як `ifconfig, route, arp`. Наприклад, назначити IP-адресу інтерфейсу `eth1` можна так:
`sudo ip addr add 192.168.56.10 dev eth1`

`iwconfig` — для перевірки, які мережі використовує бездротовий адаптер.

`iwlist wlan0 scan` – виведення списку доступних в даний час бездротових мереж.

`iwlist wlan0 scan | grep ESSID` — використовується `grep` разом з ім'ям поля для додавання в список тільки тих полів, які потрібні (наприклад, просто перерахувати ESSID).

`ping` – запуск тестування між двома пристроями, підключеними до мережі. Наприклад, `ping 10.0.0.32` надішле пакет до пристрою з IP 10.0.0.32 і буде чекати відповіді. Працює також з адресами веб-сайтів.

`wget http://www.website.com/example.txt` – завантажує файл `example.txt` з Інтернету і зберігає його в поточній директорії.

Команди для отримання інформації про систему

`cat /proc/meminfo` — показує детальну інформацію про пам'ять.

`cat /proc/partitions` — показує розмір і кількість розділів на карті SD або жорсткому диску.

`cat /proc/version` – показує, яку версію RPi використовуєте.

`df -h` — відображення інформації про доступний дисковий простір.

`df /` — показує, скільки вільного місця на диску.

`dpkg --get-selections | grep XXX` — показує всі встановлені пакети, які пов'язані з `XXX`.

`dpkg --get-selections` – показує всі встановлені пакети.

`free` – показує, скільки вільної пам'яті.

`hostname -I` – показує IP-адресу RPi.

`lshw` – найпростіший інструмент для отримання детальної інформації про апаратну конфігурацію машини. Для отримання найбільш повної інформації викликайте її з правами суперкористувача.

`lsusb` – виведення списку апаратних USB-підключень до RPi.

`UP key` – натисканням клавіші UP буде введена остання команда, що була записана в командному рядку. Це швидкий спосіб виправити команди, які були записані помилково.

`vcgencmd measure_temp` – показує температуру процесора.

`vcgencmd get_mem arm && vcgencmd get_mem gpu` – показує розподіл пам'яті між процесором і GPU.

Детальніше як шукати інформацію про систему Raspberry Pi

1.4. Завдання

1. Якщо викладач не створив сам, то необхідно створити 5 облікових записів на https://manage.realvnc.com/en/. Рекомендується використати реальні поштові скриньки gmail.com, бо сайт realvnc.com здійснює верифікацію.
2. Надіслати e-mail та паролі реєстрації облікових записів викладачу, щоб він зміг підключити 5 комп'ютерів RPi до хмари (1 бригада – перший обліковий запис, 2-га – другий...) з використанням монітора, клавіатури та миші.
3. При підключення до доступного RPi через хмарний сервіс треба буде ввести логін і пароль, які задав той, хто підключав RPi до хмари.
4. Завантажте VNC Viewer для свого гаджета.
5. Підключіться до відповідного RPi з використанням VNC Viewer та вибраних логіна і пароля. Оскільки планується не вимикати RPi, то кожен студентами зможе зручно попрацювати. Зверніть увагу, що при першому вході на обліковий запис (з нової IP-адреси) сайт може вимагати верифікацію, надіславши листа на поштову скриньку, яка використана як логін. Якщо для

кількох студентів (на бригаду) зареєстрований один обліковий запис, то зверніться до його власника, щоб він підтвердив Ваші права доступу.

6. Виконати в терміналі команди `update` і `upgrade`.

7. Знайдіть локальну IP-адресу RPi та температуру процесора.

8. Знайдіть файли, які згадувались раніше при налаштуванні підключення RPi до мережі. Якщо RPi підключався до WiFi, то спробуйте знайти пароль для підключення. Зробити скріншоти для протоколу.

9. Створіть текстовий файл на RPi і завантажте його на свій гаджет.

10. Вивчіть інтерфейс мікрокомп'ютера. Основну увагу зверніть на меню Preferences.

11. Повторіть наведені в ЛР команди для роботи з RPi, зробіть скріншоти для протоколу, незалежно, чи успішно виконана дана команда.

12. Оформіть протокол і збережіть його на платформі дистанційного навчання.

13. Для завершення роботи з RPi просто закрийте вікно віддаленого доступу до робочого столу.

1.5. Зміст звіту

1. Мета роботи.
2. Скріншоти результатів виконання завдання.
3. Відповіді на контрольні питання до роботи.
4. Короткі висновки.

1.6. Контрольні питання

1. Як змінити власника певного файла?
2. Як обмежити певного користувача лише переглядом файлів?
3. Які є команди для оновлення операційної системи RPi?

Лабораторна робота 2

ВИКОРИСТАННЯ GPIO RASPBERRY PI

Мета роботи: Вивчення можливостей інтерфейсу GPIO, безпечного використання контактів інтерфейсу при підключенні зовнішніх елементів.

Зміст. RPi має контакти GPIO, до яких можна підключити різноманітні пристрої, що особливо важливо при використанні RPi для Інтернету речей та вбудованих систем. В даній роботі вивчаються, як скористатися GPIO для організації управління у вбудованих системах та обміну даними.

2.1. Теоретичні відомості

RPi надає підтримку шин UART, SPI, I2C і 1-wire (рис.2.1).

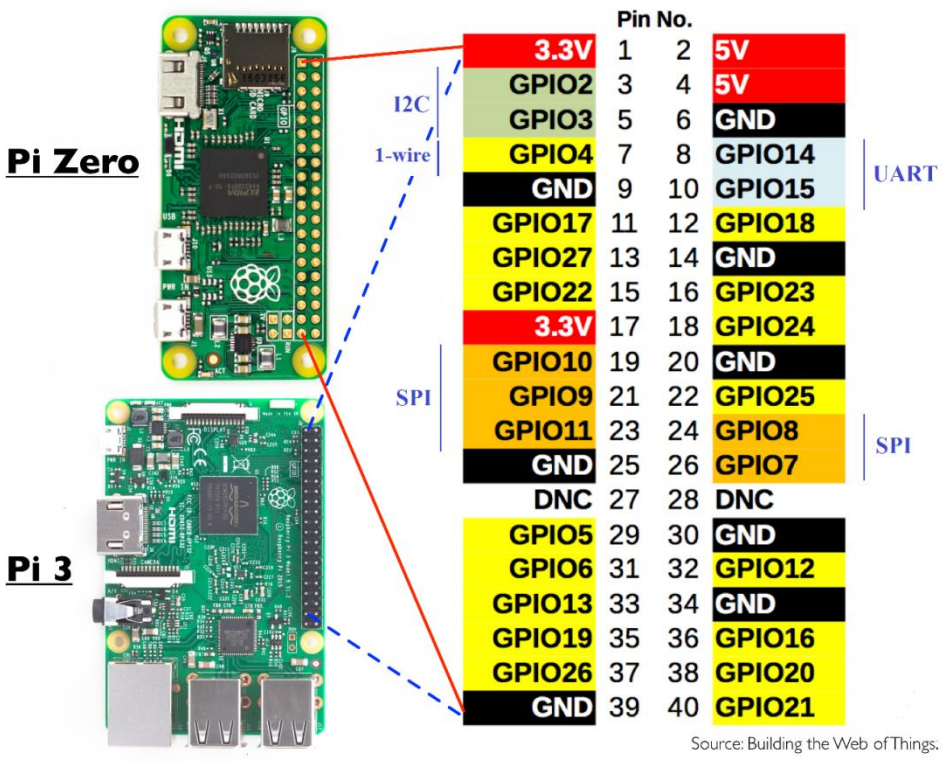


Рисунок 2.1 – Виводи GPIO

Короткі пояснення наведених на рис.2.1 позначень:

• GPIO (General Purpose Input Output) - стандартні виводи, які можуть бути використані для керування, наприклад, такими пристроями (вмикання і вимикання), як світлодіоди.

• I2C (Inter-Integrated Circuit) – виводи, які дозволяють підключати і спілкуватися з апаратними модулями, що підтримують даний протокол (I2C-протокол), який використовує, зазвичай, лише 2 виводи.

• SPI (Serial Peripheral Interface Bus) – виводи, які можуть бути використані для підключення і спілкування з пристроями SPI. Майже те ж, що і I2C, але використовується інший протокол.

• UART (Universal Asynchronous Receiver/Transmitter) – універсальний асинхронний приймач/передавач (послідовний порт), що використовуються для зв'язку з іншими пристроями.

• 1-wire (One-Wire- Interface) – послідовний протокол, який використовує єдину лінію передачі даних та землю для організації зв'язку, наприклад, з сенсорами.

Завдяки GPIO стало можливим легко вводити дані з сенсорів і формувати сигнали керування виконавчими механізмами.

Для того, щоб скористатися підтримкою різних протоколів в RPi, необхідно їх дозволити через команду `raspi-config` з терміналу (рис.2.2 і 2.3):

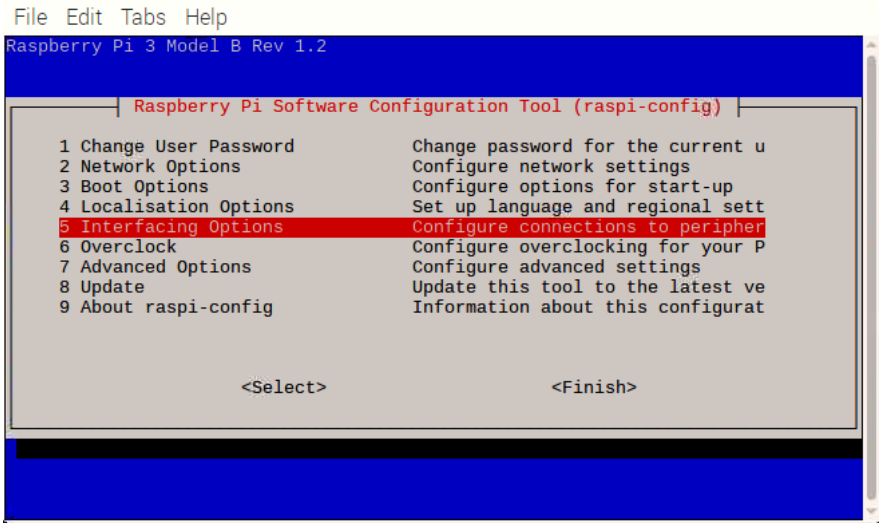


Рисунок 2.2 – Вибір закладки інтерфейсів

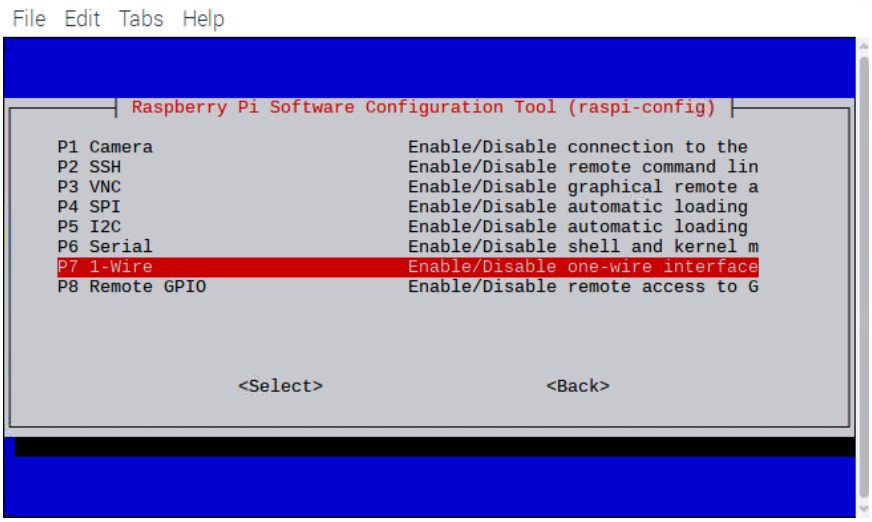


Рисунок 2.3 – Дозвіл роботи відповідних протоколів

Це можна зробити і через меню Preferences -> Raspberry Pi Configuration (рис.2.4)

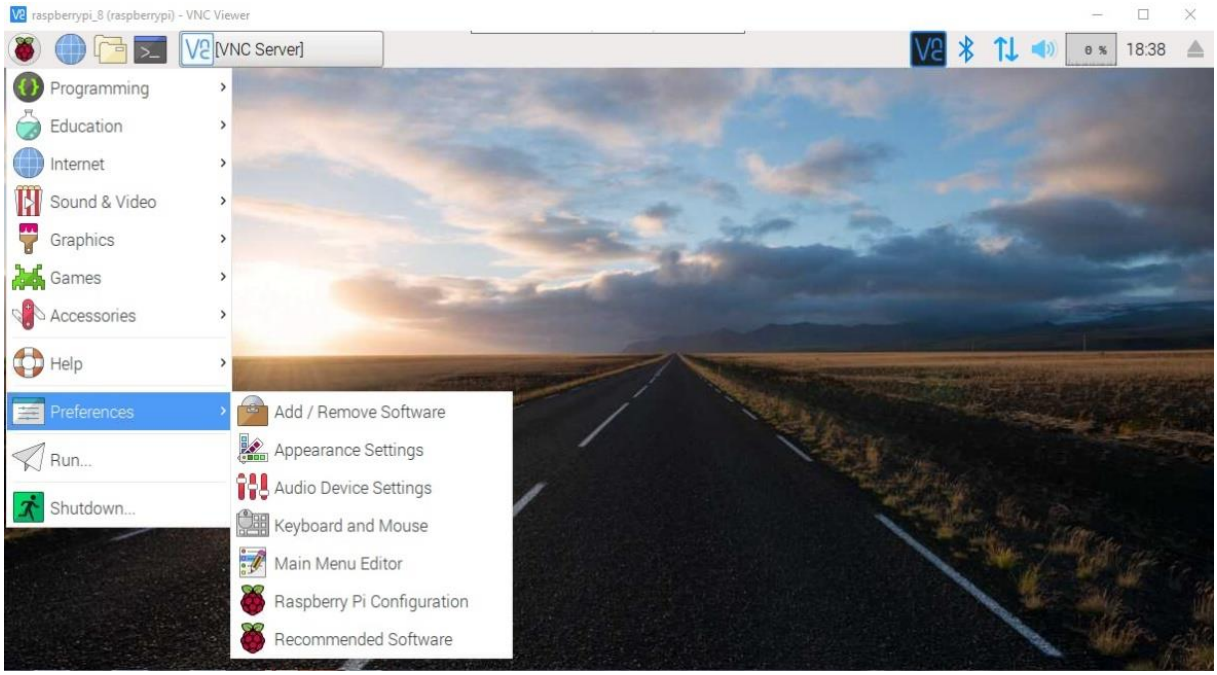


Рисунок 2.4 – Вибір меню Raspberry Pi Configuration

А далі переходимо в закладку Intefaces та дозволяємо необхідні інтерфейси (протоколи) (рис.2.5):

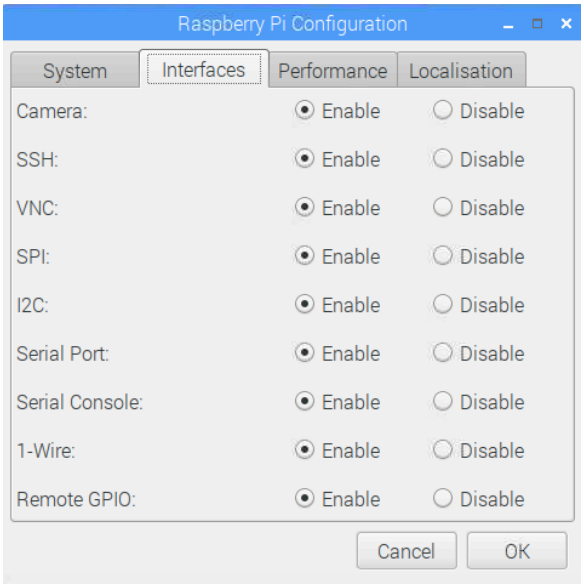


Рис.2.5 – Дозвіл використання інтерфейсів

Встановлення бібліотеки для роботи з GPIO

Заходимо в консоль RPi і завантажуємо архів:

```
wget www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz
```

Примітка: Останню версію бібліотеки можна завантажити на цьому сайті.

Розархівуємо:

```
tar zxvf bcm2835-1.71.tar.gz
```

Переходимо в директорію, куди розгорнулась бібліотека:

```
cd bcm2835-1.71
```

Встановлюємо бібліотеку:

```
sudo ./configure make
sudo make check
sudo make install
```

Після цього бібліотека для роботи з GPIO буде встановлена, а модуль `snd-bcm2835` доданий в файл `/etc/modules`, але ще не завантажений в пам’ять. Для завантаження в пам’ять скористаємося командою:

```
sudo modprobe snd-bcm2835
```

бо ми не можемо просто перезавантажитися для застосування налаштування при доступі до RPi з Інтернету.

На цьому етапі з виводами GPIO вже можна виконувати якісь прості дії, наприклад, читати стан порту або вмикати/вимикати підключене навантаження.

Наступне важливе значення має нумерація виводів GPIO, бо для різних додатків вона може бути різною. На рис.2.6 показані номери контактів роз'єму RPi, імена element14 (Name), номери wiringPi та номери Python (BCM).

Python (BCM)	WiringPi GPIO	Name	P1 Pin Number		Name	WiringPi GPIO	Python (BCM)
		3.3v DC Power	1	2	5v DC Power		
	8	GPIO02 (SDA1, I2C)	3	4	5v DC Power		
	9	GPIO03 (SCL1, I2C)	5	6	Ground		
4	7	GPIO04 (GPIO_GCLK)	7	8	GPIO14 (TXD0)	15	
		Ground	9	10	GPIO15 (RXD0)	16	
17	0	GPIO17 (GPIO_GEN0)	11	12	GPIO18 (GPIO_GEN1)	1	18
27	2	GPIO27 (GPIO_GEN2)	13	14	Ground		
22	3	GPIO22 (GPIO_GEN3)	15	16	GPIO23 (GPIO_GEN4)	4	23
		3.3v DC Power	17	18	GPIO24 (GPIO_GEN5)	5	24
	12	GPIO10 (SPI_MOSI)	19	20	Ground		
	13	GPIO09 (SPI_MISO)	21	22	GPIO25 (GPIO_GEN6)	6	25
	(no worky 14)	GPIO11 (SPI_CLK)	23	24	GPIO08 (SPI_CE0_N)	10	
		Ground	25	26	GPIO07 (SPI_CE1_N)	11	
	30	ID_SD (I2C ID EEPROM)	27	28	ID_SC (I2C ID EEPROM)	31	
5	21	GPIO05	29	30	Ground		
6	22	GPIO06	31	32	GPIO12	26	12
13	23	GPIO13	33	34	Ground		
19	24	GPIO19	35	36	GPIO16	27	16
26	25	GPIO26	37	38	GPIO20	28	20
		Ground	39	40	GPIO21	29	21

Рисунок 2.6 – Відповідність номерів контактів GPIO нумерації в додатках

Використання виводів GPIO і заходи безпеки

Перш, ніж щось під’єднати до виводів інтерфейсу GPIO, давайте розберемося які напруги/струми можна підключати до виводів в режимі входу і які ми отримаємо на виході.

Вивід GPIO в режимі входу:

• напруга 3,3В;

• рівень логічної одиниці (1) отримуємо, починаючи приблизно з напруги 1,8В;

• максимальний струм 0,5мА.

Вивід GPIO в режимі виходу:

• Напруга 0 і 3,3В (логічна одиниця – не 5В, як звикли вважати);

• Струм 2-16мА (за замовчуванням 8мА);

• Максимальний сумарний струм навантаження для всіх задіяних виводів – 50мА.

Також важливо пам'ятати що:

• Внутрішні буфери і ключі, які підключені до виводів GPIO, не захищені від перевантаження напруги і струму;

• Виводи живлення з вихідною напругою 3,3В можна навантажувати струмом не більше 50мА (рис.2.7);

• Виводи з вихідною напругою 5В намагаємося вантажити не більше, ніж на 500мА.

$$max\ I_{OUT} = 16\ mA$$

$$max \sum I_{OUT} = 50\ mA$$

$$max \sum I_{5V\text{-}OUT} = 500\ mA$$

Рисунок 2.7 – Максимальні значення струму через виводи GPIO

При проєктуванні пристроїв, в яких використовується велика кількість виводів GPIO, треба обов'язково робити розв'язку через додаткові буферні схеми, перетворювачі рівня напруги, електронні ключі. Це потрібно для того, щоб убезпечити RPi від можливого пошкодження (вигоряння) електронних комутаторів (ключів) і буферів всередині мікросхеми.

А тепер давайте подивимося, що вийде, якщо ми підключимо до трьох портів GPIO і загального для них виводу 3,3В три світлодіода, через кожен з яких буде протікати струм 20мА (над'яскраві світлодіоди): 20мА * 3 = 60мА, що є перевищенням максимально допустимого струму на виводі 3,3В і перевантаженням для виводів GPIO (по 20мА на вивід).

В даному випадку, для живлення таких світлодіодів треба підключити до виводів GPIO додаткові ключі на транзисторах або мікросхемах, які будуть керувати світлодіодами і не будуть перевантажувати виводи інтерфейсу.

Якщо до кожного з кількох виводів GPIO приєднаємо світлодіод, то їх спільну точку можна підключити до землі. В такому випадку, світлодіод буде світитися, якщо на виводі буде присутній високий (3,3В) рівень, а для обмеження струму до 15мА послідовно світлодіоду підключимо обмежувальний резистор.

Виводи з напругою 5В підключені до основної шини живлення 5В плати (після стабілізатора) і велике навантаження може вплинути на стабільність роботи платформи в цілому, а також пошкодити внутрішній стабілізатор напруги +5В. Потрібно з обережністю ставитися до використання виводу 5В: допустимий струм не повинен перевищувати 1А (без врахування споживання самої плати).

Насправді, споживаний RPi струм багато в чому залежить від завантаженості процесора і відеопроцесора (графічна оболонка вимагає набагато більше ресурсів, ніж консольний режим), а також від кількості підключених пристроїв до USB портів та їх споживання. Значення загального споживаного струму може змінюватися в межах від 250мА до 1А.

RPi, як і будь-яка інша чутлива електроніка, може бути пошкоджений статичною електрикою. Перед роботою бажано прибрати накопичений на тілі статичний заряд, зняти з себе вовняний одяг і не торкатися елементів на платі.

Перед паянням модулів і плат бажано відключити їх від роз'єму GPIO і буде непогано, якщо є можливість виконувати монтаж паяльником з напругою 36В (або нижчою) з використанням понижуючого трансформатора з 220В до 36В тощо.

Підіб'ємо невеликий підсумок із заходами безпеки:

• Перед роботою зняти з себе статичний заряд.

• Бажано виконувати пайку електроніки з відключеним роз'ємом GPIO.

• Не перевантажувати виводи живлення 3,3 (50мА) та 5В (300-500мА) і не під’єднувати їх безпосередньо до інших виводів.

• Унеможливити протікання струму величиною більш 8-15мА через виводи GPIO.

• Унеможливити підключення напруги більше 3,3В на виводи GPIO.

• Підключати безпосередньо до RPi не більше 3-х світлодіодів (на 2-15мА кожен).

• Уважно перевіряти правильність підключення перед подачею живлення на схему і RPi.

Рівні напруги на виводах GPIO за замовчуванням

Після того як RPi включений і завантажена операційна система, на виводах GPIO встановлений низький рівень (0В), і так буде доти, поки якась програма або сценарій не змінить стан портів.

Ця інформація взята з документації, вона також підтверджується вимірюванням станів виводів за допомогою тестера. Тут все логічно і зручно: після вмикання всі виводи виставлені в "0", пізніше виставляємо та перемикаємо стан необхідних виводів і, таким чином, будуємо свої апаратно-програмні додатки на основі RPi.

Але, починаючи з моменту подачі живлення на платформу і до моменту ініціалізації драйверів в ОС, на довільних виводах короткочасно можуть бути присутні високі рівні (3,3В).

Цей момент потрібно передбачити при проєктуванні схем, що підключаються до GPIO.

Іноді непоганим рішенням може бути резервування одного виводу GPIO для управління схемою подачі живлення на електроніку і виконавчі пристрої, в такому випадку живлення на електроніку надійде тільки після того, як

завантажиться ОС і сценарій нашої програми подав високий рівень на вивід управління живленням схеми.

Більш детальну інформації про те, як працює GPIO на платформі RPi, можна отримати з документації про процесори Broadcom.

Приклад програми для використання GPIO:

```
#!/bin/sh
# Номера GPIO повинні вибиратися з цього списку:
# 0, 1, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25
# Зверніть увагу, що номери GPIO, які ви тут вказуєте, відносяться
до контактів
# BCM2835, а не до номерів виводів роз’єма.
# Тому, якщо хочете активувати контакт 7 на роз’ємі, то повинні
# використати GPIO4 в цьому сценарію. Аналогічно, для активації
контакту 11 на роз’ємі повинні використати GPIO17.
# Налаштувати GPIO4 і встановити його як вихідний
echo "4" > /sys/class/gpio/export
echo "out" > /sys/class/gpio/gpio4/direction
# Налаштувати GPIO17 і встановити його як вхідний
echo "17" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio17/direction
# Зробити вивід вихідним, на якому або буде напруга 3,3В, або її
не буде
echo "1" > /sys/class/gpio/gpio4/value
# Прочитати із вхідного вивода
cat /sys/class/gpio/gpio17/value
# Очистити
echo "4" > /sys/class/gpio/unexport
echo "17" > /sys/class/gpio/unexport
```

Макетна плата для реалізації простих пристроїв

Макетна плата (**breadboar**d) - це монтажна плата для розробки прототипів або тимчасових електросхем без використання паяльника.

Всередині макетної плати прокладені провідники таким чином, щоб дозволити збирати досить складні конструкції. В платі є групи з'єднаних між собою контактів, розташованих на відстані 2,54 мм. Схематично макетну плату можна зобразити так (рис.2.8):

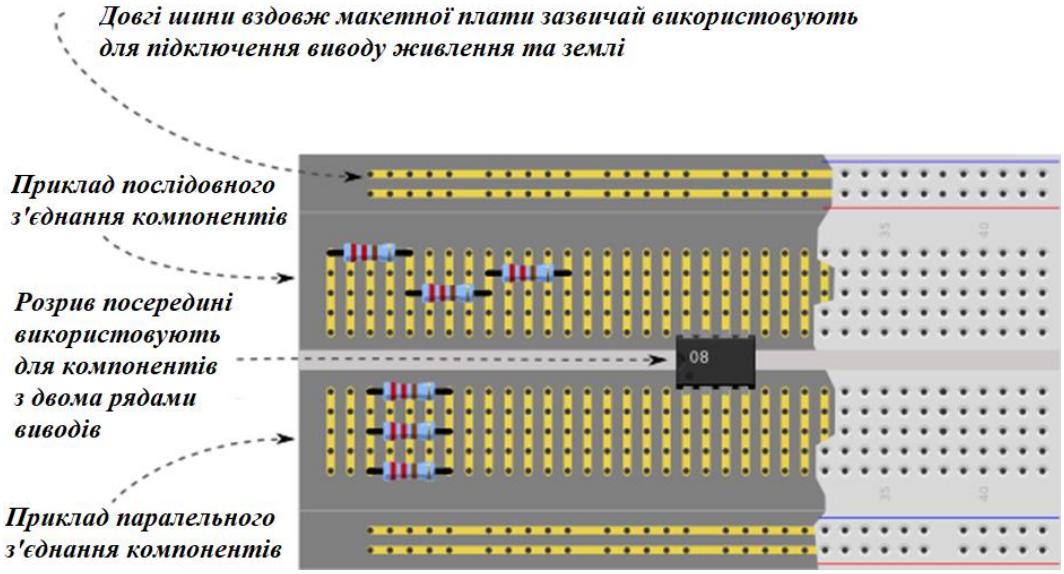


Рисунок 2.8 – Умовне зображення макетної плати

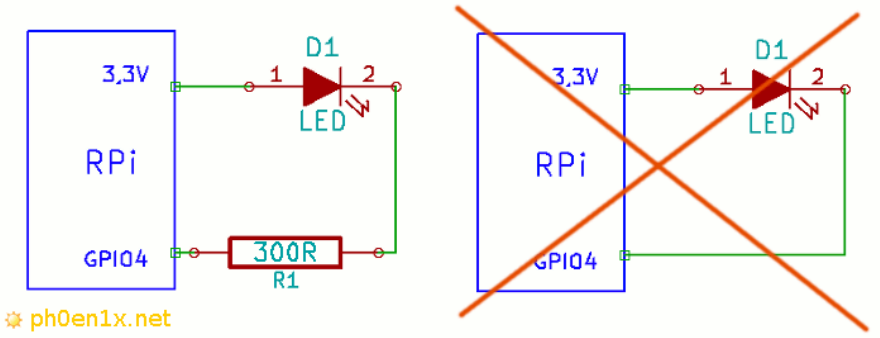
Коли ми підключаємо провідник до одного з отворів в окремому ряді, цей контакт буде одночасно підключений і до решти контактів цього ряду.

На макетних платах прийнято використовувати п'ять отворів на одній шині, і ми можемо підключити до п'яти компонентів включно до окремої шини і вони будуть пов'язані між собою.

В центрі макетної плати вздовж неї проходить окрема канавка, яка ізолює пластини одну від іншої, розділяючи кожен ряд на дві незалежні області. Завдяки цьому можна встановлювати комплектуючі вироби, не замикаючи контакти. Крім ізоляції, ця канавка дозволяє використовувати мікросхеми форм-фактора Dual in-line Package (DIP). У DIP-мікросхем контакти розташовані з двох сторін і відмінно пасують для двох шин по центру плати. У цьому випадку ізоляція контактів – чудовий варіант, який дозволяє зробити розводку кожного контакту мікросхеми на окрему шину з п'ятьма контактами.

Світлодіод – правильне і неправильне підключення

Перші проєкти зробимо з використанням світлодіодів. У наведених нижче прикладах використовується вивід "GPIO4", а в реальному проєкті замість нього може використовуватися будь-який інший з доступних.



а) б)

Рисунок 2.9 – Правильне (а) і неправильне (б) підключення світлодіода до RPi

Як видно з рис.2.9, використаний вивід GPIO4 і при правильному підключенні присутній резистор, який обмежує струм через світлодіод.

При прямому підключенні без резистора на світлодіод буде подана напруга 3,3В, що є більше норми для світлодіодів (2-3В). Пряма напруга такої величини стане причиною чималого струму в ланцюзі (понад 50мА), що може спричинити за собою вигоряння світлодіода і виходу з ладу як окремого буфера GPIO, так і процесора в цілому.

Не треба економити на резисторах, а краще встановити послідовно до кожного виводу резистор приблизно 1К і налаштувати схему для впевненої роботи з такими значеннями опору, ніж потім, в разі помилки, зіпсувати плату RPi.

Розрахунок номіналу резистора для світлодіода

Як правило, для експериментів зі світлодіодами буде досить резистора на 100-600 Ом. При більшому опорі світлодіод буде світитися трохи тьмяніше, а при меншому – яскравіше.

Для більш точного розрахунку опору резистора (при живленні від +3,3В) використовуємо формулу:

R = (3,3В - Uд) / Iд,

де:

• R - опір резистора;

• $U_д$ - робоча напруга світлодіода;

• $I_д$ - робочий струм світлодіода.

Типові значення напруги для світлодіодів різного кольору:

• червоний - 1,8 ... 2В;

• жовтий і зелений - 2 ... 2,4В;

• білі і сині (яскраві) - 3 ... 3,5В.

Робочий струм у світлодіодів, в залежності від типу, може коливатися в межах 10-25мА. Для більш точних розрахунків краще уточнити параметри конкретного світлодіода в довіднику. У нашому випадку для експериментів буде достатній і безпечний струм величиною 10-15мА.

Наприклад, візьмемо світлодіод червоного кольору і розрахуємо для нього резистор при струмі 10мА:

R = (3,3 - 2В) / 0,01А = 130Ом

При опорі резистора 130Ом і живленні напругою 3,3В світлодіод світитиме досить яскраво. Швидше за все, що і при значенні 300Ом яскравості світла діода буде цілком достатньо для експериментів з GPIO в RPi і порт не буде перенавантажений.

Використання Raspberry Pi в стилі Arduino

Однією з популярних бібліотек для роботи з GPIO на RPi сьогодні є WiringPi.

Встановлення WiringPi

1. Якщо у вас не встановлена утиліта `git` для роботи з github-репозиторіями, то встановлюємо її:

```
sudo apt install git-core
```

2. Завантажуємо первинний код бібліотеки з репозиторію:

```
git clone git://github.com/WiringPi/WiringPi
```

3. Встановлюємо:

```
cd WiringPi
sudo ./build
```

Приклади використання

Після того, як встановили бібліотеку WiringPi, розглянемо найпростішу схему: керування світлодіодом (рис.2.10)

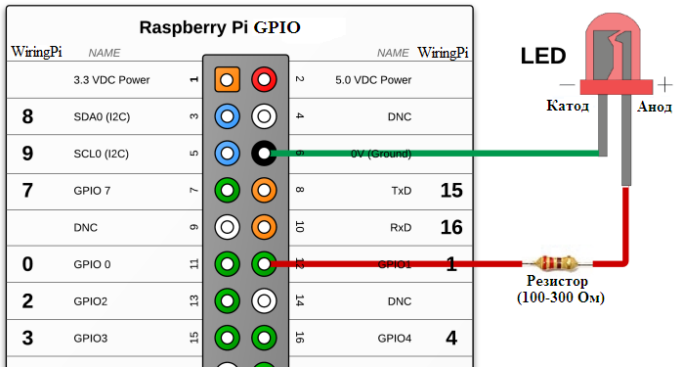


Рисунок 2.10 – Схема керування світлодіодом

На макетній платі схема буде виглядати так (рис.2.11):

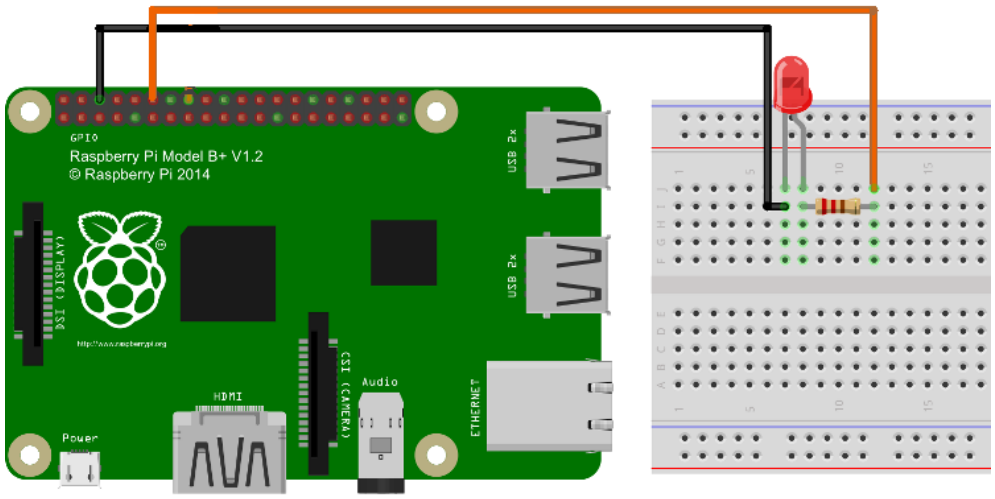


Рисунок 2.11 – Зібрана на макетній платі схема керування світлодіодом

Тестова програма для даної схеми `blink.c`:

```
/*
 * blink.c:
```

```
 *    Проста тестова програма для моргання світлодіода на виводі 1
 */
#include <wiringPi.h>
#include <stdio.h>
int main (void)
  int pin = 1;
  printf("Raspberry Pi wiringPi blink test\n");
  if (wiringPiSetup() == -1)
    exit (1);
  pinMode(pin, OUTPUT);
  for (;;){
    printf("LED On\n");
    digitalWrite(pin, 1);
    delay(250);
    printf("LED Off\n");
    digitalWrite(pin, 0);
    delay(250);
  }
  return 0;
}
```

Залишилось зібрати і запустити:

```
cc -o blink -l wiringPi -L /usr/local/lib blink.c
sudo ./blink
```

Наш світлодіод повинен миготіти. Можемо змінити частоту миготіння.

2.2. Завдання

1. Встановіть на RPi актуальну версію бібліотеки для роботи з GPIO.

2. Повторіть наведений в уроці приклад керування GPIO за допомогою команд `echo`.

3. Встановіть бібліотеку `WiringPi`, зберіть схему керування світлодіодом, наведену в лабораторній роботі, запрограмуйте вмикання/вимикання світло

діода із змінною затримкою: спочатку частота миготіння збільшується, потім – зменшується тощо.

4. В протоколі наведіть приклад програми `blink.c` з доданими до кожного рядка коментарями.

5. Напишіть програму керування трьома світлодіодами за алгоритмом роботи світлофора, коли червоний і зелений світлодіоди горять 2 с, а жовтий – 1 с. Код з коментарями додайте в протокол. Схема підключення діодів показана на рис.2.12.

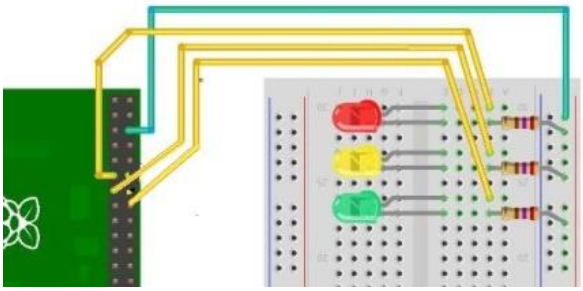


Рисунок 2.12 – Схема для симуляції роботи світлофора

2.3. Зміст звіту

1. Лістинги програм з коментарями.
2. Схеми підключення до RPi.
2. Відповіді на контрольні питання до роботи.
3. Короткі висновки.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

2.4. Контрольні питання

1. Яку максимальну потужність можуть споживати підключені до RPi пристрої?
2. Який рівень логічної «1» на виході RPi?

Лабораторна робота 3

СТВОРЕННЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА З TKINTER

Мета роботи: Навчитися створювати прості графічні інтерфейси користувача в Python з використанням віджетів.

Зміст. Розглядається використання набору інструментів `tkinter` для створення GUI в Python.

3.1. Теоретичні відомості

Елементи GUI - це примірники класів модуля `tkinter`, який входить в пакет Tkinter. При програмуванні треба пройти приблизно такі етапи, щоб створити програму з GUI:

1. Імпортувати бібліотеку.
2. Створити головне вікно.
3. Створити елементи управління (віджети).
4. Встановити властивості віджетів.
5. Визначити події.
6. Визначити обробників подій.
7. Розмістити віджети в головному вікні.
8. Відобразити головне вікно.

Приклад GUI з окремими елементами управління (віджетами) (рис.3.1):

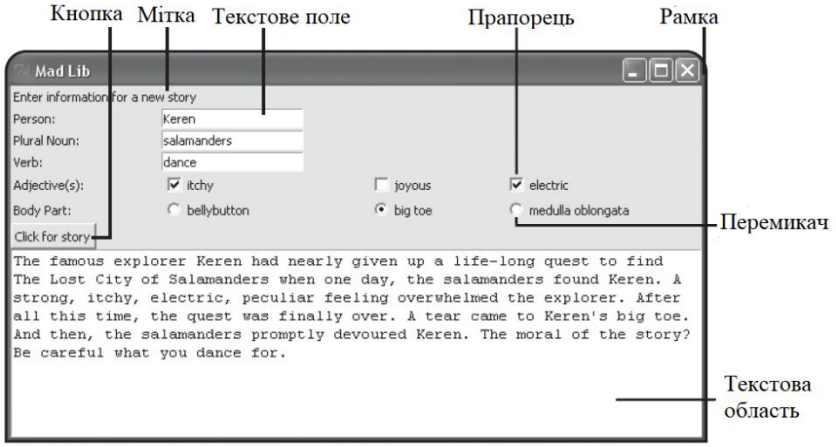


Рисунок 3.1 – Приклад GUI з елементами управління

Tkinter надає різні елементи управління: кнопки, мітки, текстові поля тощо, які використовуються в додатках з графічним інтерфейсом. В даний час є 19 типів віджетів в Tkinter. Представимо ці віджети, а також короткий їх опис в таблиці 3.1:

Таблиця 3.1 – Елементи GUI

Елемент (віджет)	**Клас tkinter**	**Опис класу**
Кнопка	Button	Після натискання користувача виконується якась дія
Фігури	Canvas	Використовується для малювання таких фігур, як лінії, овали, багатокутники і прямокутники у додатку
Прапорець	Checkbutton	Дозволяє користувачу увімкнути або вимкнути будь-який параметр
Текстове поле	Entry	Приймає і відображає один рядок тексту
Рамка	Frame	Служить базою для інших елементів GUI
Мітка	Label	Відображає значок або текст, яким не можна керувати
Параметри	Listbox	Використовується для надання користувачеві списку опцій
Кнопка меню	Menubutton	Використовується для надання користувачеві різних команд, які розміщені всередині кнопки MENU
Меню	Menu	Використовується для відображення в додатку меню
Повідомлення	Message	Використовується для відображення багаторядкових текстових полів для прийому значень від користувача

Продовження табл.3.1.

Перемикач	Radiobutton	Дозволяє користувачу вибрати один параметр із кількох згрупованих
Шкала	Scale	Використовується для забезпечення слайдера віджета
Смуга прокрутки	Scrollbar	Використовується для додавання можливості прокрутки, наприклад, для списків
Текстова область	Text	Приймає і відображає кілька рядків тексту
Вищий рівень	Toplevel	Використовується для забезпечення контейнера окремого вікна
Текстове поле з вибором	Spinbox	Варіант стандартного Tkinter-віджета Entry, який можна використовувати для вибору з фіксованого числа значень
Вікно панелей	PanedWindow	Контейнер, який може містити будь-яку кількість панелей, розташованих горизонтально або вертикально
Рамка мітки	LabelFrame	Простий контейнер, який використовується як прокладка або контейнер для складних віконних розкладок
Вікно повідомлень	tkMessageBox	Використовується для відображення вікон повідомлень в додатках

Зі стовпця класів `tkinter` можемо перейти за посиланням до англомовного підручника, де більш детально пояснюється використання параметрів віджета.

Віджети створюються викликом конструктора відповідного класу. Перший аргумент (як правило неіменований, але можна використовувати ім'я

`master()` – це батьківський віджет, в який буде упакований (поміщений) наш віджет. Батьківський віджет можна не вказувати – в такому випадку буде використане головне вікно програми. Далі йдуть іменовані аргументи, які конфігурують віджет. Це може бути використовуваний шрифт (`font = ...`), колір віджету (`bg = ...`), команда, що виконується при активації віджета (`command = ...`) тощо.

Що таке подійно-орієнтоване програмування

Програми GUI, зазвичай, подійно-орієнтовані. Це означає, що вони відповідають на дії користувача незалежно від порядку здійснення операцій. Подійно-орієнтоване програмування (ПОП) – особливий шлях написання коду, незвичний спосіб мислення програміста. Але, якщо нам колись приходилось користуватися програмами з GUI (наприклад, браузером), то працювати з подійно-орієнтованими системами ми вже вміємо.

Щоб прояснити суть ПОП, розглянемо із зовнішньої сторони інтерфейс користувача на рис.3.1. Якщо б ми вирішили створити щось подібне, користуючись тільки нинішніми навичками програмування на Python, то наша програма, ймовірно, задавала б користувачу серію питань, відповідь на які отримувались би функцією `input()`. Комп'ютер просив би ввести спочатку ім'я людини, потім іменник в множині, потім дієслово тощо. Всі ці дані користувач повинен був би вводити послідовно, по черзі. А подійно-орієнтована програма, наприклад, програма з GUI, дозволить вводити інформацію в довільному порядку.

В ПОП подія (те, що може відбутися з об'єктами програми) зв'язується з обробниками (кодом, який запускається при відповідних подіях). Ось конкретний приклад. Коли в наведеному вище вікні користувач натискає кнопку *Отримати оповідання* (це подія), програма викликає метод, який виводить оповідання на екран (це обробник події). Щоб все назване відбулось, треба зв'язати натискання кнопки з методом, який виводить текст.

Задаючи об’єкти, події і обробники подій, ми задаємо порядок роботи програми. Потім програму треба запустити, створивши цикл події, всередині якого вона буде чекати описаних вами подій. Коли будь-яка з цих подій відбувається, програма обробляє її запропонованим нами чином. Засвоївши логіку роботи кількох зразків, зрозуміємо, як створювати власні подійно-орієнтовані додатки.

Базове вікно (Frame)

Відправною точкою будь-якої програми з графічним інтерфейсом - базове, або кореневе, вікно, над яким розміщуються решта елементів GUI. Якщо представити собі GUI у вигляді дерева, то це буде корінь. Наше дерево може «гілкуватися» в різних напрямках, але кожна його частина прямо або опосередковано буде прив’язана до коріння.

Програма «Найпростіший GUI»

Якщо запустити програму на основі `tkinter` прямо з IDLE, то або програма, або IDLE зависне. Простіше всього вирішити цю проблему, безпосередньо запускаючи програму; в Windows для цього достатньо двічі натиснути на її значок. Хоча після подвійного клацання кнопкою миші віконний додаток і почне працювати, виникає друга складність: якщо код програми має помилку, то консольне вікно закриється раніше, ніж ми встигнемо прочитати опис цієї помилки. При роботі в Windows оптимально створити пакетний файл, який буде запускати програму і чекати закінчення її роботи. В результаті, вікно консолі залишиться відкритим і повідомлення про помилки буде видно. Якщо, наприклад, програма називається `simple_gui.py`, то пакетний файл, який запускає її, повинен складатися всього з двох рядків:

```
simple_gui.py
pause
```

Цей файл треба викликати для виконання, двічі клацнувши на його значок. Щоб створити пакетний файл, відкриємо простий текстовий редактор, наприклад

Блокнот (але не Word або WordPad), потім введемо код. Збережемо файл з розширенням `.bat` (наприклад, `simple_gui.bat`). Переконаємося, що за `bat` не йде розширення `.txt`.

Програма «Найпростіший GUI» може (не на всіх операційних системах) викликати до життя ще одне вікно – знайому нам консоль. Коли (або якщо) у програмі з інтерфейсом Tkinter виникне помилка, у вікні консолі з'являться детальні відомості про неї. Консоль не варто закривати ще й тому, що в цьому випадку GUI також зупинить роботу.

Отримавши безпомилкову роботу свого GUI-додатка, ми, можливо, побажаємо видалити друге вікно. Щоб зробити це на комп'ютері з операційною системою Windows, найпростіше змінити розширення файлу з `.py` на `.pyw`.

Імпорт модуля tkinter

Найперше, що треба зробити в програмі «Найпростіший GUI» – це завантажити модуль `tkinter`:

```
# Найпростіший GUI
# Демонструє створення вікна
from tkinter import *
```

Цей рядок безпосередньо перенесе весь зміст `tkinter` в глобальну область видимості програми.

Примітка. В Python 2.x.x при імпорті модуля `Tkinter`, він пишеться з великої літери (інакше виникає помилка). При імпорті `tkinter` у версіях 3.x.x Python ім'я модуля пишеться з маленької літери.

Створення базового вікна

Щоб створити базове вікно, ініціалізуємо клас `Tk` з модуля `tkinter`:

```
# створення базового вікна
root = Tk()
```

Звернемо увагу на те, що префікс `tkinter` до назви класу `Tk` не додається. Власне, будь-яка частина модуля `tkinter` доступна без префіксу. Оскільки

зазвичай в програмах на основі цього пакету є багато посилань на класи і константи всередині модуля, програміст таким чином звільняється від маси непотрібної праці, а лише читає код. В програмах з інтерфейсом `Tkinter` може бути тільки одне базове вікно. Якщо створити два їх примірники, то програма перестане відповідати, бо вікна будуть сперечатися за пріоритет.

Зміна вигляду базового вікна

Тепер скористаємось двома методами базового вікна, щоб змінити його зовнішній вигляд:

```
# зміна вікна
root.title("Найпростіший GUI")
root.geometry("200x100")
```

Метод `title()` призначає заголовок вікна. Досить передати йому рядок, який хочемо відобразити в заголовку. В прикладі самий верхній рядок вікна отримує текст "Найпростіший GUI" (рис.2).

Метод `geometry()` встановлює розміри базового вікна в пікселях. Цей метод приймає рядковий (а не цілочисельний) аргумент, в якому ширина і висота вікна повинні бути розділені символом x. Тут назначили ширину вікна рівною 200 пікселів, а висоту – 100 пікселів.

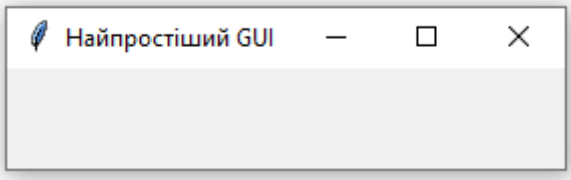


Рисунок 3.2 – Найпростіший GUI

Запуск подійного циклу базового вікна

Тепер, нарешті, можна викликати метод `mainloop()`, щоб почати цикл події базового вікна:

```
# старт циклу події
root.mainloop()
```

Як результат, вікно залишається відкритим і чекає подій для подальшого оброблення. Оскільки не описана жодна подія, функціональність отриманого вікна невелика. Однак це вже абсолютно повноцінне вікно, яке можна змінювати за розмірами, згорнути або закрити. Для експериментів із ним просто натиснемо двічі на значок пакетного файлу `simple_gui.bat`.

Застосування міток

До складу GUI входять різні елементи управління. Мабуть, немає іншого настільки ж простого елементу управління, як мітка – текст і/або зображення, які не можна редагувати. У відповідності зі своєю назвою мітка «позначає» якусь частину інтерфейсу, наприклад, другий елемент. На відміну від більшості елементів управління, мітки не інтерактивні: якщо користувач натисне на мітку, то система ніяк не відреагує. Втім, це не заважає міткам бути по-своєму корисними. При створенні будь-якого графічного інтерфейсу ми хоч раз використаємо мітку.

Програма «Це я, мітка»

Налаштування програми

Спочатку виконаємо налаштування: імпортуємо модуль `tkinter` і створимо базове вікно.

```
#Це я, мітка
# Демонструє застосування міток
from tkinter import *
# створення базового вікна
root = Tk()
root.title("Це я, мітка")
root.geometry("200x50")
```

Запуск циклу події базового вікна

Нарешті, починає роботу цикл події базового вікна, а разом з ним і весь GUI:

```
# старт циклу події
root.mainloop()
```

Створення рамки

Рамка - це такий елемент управління, всередині якого можуть бути інші елементи, наприклад, мітки. Створюємо нову рамку:

```
# всередині вікна створюється рамка для розміщення інших елементів
app = Frame(root)
```

Щоразу при створенні чергового елементу управління ми повинні передавати конструктору нового об'єкта його батьківський елемент, тобто той елемент, всередині якого він знаходиться. В даному випадку методу-конструктору `Frame` було передано `root`. Як наслідок, всередині базового вікна з'являється нова рамка. Тепер викликаємо метод `grid()` нового об'єкта:

```
app.grid()
```

Метод `grid()` є у всіх елементів управління. Він зв'язаний з менеджером розміщення, з якого можна управляти розміщенням елементів у вікні.

Створення мітки

В програмі всередині базового вікна створюється елемент `Label`, який просто оголошує про своє існування (рис.3.3).

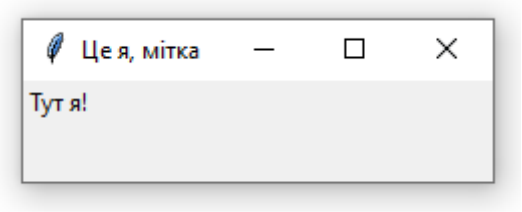


Рисунок 3.3 – Створення мітки

Для створення мітки ініціалізуємо клас `Label`:

```
# створення мітки всередині рамки
lbl = Label(app,text = "Це я, мітка!")
```

Конструктору об’єкту `Label` передали аргумент `app`. Тим самим заставили інтерпретатор вважати, що рамка `app` – батьківський елемент відносно мітки. От чому мітка буде розміщена всередині рамки. В елементів управління є параметри, значення яких може вибирати програміст. Багато з цих параметрів визначають зовнішній вигляд елемента. Так, наприклад, ми передали параметру `text` текст `"Тут я!"`, і при відображенні мітки на екрані з’являються слова `Тут я!`. Тепер викликаємо метод `grid()` об’єкту-мітки: `lbl.grid()`

Нарешті, але не менш важливо, я викликаю цикл подій кореневого вікна, щоб запустити GUI: `root.mainloop()`

Завдяки цьому мітка буде відображена в складі GUI.

Керування розміщенням віджетів у вікні

Для керування розміщенням використовують функцію `pack()` – це так званий пакувальник, або менеджер розміщення. Він відповідає за те, як віджети будуть розміщуватися у головному вікні. Для кожного віджета треба викликати метод пакувальника, в іншому випадку він не буде відображений. Всього пакувальників три:

`pack()`. Автоматично розміщує віджети в батьківському вікні (рис.3.4).

Має параметри `side`, `fill`, `expand`.

Приклад:

```
from tkinter import *
root = Tk()
Button(root, text = '1').pack(side = 'left')
Button(root, text = '2').pack(side = 'top')
Button(root, text = '3').pack(side = 'right')
Button(root, text = '4').pack(side = 'bottom')
Button(root, text = '5').pack(fill = 'both')
root.mainloop()
```

Рисунок 3.4 – Метод `pack()`

`grid()`. Розміщує віджети на сітці. Основні параметри: `row/column` – рядок/стовпець у сітці, `rowspan/columnspan` – скільки рядків/стовпців займає віджет (рис.3.5).

Приклад:

```
from Tkinter import *
root = Tk()
Button(root, text = '1').grid(row = 1, column = 1)
Button(root, text = '2').grid(row = 1, column = 2)
Button(root, text = '__3__').grid(row = 2, column = 1, columnspan
= 2)
root.mainloop()
```

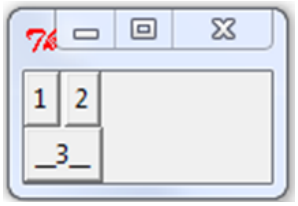


Рисунок 3.5 – Метод `grid`

`place()`. Дозволяє розміщувати віджети в указаних координатах із зазначеними розмірами (рис.3.6).

Основні параметри: `x, y, width, height`.

Приклад:

```
from tkinter import *
root = Tk()
Button(root, text = '1').place(x = 10, y = 10, width = 30)
Button(root, text = '2').place(x = 45, y = 20, height = 15)
Button(root, text = '__3__').place(x = 20, y = 40)
root.mainloop()
```

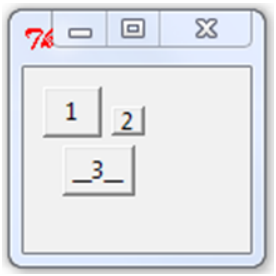


Рисунок 3.6 – Метод `place`

Застосування кнопок

Кнопку, елемент управління класу Button, користувач може потім-або активувати – натиснути. Оскільки нам уже відомо, як створювати мітки, процедура створення кнопок не повинна викликати труднощів.

Програма «Безкорисні кнопки»

Ми вже створювали кілька кнопок, які, якщо їх натискати, ніяк не відповідають. Розмістити на графічному інтерфейсі такі кнопки – все рівно що прикрутити до стелі люстру без лампочок. Вона вже на своєму місці, але ще не функціональна, як кнопки на рис.3.7.

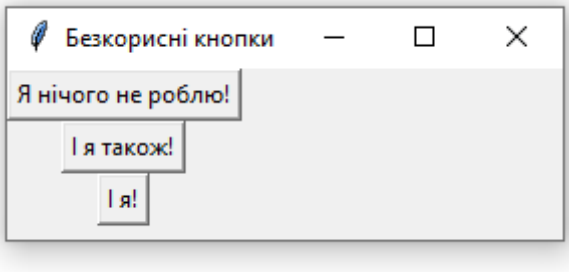


Рисунок 3.7 – Рамка з кнопками

Налаштування програми з безкорисними кнопками

Спочатку, як зазвичай, імпортуємо `tkinter`, створимо базове вікно і рамку в ньому:

```
# Безкорисні кнопки
# Демонструє створення кнопок
from tkinter import *
# створення базового вікна
root = Tk()
root.title("Безкорисні кнопки")
```

```
root.geometry("200x85")
# всередині вікна створюється рамка для розміщення інших елементів
app = Frame(root)
app.grid()
```

Створення кнопок

Щоб створити кнопку, інстанціюваний клас `Button`. Це виконують наступні рядки коду:

```
# створення кнопки всередині рамки
bttn1 = Button(app, text = "Я нічого не роблю!")
bttn1.grid()
```

Тут створюється нова кнопка з написом `Я нічого не роблю!`. Її батьківський елемент - раніше створена рамка; таким чином, кнопка буде розміщена всередині рамки.

Коли справа доходить до створення, опису і зміни зовнішнього вигляду елементів управління, модуль `tkinter` виявляє немало гнучкості. Так, можна створити елементи і визначити всі його параметри одним рядком коду, як тут зроблено. Можна, навпаки, спочатку створити елемент, а потім задати чи змінити його вигляд. На прикладі наступної кнопки зрозумієте, про що йде мова. Спочатку створюється нова кнопка:

```
# створення другої кнопки всередині рамки
bttn2 = Button(app)
bttn2.grid()
```

Звернемо увагу, що конструктору об’єкту передається лише значення `app` – ім’я батьківського елементу кнопки. Отже, поки лише добавили всередині рамки пусту кнопку. Але це легко виправити. Вигляд кнопки можна змінити після створення, для чого існує метод об’єкту `configure()`:

```
bttn2.configure(text = "І я також!")
```

Цей рядок присвоює параметру `text` даної кнопки значення `"І я також!"`, тому відповідний текст з’являється на кнопці. Метод `configure()` дозволяє

конфігурувати будь-який параметр елемента будь-якого типу. З його допомогою можна навіть назначити нове значення вже установленому параметру. Тепер створимо ще одну кнопку:

```
# створення третьої кнопки всередині рамки
bttn3 = Button(app)
bttn3.grid()
```

В цьому випадку значення параметру `text` буде вибране іншим способом:

```
bttn3["text"] = "I я!"
```

Як бачите, здійснений доступ до параметру `text` через інтерфейс, подібний словнику. Установлено значення `"I я!"`, так що на кнопці буде відображатися текст `I я!`. Щоб присвоювати значення в такому «словниковому» стилі, треба вказувати як ключі імена параметрів, взяті в лапки.

Запуск циклу події базового вікна

Як завжди, для запуску GUI треба почати роботу циклу події базового вікна

```
# старт циклу події
root.mainloop()
```

Визначення подій та їх обробників

Подій та способів їх обробки безліч, тому розглянемо лише приклад, коли задачею кнопки є виведення якого-небудь повідомлення в потік виведення, використовуючи функцію `print`. Робити це вона буде при натисканні на неї лівою кнопкою миші.

Дії (алгоритм), які відбуваються при тій чи іншій події, можуть бути досить складними. Тому часто їх оформляють у вигляді функції, а потім викликають, коли вони потрібні. Нехай у нас виведення на екран оформлене у вигляді функції `printer`:

```
def printer(event):
    print ("Як завжди, черговий 'Hello World!'")
```

Відзначимо, що функцію бажано (майже обов’язково) розміщувати на початку коду. Параметр `event` – це якась подія.

Подія натискання лівою кнопкою миші виглядає так: `<Button-1>`. Потрібно зв'язати цю подію з обробником (функцією `printer`). Для зв'язку призначений метод `bind`. Синтаксис зв'язування події з обробником виглядає так:

```
but.bind("<Button-1>",printer)
```

3.2. Завдання

Завдання 1. Повторіть наведені в модулі вправи. Скріншоти виконання додайте в протокол ЛР.

Завдання 2. Зробіть ескіз та створіть GUI для керування яскравістю та кольором RGB-діода за допомогою двох віджетів Scale, щоб колір можна було змінювати від червоного до фіолетового (як в спектрі сонячного світла).

3.3. Зміст звіту

1. Скріншоти виконання завдання 1.
2. Ескіз, код з коментарями для завдання 2.
2. Відповіді на контрольні питання до роботи.
3. Короткі висновки.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

3.4. Контрольні питання

1. Які параметри можна задати/змінити в класі Listbox?
2. Що буде з GUI, якщо заданий розмір кнопки буде більший заданого розміру рамки?

Лабораторна робота 4

ВИКОРИСТАННЯ ШІМ ТА СТВОРЕННЯ ІНТЕРФЕЙСУ ДЛЯ КЕРУВАННЯ СВІТЛОДІОДАМИ

Мета роботи: Вивчити використання широтно-імпульсної модуляції (ШІМ) для керування потужністю сигналу, який подаємо з виходів мікрокомп’ютера.

Зміст. Розглядається управління яскравістю світлодіода за допомогою ШІМ, бо регулюючи напругу на світлодіоді, це зробити не зможемо із-за великої мертвої зони, коли напруга недостатня і світлодіод не буде світитися.

4.1. Теоретичні відомості

Для управління яскравістю світлодіодів використовують аналогові виходи ШІМ – широтно-імпульсна модуляція (PWM - Pulse-Width Modulation). Дійсно, ШІМ є методом управління потужністю. Ми використовуємо її, щоб контролювати потужність, подану на світлодіод, а отже, змінювати яскравість його світла.

На діаграмі (рис.4.1) показаний сигнал на виводі ШІМ Raspberry Pi:

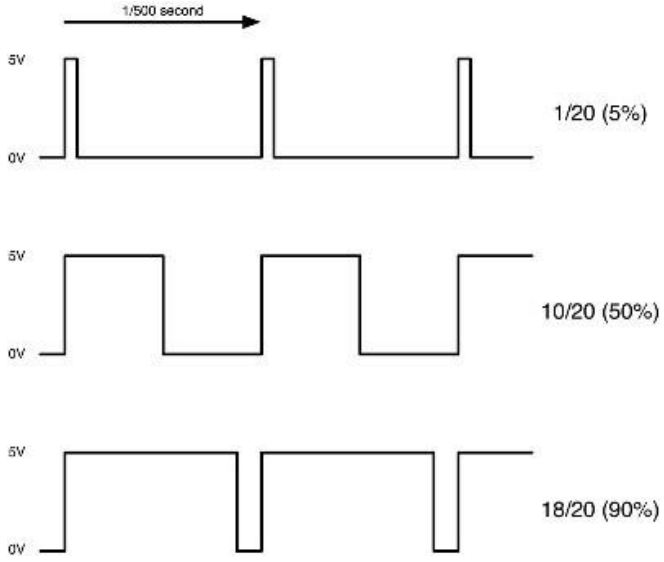


Рисунок 4.1 – Сигнали на виході ШІМ

Кожні 1/500 секунди ШІМ-вихід виробляє імпульс. Довжина цього імпульсу регулює кількість енергії, яку отримує світлодіод чи двигун. Зовсім немає жодного імпульсу – світлодіод не світиться, короткий імпульс – діод буде світитися тьмяно. Якщо імпульс активний протягом половини періоду, то світлодіод отримає половину потужності тощо.

Світлодіоди можуть вмикатися і вимикатися дуже швидко, зазвичай, менше, ніж за мільйонну частку секунди, тому при використанні ШІМ зі світлодіодами, вони насправді блимають з частотою ШІМ, але око бачить ту їх яскравість, яка залежить від частки часу, коли світлодіод насправді горить.

Червоний, зелений і синій світлодіоди (RGB LED) можуть бути одним світлодіодом, який фактично містить три світлодіоди. Три світлодіоди: червоний, зелений і синій. При використанні ШІМ для керування яскравістю кожного зі світлодіодних кольорів, ми можемо зробити, щоб світлодіод світився будь-яким кольором.

Хоча RGB LED містить три нормальні двох вивідні світлодіоди, це не означає, що світлодіодний корпус повинен мати шість виводів, тому що один вивід кожного світлодіоду може бути єдиним спільним (рис.4.2):

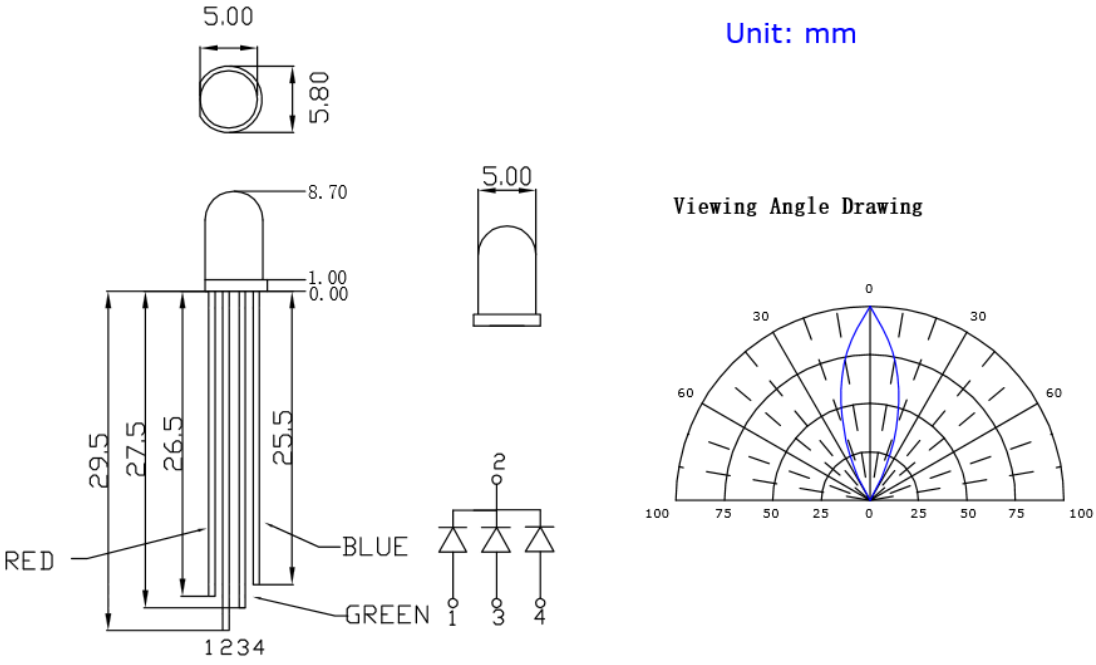


Рисунок 4.2 – RGB-діод

Якщо катоди кожного світлодіоду зв'язані один з одним, то отриманий спільний вивід називається загальним катодом, а якщо аноди є загальними, то спільний вивід називається загальним анодом. Корпус RGB LED може бути прозорим або дифузним. Якщо він прозорий, то ми зможемо побачити червоні, зелені та сині світлодіоди всередині корпусу, і колір буде змішуватися не зовсім добре. Дифузні корпуси змішують світло від трьох світлодіодів набагато краще.

Використання PWM в RPI.GPIO

Для створення прикладу PWM:

```
p = GPIO.PWM(канал, частота)       # де канал - вивід RPi, частота
- значення частоти в Гц
```

Для запуску PWM:

```
p.start(dc)       # де dc - шпаруватість (0.0 <= dc <= 100.0)
```

Для зміни частоти:

```
p.ChangeFrequency(частота)    # де частота є нова частота в Гц
```

Для зміни шпаруватості:

```
p.ChangeDutyCycle(dc)         # де 0.0 <= dc <= 100.0
```

Для зупинки PWM:

```
p.stop()
```

Звертаємо увагу, що PWM також зупиняється, якщо змінна примірника 'p' виходить з області видимості.

Змішування кольорів

Використаємо RPi, щоб контролювати колір з RGB LED. Для цього створимо графічний інтерфейс користувача (GUI) з трьома повзунками, які використовуються для управління кольором.

Ми вже знаємо, що GUI – засіб візуальної взаємодії між користувачем і комп'ютером. GUI застосовується у всіх популярних операційних системах на комп'ютерах; він спрощує дії користувача і робить їх однообразними.

Схема під'єднання світлодіода (рис.4.3):

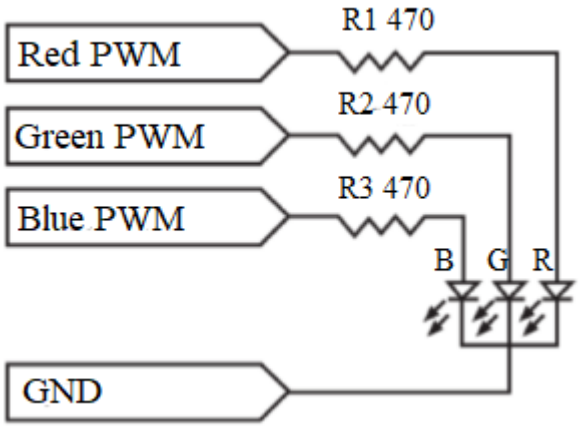


Рисунок 4.3 – Схема під'єднання RGB-діода

Для оптимальної яскравості і кращої передачі кольору, ми повинні ретельно вибрати номінали резисторів. Проте, легше купити свої компоненти, якщо використовуємо один і той же номінал резистора для всіх трьох каналів. У цьому випадку, резистори 470Ом вважаються "універсальним" варіантом, який буде чудово працювати з RPi.

Яскравість і ефективність RGB LED така, що навіть для 3мА світлодіод все ще буде виглядати досить яскраво.

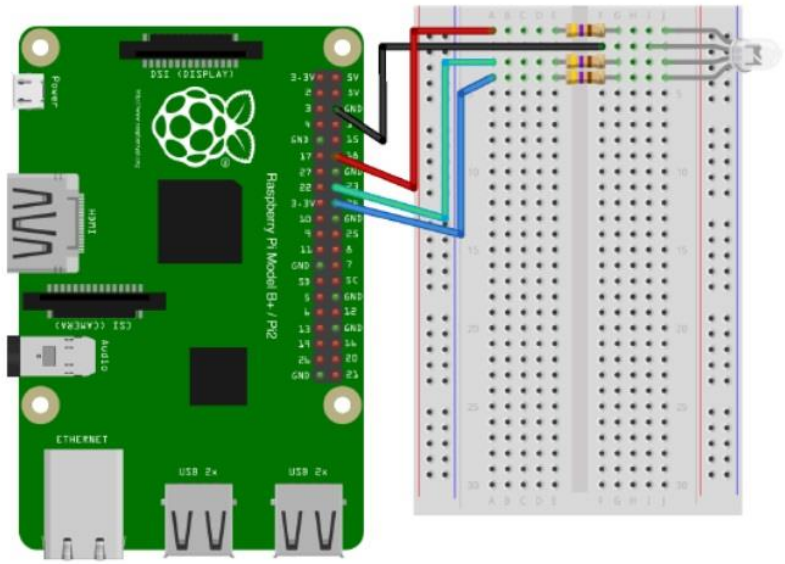


Рисунок 4.4 – Схема підключення RGB-діода до Raspberry Pi

Програмне забезпечення Raspberry Pi

Програма Python для цього експерименту використовує фреймворк під назвою Tkinter, що дозволяє створювати додатки, які запускаються у вікні і мають елементи управління користувальницького інтерфейсу, а не простий командний рядок, який ми використовували досі. Це робить програму трохи довшою, ніж зазвичай. Це також змушує використовувати деяке більш просунуте програмування.

Давайте подивимося на код програми, який потім збережемо у файлі `mixing_colors.py`:

```
from tkinter import *
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)                    #1
GPIO.setup(18, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
GPIO.setup(24, GPIO.OUT)
pwmRed = GPIO.PWM(23, 500)                #2
pwmRed.start(100)
pwmGreen = GPIO.PWM(18, 500)
```

```
pwmGreen.start(100)
pwmBlue = GPIO.PWM(24, 500)
pwmBlue.start(100)
class App:
    def __init__(self, master):        #3
        frame = Frame(master)          #4
        frame.pack()
        Label(frame, text='Red').grid(row=0, column=0)      #5
        Label(frame, text='Green').grid(row=1, column=0)
        Label(frame, text='Blue').grid(row=2, column=0)
        scaleRed = Scale(frame, from_=0, to=100,            #6
        orient=HORIZONTAL, command=self.updateRed)
        scaleRed.grid(row=0, column=1)
        scaleGreen = Scale(frame, from_=0, to=100,
        orient=HORIZONTAL, command=self.updateGreen)
        scaleGreen.grid(row=1, column=1)
        scaleBlue = Scale(frame, from_=0, to=100,
        orient=HORIZONTAL, command=self.updateBlue)
        scaleBlue.grid(row=2, column=1)
    def updateRed(self, duty):                               #7
        # change the led brightness to match the slider
        pwmRed.ChangeDutyCycle(float(duty))
    def updateGreen(self, duty):
        pwmGreen.ChangeDutyCycle(float(duty))
    def updateBlue(self, duty):
        pwmBlue.ChangeDutyCycle(float(duty))
root = Tk()                                                  #8
root.wm_title('RGB LED Control')
app = App(root)
root.geometry("200x150+0+0")
try:
    root.mainloop()
finally:
    print("Cleaning up")
    GPIO.cleanup()
```

Пояснимо пронумеровані рядки:

1. Конфігуруємо RPi, використовуючи імена Broadcom (BCM) виводів, а не їх позиції.

2. Запускаємо широтно-імпульсну модуляцію (ШІМ) на червоному, зеленому і синьому каналах для управління яскравістю світлодіодів.

3. Ця функція викликається при створенні додатка.

4. Фрейм має різні елементи управління GUI.

5. Створюємо мітки і розташовуємо їх в сітці макету.

6. Створюємо повзунки і розташовуємо їх в сітці макету. Атрибут `command` задає метод для виклику, коли повзунок переміщається.

7. Цей метод та аналогічні методи для інших кольорів викликаються, коли їх повзунок переміщається.

8. Встановлюємо GUI на запуск і задаємо вікну заголовок, розмір і положення.

Запускамо програму від імені суперкористувача, використовуючи наступну команду:

```
sudo python mixing_colors.py
```

Через секунду або дві, з'явиться вікно, показане на рис.4.5:

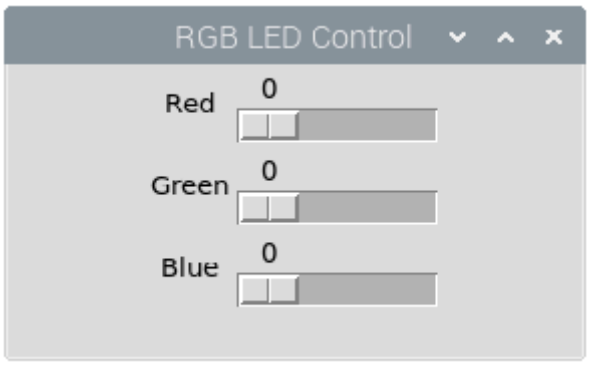


Рисунок 4.5 – Інтерфейс керування змішуванням кольорів світлодіода

При переміщенні повзунка колір світлодіоду змінюється.

4.2. Завдання

1. Зібрати схему керування одним світлодіодом

2. Написати програму за допомогою якої перевірити роботу розглянутих вище команд `GPIO.PWM`.

4.3. Зміст звіту

1. Скріншоти виконання завдання.

2. Код з доданими до рядків сценарію коментарями.

2. Відповіді на контрольні питання до роботи.

3. Короткі висновки.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

4.4. Контрольні питання

1. Чому не можна керувати яскравістю світлодіоду, подаючи напругу з виходу мікросхеми безпосередньо, без ШІМ?

2. Як визначити величину опору обмежувального резистора для світлодіоду?

Лабораторна робота 5

ПІДКЛЮЧЕННЯ СЕНСОРІВ З ІНТЕРФЕЙСОМ 1-WIRE ДО RASPBERRY PI

Мета роботи: Навчитися використовувати інтерфейс 1-Wire для організації взаємодії мікрокомп’ютера із периферійними пристроями.

Зміст. В роботі досліджується використання протоколу1-Wire для передачі даних в обидві сторони по одній лінії.

Особливості протоколу 1-Wire

Протокол 1-Wire розроблений корпорацією Dallas Semiconductor (зараз Maxim Integrated) ще в 90-х роках, але активно використовується і зараз. На 1- Wire сьогодні працює більшість "пігулок" - домофонних чіпів (DS1990A), карток доступу, а також через 1-Wire спілкуються популярні сенсори температури (DS18S20 і DS18B20), транзисторні ключі (DS2405, DS2406), програмовані порти введення-виведення (DS2408), АЦП і ЦАП, годинник реального часу (DS2417) тощо.

Режим зв'язку в цьому протоколі – асинхронний і напівдуплексний, а при надсиланні мультибайтних цілих передача йде від молодшого байта до старшого. При цьому у нас завжди є ведучий – один пристрій на шині, який відсилає команди, і ведені – пристрої, які ці команди приймають і відповідають на них, якщо необхідно; кожний з ведених пристроїв підключається безпосередньо до загальної шині.

Ще раз підкреслимо – на шині може бути лише один ведучий, інакше виникнуть конфлікти, коли обидва ведучих «тягнутимуть ковдру на себе».

Протокол 1-Wire хороший тим, що не складний в реалізації і вимагає для зв'язку всього два-три дроти (шина даних, земля і, при необхідності, живлення); однак при цьому він не позбавлений і недоліків – цей протокол досить чутливий до часу і до завад. Також 1-Wire не призначений для передачі великих обсягів

інформації та для швидкісного обміну даними – максимальна швидкість 15,4 Кбіт/с.

Відстань передачі досягає 300м, якщо дотримуватися певних умов:

- застосування кабеля типу "вита пара";
- використання спеціального драйвера мережі (активне підтягування з врахуванням струму в лінії);
- використання технології «спільна шина» з єдиним стовбуром (не вільна топологія).

Фізично для організації інтерфейсу необхідні як мінімум лінія для даних і "земля"; досить часто також для підключення пристроїв необхідна також лінія живлення, проте деякі ведені пристрої можуть живитися і паразитно – отримувати "підживлення" через шину даних. Згідно інструкції з використання, шина даних повинна бути підтягнута до живлення резистором 4,7кОм, однак даний номінал використовується при відносно коротких лініях; якщо ж відстань між пристроями досить велика, то опір резистора можна зменшити.

Обмін інформацією ведеться так званими тимчасовими, або тайм-слотами (60 мкс): один тайм-слот служить для обміну одним бітом інформації. Дані передаються біт за бітом, починаючи з молодшого біта молодшого байта - це, до речі, досить часто призводить до помилок у новачків – здається, що потрібно передавати дані зліва направо, так, як вони зберігаються в пам'яті – але при передачі по 1-Wire, наприклад, двобайтового числа порядок передачі буде таким:

Маємо число 1023410, яке в двійковому вигляді виглядає як: 00100111 11111010.

Передача по 1-Wire буде виглядати так:

$$0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow 0 \rightarrow 1 \rightarrow 0 \rightarrow 0$$

При обміні інформацією ведучий ініціює кожний зв'язок на бітному рівні. Це означає, що передача кожного біта, незалежно від напрямку (передача чи прийом), повинна бути ініційована ведучим. Шина даних за замовчуванням

підтягується до "одиниці", тому для початку як прийому, так і для передачі ведучий опускає лінію на деякий час в "нуль".

Увага: ні ведучий, ні ведені не виставляють на шині "одиницю" – це може призвести до короткого замиканням: якщо один пристрій виставить на шині "1", а інший - "0"; тому як ведучий, так і ведений можуть використовувати тільки два стани: "на вихід в нуль" і "z-стан" (на вхід без підтягування). Підтягування до живлення здійснюється резистором.

Розглянемо 5 основних команд для зв'язку по шині 1-Wire: "Запис 1", "Запис 0", "Читання", "Скидання" і "Присутність".

На малюнках червоним виділено управління лінією від ведучого, синім - управління лінією від веденого, чорним – звільнена лінія (за допомогою підтягування шина автоматично переходить в "одиницю").

Сигнал "Запис 1". Ведучий встановлює низький рівень протягом 1 ... 15 мкс (рис.5.1). Після цього, протягом решти тимчасового слота він звільняє шину.

Рисунок 5.1 – Встановлення низького рівня ведучим

Сигнал "Запис 0". Ведучий формує низький рівень протягом не менше 60 мкс (рис.5.2), але не довше 120 мкс.

Рисунок 5.2 – Формування сигналу запису 0

Сигнал "Читання". Ведучий встановлює низький рівень протягом 1 ... 15 мкс. Після цього ведений, якщо хоче передати 0, утримує шину в низькому стані до 60 мкс; якщо ж ведений хоче передати 1, то він просто звільняє лінію. Ведучий

зазвичай сканує стан шини після закінчення 15 мкс після встановлення низького рівня на шині (5.3).

Рисунок 5.3 – Формування сигналу читання

Так, ведений утримує лінію до землі, якщо хоче передати "0", і просто відпускає лінію, якщо хоче передати "1".

Таким чином при читанні отримуємо наступні діаграми (рис.5.4-5.5):

Сигнал "Читання при отриманні 1" (рис.5.4):

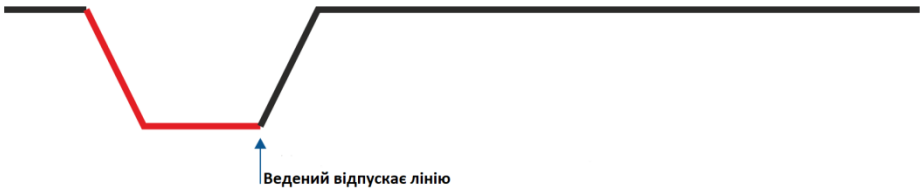


Рисунок 5.4 – Сигнал читання при отриманні 1

Сигнал "Читання при отриманні 0" (рис.5.5):

Рисунок 5.5 – Сигнал читання при отриманні 0

Основні проблеми, які виникають при реалізації читання-запису – це проблеми з часом, тобто "невитримування", або навпаки, "перевитримування" тимчасових затримок при читанні лінії. Виникають ці проблеми через те, що часто не роблять поправку на затримку виконання команд мов програмування високого рівня. Особливо це стосується різних "додаткових" функцій.

Сигнал "скидання/присутність". Тут тимчасові інтервали імпульсів відрізняються. Ведучий встановлює низький рівень протягом 8 тимчасових слотів (480 мкс), а потім звільняє шину. Даний тривалий період низького стану називається сигналом "скидання".

Якщо на шині присутній ведений, то він повинен протягом 60 мкс після звільнення вкдучим шини встановити низький рівень тривалістю не менше 60 мкс. Даний відгук носить назву "присутність". Якщо такий сигнал не виявляється, то ведучий повинен думати, що немає підключених пристроїв до шини і подальший зв'язок неможливий.

Дана послідовність сигналів завжди починає будь-який обмін інформацією між пристроями.

Крім цього, потрібно враховувати, що будь-який ведений пристрій після підключення живлення відразу ж видає сигнал присутності.

Сигнал "скидання" (рис.5.6) дозволяє ведучому достроково завершити обмін інформацією – наприклад, якщо сенсор температури передає нам всю свою пам'ять, а нам потрібні тільки перші два байта, які містять значення температури, то після отримання цих двох байт мікросхема просто може опустити лінію в нуль на потрібну тривалість часу – сенсор зрозуміє, що більше нічого пересилати не треба.

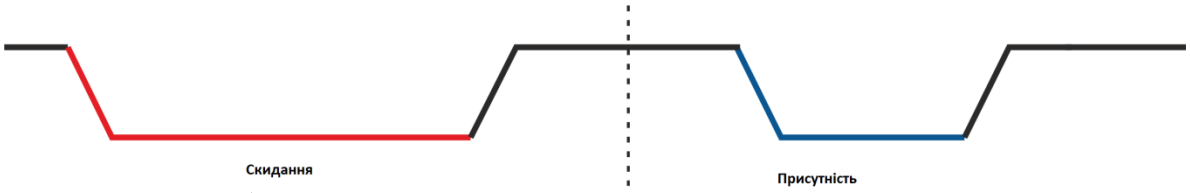


Рисунок 5.6 – Формування сигналу скидання

При реалізації зазвичай необхідно запрограмувати мікроконтролер чи мікрокомп’ютер як ведучий пристрій, тому мікрокомп’ютеру варто генерувати сигнали тривалості трохи більшої необхідного мінімуму, а відповіді від ведених пристроїв чекати за найгіршими прогнозами – тоді взаємодія по протоколу буде оптимальною.

При цьому ведучому треба не забувати періодично перевіряти стан лінії даних, що вона повертається в підтягнутий стан з плином часу, а то може виникнути ситуація. Наприклад, який-небудь ведений зламався і закорочує, лінію в нуль – і, в принципі, протокол не перевіряє дану проблему сам, хоча помилка може і не виникнути.

Розглянемо "вищий" рівень протоколу 1-Wire – послідовність дій при взаємодії ведучого і веденого, а також основні команди на прикладі використання температурних сенсорів.

Використання температурних сенсорів з Raspberry Pi

RPi не має виводів АЦП (аналого-цифрового перетворювача) в GPIO, і тому зможемо використати лише цифровий сенсор температури.

Сенсор, який найчастіше вибирають, це Dallas DS18B20 – сенсор температури з інтерфейсом 1-Wire (однопровідний). Він відносно дешевий, його легко знайти, простий у використанні і показує температуру з точністю до +/-0,5 градуса у всьому діапазоні від -10 до +85 градусів Цельсія.

Для того, щоб він взаємодіяв з RPi, нам потрібний лише один резистор на 4,7К, який діє як «підтягуючий» резистор, і підключити все, як показано на рис.5.7:

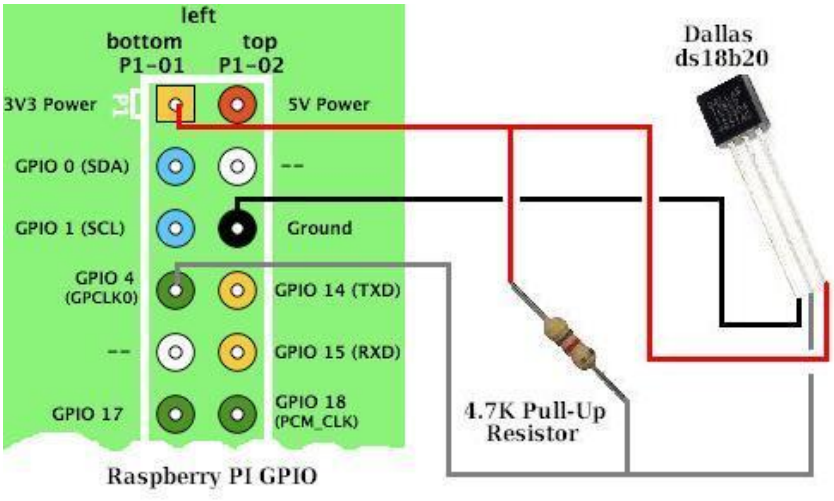


Рисунок 5.7 – Підключення сенсора температури до Raspberry Pi

Спочатку дозволимо інтерфейс 1-Wire в меню Налаштування -> Інтерфейси, щоб RPi зміг приймати дані від сенсора.

Далі, увійдемо на RPi і в командному рядку виконаємо такі команди:

```
sudo modprobe w1-gpio
sudo modprobe w1_therm
```

щоб завантажити модулі ядра однопровідних пристроїв зв'язку, необхідні для використання сенсора температури.

(Щоб уникнути необхідності ручного введення цих команд щоразу, коли повторно завантажуємо RPi, відредагуємо з використанням `nano` або подібного редактора файл "`/etc/modules`" додаванням рядків `w1-gpio` і `w1_therm` у кінець файлу.)

Тепер в командному рядку вводимо:

```
ls -l /sys/bus/w1/devices/
```

щоб побачити список пристроїв, підключених в даний момент до RPi.

Наш сенсор температури з'явиться з адресою у форматі 28-00000xxxxxx. На рис.5.8 можемо побачити, що адреса нашого сенсора температури була 28-00000283c6cd. Кожен сенсор температури DS18B20 має унікальну жорстко задану адресу, так що при необхідності можемо підключити кілька сенсорів температури і, як і раніше, читати їх окремо.

Рисунок 5.8 – Адреса сенсора та отримане значення температури

Знайдемо свій сенсор температури і введемо наступну команду в терміналі, щоб відкрити файл сенсора і переглянути показання температури:

```
cat /sys/bus/w1/devices/28-00000283c6cd/w1_slave
```

(переконавшись, звичайно, що підставили адресу свого власного сенсора температури).

На рис.5.8 бачимо на виході наведеної вище команди два рядки даних. Наприкінці другого рядку після "t=" виведене числове значення. Це значення є температурою сенсора в градусах Цельсія, помноженою на 1000. Тому, в нашому прикладі, ми бачимо показання t=24812 і температура буде 24812/1000 = 24,812 градусів за Цельсієм.

Читати температуру за допомогою командного рядка не дуже зручно. Загалом ми хочемо вимірювати температуру через регулярні проміжки часу і або відображати її десь, або зберігати в файлі для подальшого аналізу. Для цього ми повинні написати сценарій. Простий сценарій на Python для вимірювання та відображення температури один раз в секунду може бути таким (рис.5.9):

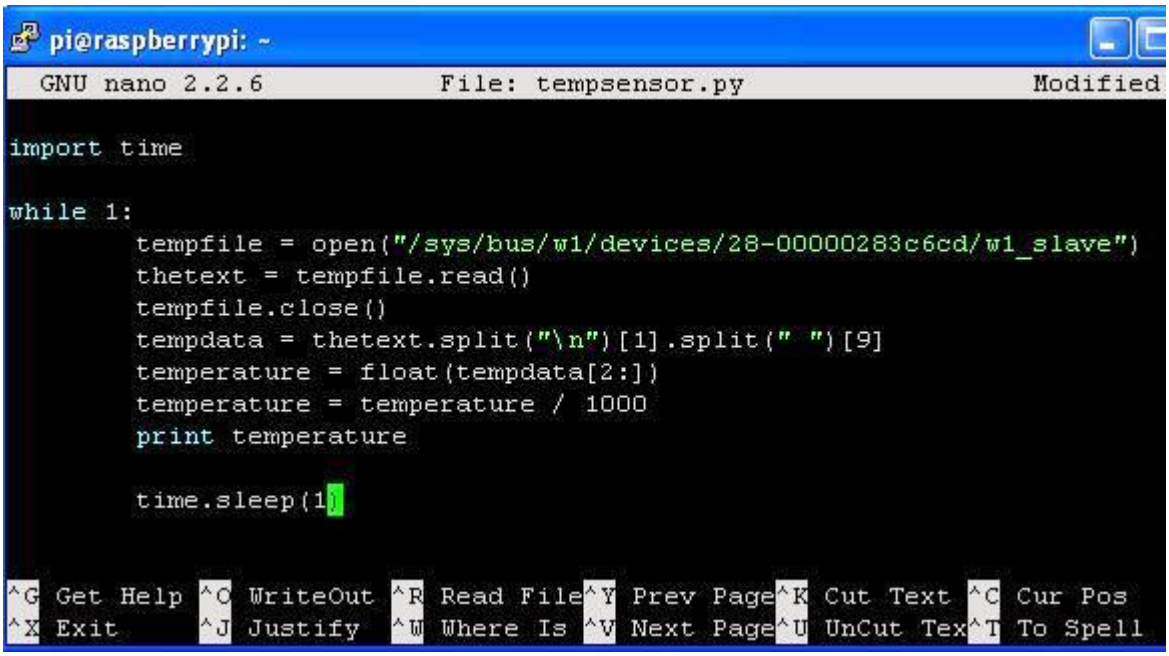


Рисунок 5.9 – Сценарій на Python для отримання температури з сенсора

Цей сценарій один раз на секунду читає файл сенсора температури, отримує другий рядок, розбиває його на розділені пробілами елементи, отримує десятий з цих елементів (деномінованих '9', бо '0' є першим елементом),

пропускає перші два символи ("t=") цього елемента і перетворює те, що залишається (24812 в нашому прикладі), з рядка в числове значення, яке потім ділиться на 1000, щоб показати нам температуру, яка виводиться на екран (рис.5.10).

Рисунок 5.10 – Виведені на екран значення температури

На рис.5.10 показаний запуск сценарію поки сенсор температури проходить нагрівання.

Сценарій Python для читання даних з сенсора

Створимо новий сценарій Python temperature.py у вікні терміналу (наприклад, з nano) або з IDLE (Interactive Development Environment).

Першим кроком буде імпорт необхідних модулів: `os` дозволяє нам включити драйвери 1-Wire та інтерфейс з нашим сенсором, а `time` дозволяє RPi визначити час та використовувати часові періоди в нашому коді.

Введемо наступний код:

```
# Імпорт бібліотек
import os
import time
# Ініціалізація виводів GPIO
os.system('modprobe w1-gpio') # Вмикання модуля GPIO
os.system('modprobe w1-therm') # Вмикання модуля Temperature
```

```
    # Отримання device file (w1_slave file), який зберігає дані
температури
    temp_sensor = ‘sys/bus/w1/devices/28-000005e2fdc3/w1_slave
    # Функція, яка читає дані з сенсора
    def temp_raw():
        f = open(temp_sensor, 'r') # Відкриття значення
температури в device file
        lines = f.readlines() # Повернення тексту
        f.close()
        return lines

    # Перетворення значення сенсора в температуру
    def read_temp():
        lines = temp_raw() # Читання температури 'device file'

        # Якщо перший рядок не має 'YES', чекати 0.2 с
        # і потім прочитати device file знову.
        while lines[0].strip()[-3:] != 'YES':
            time.sleep(0.2)
        lines = temp_raw()

        # Знаходження позиції '=' в другому рядку
        # device file.
        temp_output = lines[1].find('t=')

        # Якщо '=' знайдено, то перетворення решти рядка після
        # '=' в градуси Цельсія, а потім в градуси Фаренгейта
        if temp_output != -1:
            temp_string = lines[1][temp_output +2:]
            temp_c = float(temp_string) / 1000.0
            temp_f = temp_c * 9.0 / 5.0 + 32.0
            return temp_c, temp_f

    # Виведення температури, поки не зупинимо програму.
```

```
while True:
    print(read_temp())
    time.sleep(1)
```

Код необхідно запустити з правами Super User:

```
sudo python temperature.py
```

Швидке відображення температури в командному рядку – Bash

Замість того, щоб писати сценарій Python для читання температури з сенсора, можна написати один рядок команди, яка буде читати сенсор, витягувати температуру з частини повернутих даних і обробляти результат за допомогою `Bash` в командному рядку RPi.

Є багато способів для досягнення цієї мети, наведемо один приклад з використанням `sed` і `awk` (рис.5.11):

```
cat /sys/bus/w1/devices/28-00000283c6cd/w1_slave | sed -n
's/^.*\(t=[^ ]*\).*/\1/p' | sed 's/t=//' | awk
'{x=$1}END{print(x/1000)}'
```

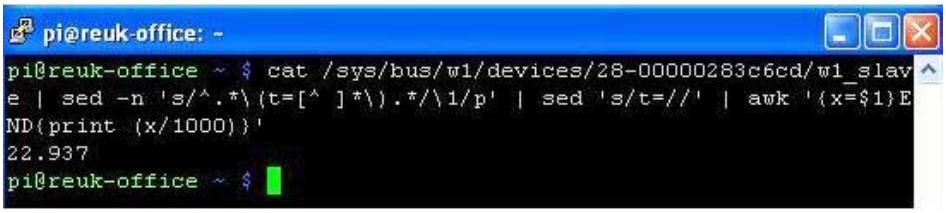


Рисунок 5.11 – Сценарій для читання даних з сенсора температури

Символ труби "`|`" використовується для передачі результату виведення однієї команди до наступної команди. Цей скрипт спочатку читає температуру з сенсора, перш ніж повернути два рядки даних, остання частина другого рядка, яка є `t=XXXXX`, де `XXXXX` - температура в мілліградусах Цельсія. Потім ми використовуємо `sed`, щоб знайти «слово», яке починається з 't=' в даних, далі пересилаємо його до другої команди `sed`, яка видаляє 't='. Нарешті, чисельний результат пересилається до команди `awk`, яка ділить цей результат на 1000 і відображає температуру в градусах Цельсія.

Очевидно, що ми не хотіли б набирати цей довгий рядок щоразу, коли перевіряємо температуру свого сенсора, тому можемо створити псевдонім:

```
alias checktemp="cat /sys/bus/w1/devi.....x/1000)}"
```

Тепер при введенні команди `checktemp` в командному рядку, будуть запущені всі збережені команди.

Альтернативно, можемо зберегти команди як сценарій оболонки. Для цього відкриємо текстовий редактор за допомогою команди:

```
sudo nano checktemp
```

який створить порожній файл з ім'ям `checktemp`. Введемо:

```
#!/bin/bash
```

в першому рядку, а потім в другому рядку вводимо команди `cat /sys/bus...` тощо. Натиснемо Ctrl-X, щоб вийти і зберегти файл. Щоб зробити даний файл виконуваним для можливості його запуску, вводимо:

```
chmod +x checktemp
```

і, нарешті, перемістимо його в каталог `/usr/bin`, де зберігається більшість виконуваних файлів:

```
mv checktemp /usr/bin/
```

Тепер, коли в командному рядку набираємо `checktemp`, то сценарій буде запускатися наш і буде відображатися прочитана з сенсора температура.

Підключення декількох сенсорів температури до Raspberry Pi

Розглянемо, як можна під’єднати кілька сенсорів температури до RPi, що особливо корисне для деяких додатків вимірювання різниці температур, де потрібні два або більше сенсорів температури.

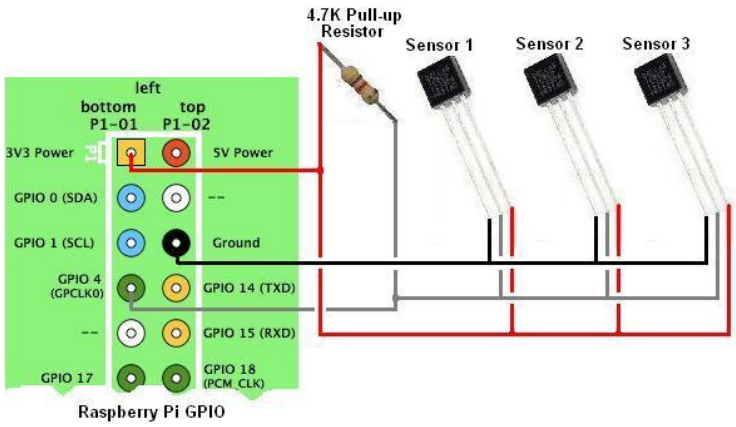


Рисунок 5.12 – Підключення кількох сенсорів до Raspberry Pi

На рис.5.12 схематично показано, як підключити три температурних сенсори DS18B20 до GPIO виводів RPi. Відзначимо, що хоча ми використовуємо декілька температурних сенсорів для читання, ми маємо три з'єднання з RPi: 3,3В і 0В підключення живлення сенсорів, і один провід даних, який повертає прочитану температуру від всіх сенсорів.

Прочитати показання температури з декількох сенсорів через один провід дозволяє серійний номер сенсора, закодований в ньому при виготовленні, який RPi може використати для ідентифікації.

При використанні декількох сенсорів, перше, що треба зробити, це отримати серійний номер для кожного сенсора і фізично маркувати їх, щоб знати, де який сенсор, коли будемо встановлювати їх у різних місцях комлексу.

Дотримуємося інструкцій, наведених вище, про підключення одного сенсора, щоб встановити кожен з них на своєму RPi, і скористаємося командою:

```
ls -l sys/bus/w1/devices
```

щоб знайти серійний номер для першого сенсора.

Потім додамо другий сенсор і визначимо його, повторно перевіривши підключені пристрої та відзначимо його серійний номер. Повторимо цей процес, поки не підключимо і не визначимо всі сенсори.

Щоб підтвердити, що все працює як треба, отримаємо температуру з кожного сенсора один за одним за допомогою команди:

```
cat /sys/bus/w1/devices/28-00000xxxxxxx/w1_slave
```

замінивши `XXXXXXX` правильним серійним номером для кожного сенсора.

Тепер ми готові використовувати сенсори температури у своєму проєкті на RPi. Подібним чином можна використовувати з RPi й інші цифрові сенсори з інтерфейсом 1-Wire.

5.2. Завдання

1. Вивчити особливості використання бібліотеки `OS` (знайти опис в мережі).

2. Виконати підключення сенсора температури, запрограмувати та отримати значення температури.

5.3. Зміст звіту

1. Опис роботи бібліотеки `os`

2. Схема підключення сенсора температури до мікрокомп'ютера.

3. Скріншоти результатів виконання роботи за допомогою доступу через Інтернет.

4. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

5.4. Контрольні питання

1. Для чого потрібний «підтягуючий резистор при підклчені сенсорів з інтефейсом 1-Wire?

2. Для чого використовується команда `sed`? Які параметри вона має

3. Для чого використовується команда `awk`? Які параметри вона має?

Лабораторна робота 6
ВИКОРИСТАННЯ Н-МОСТА ДЛЯ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ

Мета роботи: Вивчити варіанти застосування Н-моста для керування швидкістю та напрямком обертання двигуном постійного струму.

Зміст. Розглядається схема керування двигуном на основі мікросхем, які реалізують Н-міст, та пропонується сценарій на Python для створення інтерфейсу користувача для такого керування двигуном.

6.1. Теоретичні відомості

Простою у використанні інтегральною схемою (IC) Н-моста, яка користується популярністю, є L293D. Це мікросхема призначена для невеликих двигунів з максимальним струмом 600 мА і напругою до 36 В.

L293D містить два Н-мости і деякі додаткові схеми автоматичного вимкнення IC, якщо вона починає перегріватися. Але не варто експериментувати з L293D, хоча її важко спалити. Розміщення виводів L293D наведено на рис.6.1:

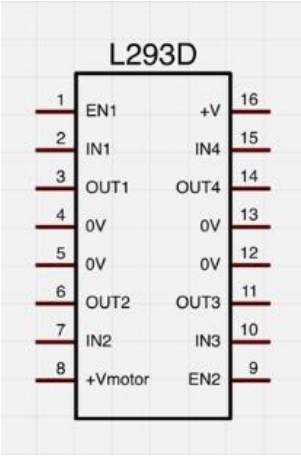


Рисунок 6.1 – Виводи мікросхеми L293D

Параметри L293D:

• Діапазон напруги живлення двигуна від 4,5В до 36В.

• Постійний струм двигуна до 600мА.

• Піковий струм двигуна 1,2А.

• Діоди на всіх виходах для захисту від стрибків значення напруги на виводах двигуна.

• Наявність теплового захисту.

• Сумісність з 3В і 5В логікою (RPi і Arduino).

ІС організована у вигляді чотирьох половин Н-мостів, а не двох повних Н-мостів. Можна розглядати кожну половину Н-моста як цифровий вихід високої потужності, здатний забезпечити стік або витік струму до 600мА. Це забезпечує більшу гнучкість при використанні ІС.

ІС має окремі контакти для живлення логіки і двигуна, що дозволяє, наприклад, управляти 6В двигунами з використанням 3,3В логіки RPi.

Функція кожного з виводів ІС L293D наведена в таблиці 6.1.

Таблиця 6.1 – Призначення виводів L293D

Номер виводу	Назва виводу	Опис
1	1,2EN	Вивід дозволяє виходи напівмостів 1 і 2 (тобто, якщо на цьому контакті високий рівень, виходи не робитимуть нічого); це часто використовується з ШІМ-сигналом для управління швидкістю двигуна
2	1A	Вхід керування напівмостом 1; якщо це високий рівень, то вихід на виводі 3 буде високим
3	1Y	Вихід напівмоста 1
4,5, 12, 13	GND	Земля (на друкованій платі всі ці виводи припаяні до великого майданчика, який виступає як поглинач тепла)
6	2Y	Вихід напівмоста 2
7	2A	Вхід керування напівмостом 2
8	VCC2	Напруга живлення для двигунів, аж до 36В; роздільне живлення двигуна і логіки допомагає зберегти стабільність
9	3,4EN	Дозвіл напівмостів 3 і 4
10	3A	Вхід керування напівмостом 3
11	3Y	Вихід напівмоста 3
14	4Y	Вихід напівмоста 4
15	4A	Вхід керування напівмостом 4
16	VCC1	Напруга живлення для логіки; вона може бути нижчою, ніж напруга живлення двигуна на виводі 8 і зазвичай рівна 5В

Схема підключення Н-моста на L293D в проєкті показана на рис.6.2:

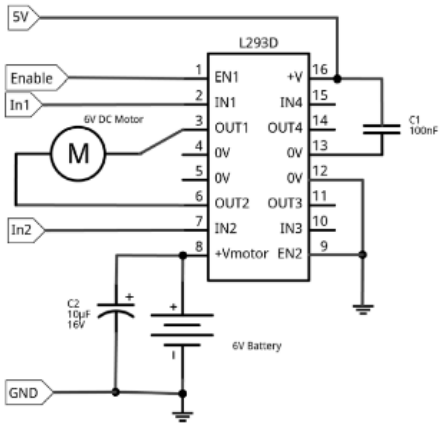


Рисунок 6.2 – Схема підключення мікросхеми L293D для керування двигуном

RPi забезпечує 5В для живлення логіки через контакт 16, а живлення для двигуна 6В подається на контакт 8 з акумуляторної батареї. Фактично використовується тільки один з H-мостів в IC і тому вивід EN2 з'єднаний з землею, щоб відключити невикористану половину IC.

Виводи EN1, IN1 і IN2 будуть підключені до цифрових вихідних контактів на RPi.

Схема буде працювати і без конденсаторів C1 та C2, але буде хорошою звичкою використовувати конденсатори, щоб проєкт був розгорнутий по-справжньому.

Вказане розташування конденсаторів дуже типове для IC схеми H-моста. C1 називають роздільним конденсатором. Він повинен розташовуватися якомога ближче до IC і бути між джерелом живлення логіки і GND. Досить 100 нФ (конденсатор невеликої ємності), який усуває будь-які електричні завади, що можуть створювати перешкоди в роботі логіки IC. C2 забезпечує короткочасний запас енергії, але для двигунів, а не для логіки перемикання. Значення цього конденсатора, зазвичай, значно більше, ніж C1: 100 мкФ або більше.

На монтажній платі звернемо особливу увагу на IC. Переконаємося, що розташували мікросхему правильно. Можемо також підключити батарею. Спочатку двигун не повинен обертатися.

Перевагою використання такої IC H-моста, як L293D, є те, що виводи керування вимагають дуже мало струму для керування двигуном. Дійсно,

технічний опис IC вказує, що це завжди менше, ніж 100мкА (0,1мА), що означає відсутність будь-яких проблем з використанням слабкострумових виходів RPi.

Підключення макетної плати до RPi (рис.6.3):

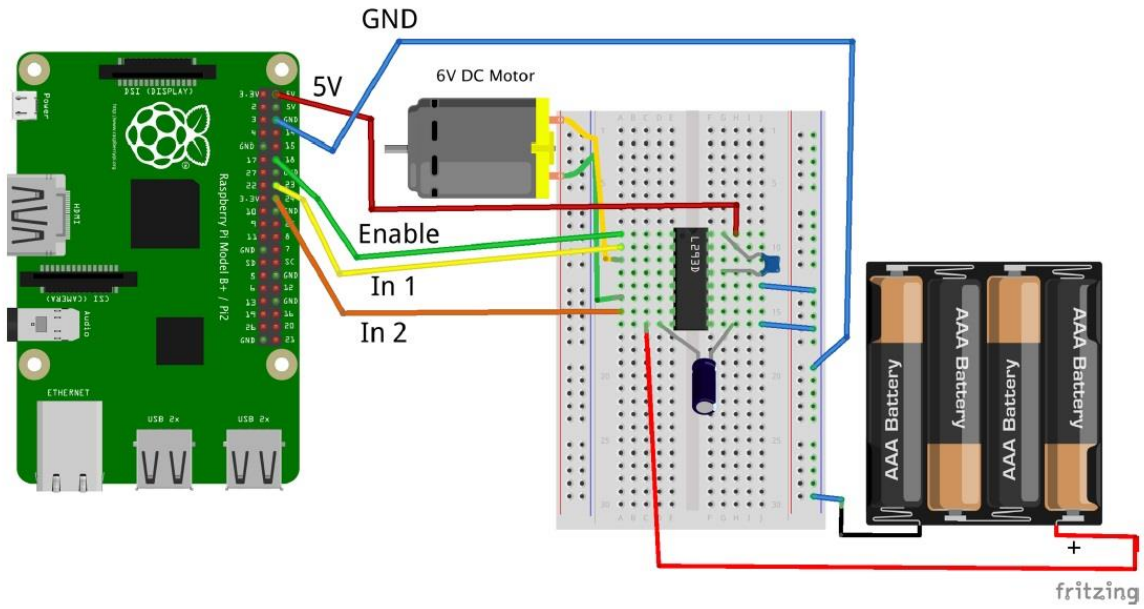


Рисунок 6.3 – Підключення макетної плати до Raspberry Pi

Доповнимо та збережемо у файлі `full_motor_control.py` сценарій на Python для RPi, щоб можна було змінювати напрям обертання двигуна, швидкість обертання та мати змогу його зупинити:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
enable_pin = 18                    #1
in_1_pin = 23
in_2_pin = 24
GPIO.setup(enable_pin, GPIO.OUT)
GPIO.setup(in_1_pin, GPIO.OUT)
GPIO.setup(in_2_pin, GPIO.OUT)
motor_pwm = GPIO.PWM(enable_pin, 500)
motor_pwm.start(0)
```

```
    def forward(duty):              #2
        GPIO.output(in_1_pin, True)
        GPIO.output(in_2_pin, False)
        motor_pwm.ChangeDutyCycle(duty)
    def reverse(duty):              #3
        GPIO.output(in_1_pin, False)
        GPIO.output(in_2_pin, True)
        motor_pwm.ChangeDutyCycle(duty)
    def stop():
        GPIO.output(in_1_pin, False)
        GPIO.output(in_2_pin, False)
        motor_pwm.ChangeDutyCycle(0)
    try:
        while True:                 #4
            direction = input('Enter direction letter (f -
forward, r - reverse, s - stop): ')
            if direction[0] == 's':
                stop()
            else:
                duty = int(input('Enter Duty Cycle (0 to 100):
'))
            if direction[0] == 'f':
                forward(duty)
            elif direction[0] == 'r':
                reverse(duty)
    finally:
        print("Cleaning up")
        GPIO.cleanup()
```

Коментарі:

1. У верхній частині файлу звичайний код налаштування GPIO і призначення виводів. Вивід **Enable** L293D використовується для управління швидкістю обертання двигуна, тому вивід 18 підключений до нього і налаштований як вихід ШІМ.

2. Функція **forward** встановлює контакти IN1 і IN2 для керування напрямком обертання, а потім задає шпаруватість каналу ШІМ.

3. Якщо порівняти функцію **reverse** з **forward**, то можемо побачити, що значення контактів IN1 і IN2 помінялися місцями. Функція **stop** задає на виводах напрямку значення для зупинки (обидва LOW) і робочий цикл ШІМ рівний 0.

4. Команда запитує у користувача значення шпаруватості, а потім викликає **stop**, **forward**, або **reverse**, в залежності від введеного.

Запустимо програму `full_motor_control.py`, і переконаємося, що можемо змінювати напрямок обертання та швидкість:

```
sudo python3 full_motor_control.py
Enter direction letter (f - forward, r - reverse, s - stop): f
Enter Duty Cycle (0 to 100): 50
Enter direction letter (f - forward, r - reverse, s - stop): f
Enter Duty Cycle (0 to 100): 100
Enter direction letter (f - forward, r - reverse, s - stop): s
Enter direction letter (f - forward, r - reverse, s - stop): r
Enter Duty Cycle (0 to 100): 50
Enter direction letter (f - forward, r - reverse, s - stop): r
Enter Duty Cycle (0 to 100): 100
Enter direction letter (f - forward, r - reverse, s - stop): s
Enter direction letter (f - forward, r - reverse, s - stop):
```

Приклад сценарію для реалізації графічного інтерфейсу керування двигуном:

```
from tkinter import *
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
enable_pin = 18
```

```
    in_1_pin = 23
    in_2_pin =.24
    GPIO.setup(enable_pin, GPIO.OUT)
    GPIO.setup(in_1_pin, GPIO.OUT)
    GPIO.setup(in_2_pin, GPIO.OUT)
    motor_pwm = GPIO.PWM(enable_pin, 500)
    motor_pwm.start(0)

    class App:

        def __init__(self, master):
            frame = Frame(master, bg = 'purple')
            frame.pack()
            Label(frame, text='velocity', bg = 'purple', fg =
'white').grid(row=0, column=0)
            scaleVel = Scale(frame, from_=0, to=100, length=250,
orient=HORIZONTAL, command=self.motorduty, bg = 'purple', fg =
'white')
            scaleVel.grid(row=0, column=1)
            Button(root, text = 'reverse', command=self.reverse,
bg = 'purple', fg = 'white').place(x = 10, y = 60, width = 130,
height = 80)
            Button(root, text = 'forward', command=self.forward,
bg = 'purple', fg = 'white').place(x = 160, y = 60, width = 130,
height = 80)
            Button(root, text = 'stop', command=self.stop, bg =
'purple', fg = 'white').place(x = 20, y = 155, width = 260, height
= 90)
        def motorduty(self,duty):
            motor_pwm.ChangeDutyCycle(float(duty))
            print(duty)

        def forward(self):
            GPIO.output(in_1_pin, True)
            GPIO.output(in_2_pin, False)
```

```
    def reverse(self):
        GPIO.output(in_1_pin, False)
        GPIO.output(in_2_pin, True)
    def stop(self):
        GPIO.output(in_1_pin, False)
        GPIO.output(in_2_pin, False)
        motor_pwm.ChangeDutyCycle(0)

root = Tk()
root.wm_title('Motor Control')
app = App(root)
root.geometry("300x250+400+250")
frame = Frame(root)
frame.pack()

try:
    root.mainloop()

finally:
    print("Cleaning up")
    GPIO.cleanup()
```

Результат виконання сценарію показаний на рис. 6.4.

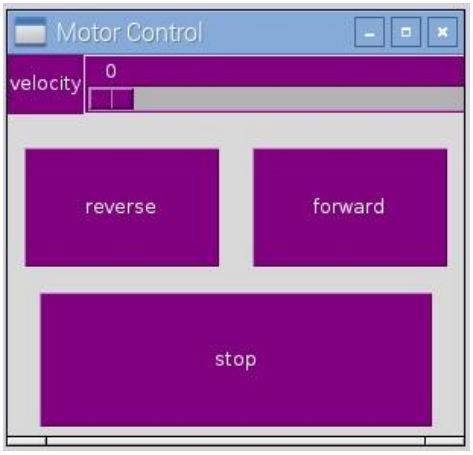


Рисунок 6.4 – Графічний інтерфейс користувача для керування двигуном

6.2. Завдання

1. Додайте в сценарій проєкту керування швидкістю і напрямком обертання двигуна графічний інтерфейс користувача, використовуючи наведений нижче код.

2. Інтерфейс користувача виконати у вигляді «хрестика»: в центрі червона кнопка ***Стоп***, а решта 4 кнопки – ***Вперед***, ***Назад***, ***Вліво***, ***Праворуч***.

3. Реалізувати шкалу для керування швидкістю руху як в наведеному прикладі.

4. Колір інтерфейсу вибрати інший, ніж в прикладі на рис.6.4.

Примітка. Кнопки ***Вліво*** і ***Праворуч*** не програмувати – це вже для керування двома двигунами.

6.3. Зміст звіту

1. Схема підключення двигуна до мікросхеми Н-моста та мікрокомп’ютера.

2. Сценарій на Python з коментарями.

3. Скріншот результатів виконання сценарію.

4. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

6.4. Контрольні питання

1. Як за допомогою вивчених раніше віджетів реалізувати керування швидкістю двигуна не у вигляді горизонтального повзунка? Запропонуйте варіанти.

2. Чому для керування двигуном використовують ШІМ, а не просте підключення його до виводів RPi?

3. Яке рішення можете запропонувати, якщо вихідного рівня напруги на виводах RPi буде недостатньо для керування мікросхемою Н-моста?

Лабораторна робота 7
ДОСЛІДЖЕННЯ КЕРУВАННЯ СЕРВОПРИВОДОМ

Мета роботи: Дослідити методи керування сервоприводом (серводвигуном) за допомогою сформованої ШІМ

Зміст. Розглядаються різні методи формування ШІМ та вивчаються приклади сценаріїв на Python для керування сервоприводом.

7.1. Теоретичні відомості

Підключення сервоприводу, зазвичай, здійснюється за допомогою шлейфа з трьох проводів (рис.7.1):

червоний – живлення – підключається до контакта 5В (вивід 2 RPi – для малопотужних сервоприводів) або безпосередньо до джерела живлення;

коричневий або **чорний** – земля GND (вивід 6 RPi);

жовтий або **білий** – сигнал; підключається до виводу 11 RPi (GPIO17).

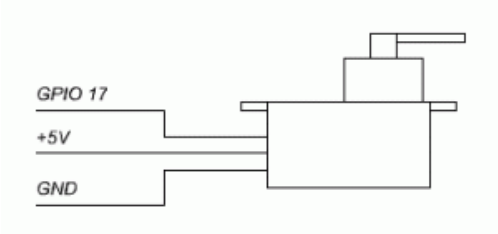


Рисунок 7.1 – Виводи сервоприводу

Нагадаємо призначення виводів GPIO (рис.7.2):

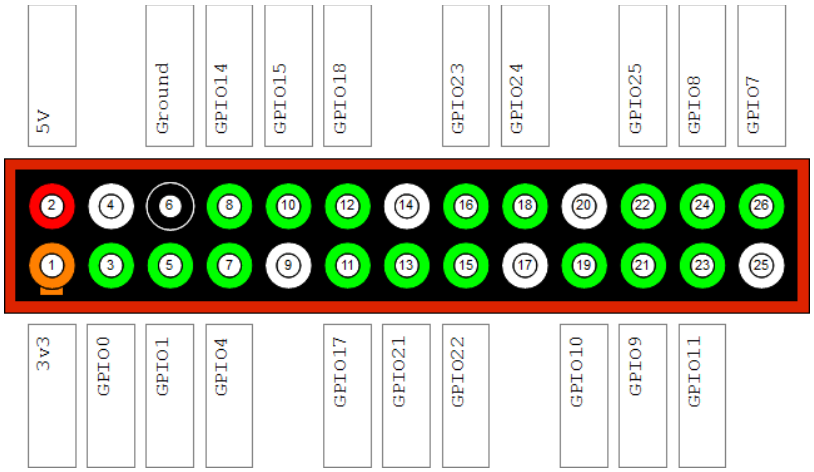


Рисунок 7.2 – Призначення виводів GPIO

Малопотужний сервопривід можна живити від RPi. Але, якщо привід споживає досить великий струм, або підключаємо кілька сервоприводів, то використовуємо окреме джерело живлення.

Для встановлення сервоприводу в нейтральне положення необхідно подати сигнал високого рівня тривалістю 1,5мс, в 0 градусів – 0,5мс, в 180 градусів – 2,5мс. Подаючи на сервопривід імпульси різної довжини, ми можемо змінювати його положення.

7.1. Керування сервоприводом за допомогою програмної ШІМ

Приклад 1 – Програмне формування ШІМ

Для програмного керування сервоприводом напишемо сценарій servo.py, код якого наведений у прикладі нижче

Спочатку створимо файл servo.py:

```
nano servo.py
```

Код на Python:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.OUT)
p=GPIO.PWM(17,50)
p.start(7.5)
try:
    while True:
        p.ChangeDutyCycle(7.5)
        print("Left")
        time.sleep(1)
        p.ChangeDutyCycle(12.5)
        print("Center")
        time.sleep(1)
        p.ChangeDutyCycle(2.5)
```

```
        print("Right")
        time.sleep(1)
except KeyboardInterrupt:
    p.stop()
    GPIO.cleanup()
```

Запустимо servo.py:

```
python3 ./servo.py
```

Функції `print` спеціально вставлено в код. Наявність цих функцій створює проблему нестабільності програмно сформованого ШІМ. Сервопривід не фіксується в заданому положенні, а смикається. Якщо видалити функцію `print`, то проблема зменшиться або навіть зовсім зникне.

7.2. Керування сервоприводом за допомогою ШІМ, сформованого через DMA

RPIO.PWM забезпечує ШІМ через DMA (direct memory access – прямий доступ до пам’яті) для RPi, використовуючи вбудований модуль для напів апаратної ШІМ з точністю до 1 мкс.

З RPIO.PWM ми можемо використовувати будь-який з 15 DMA-каналів і будь-яку кількість виводів GPIO для кожного каналу. Оскільки ШІМ здійснюється через DMA, то RPIO.PWM практично не використовує ресурси центрального процесора і може генерувати стабільні імпульси з дуже високою роздільною здатністю. RPIO.PWM реалізований на C, але його можна використовувати на Python за допомогою наданої обгортки.

RPIO.PWM надає методи низького рівня для ручного контролю всім, а також допоміжні класи, які спрощують ШІМ для певних використань, наприклад, RPIO.PWM.Servo. Для цього модуля, поки що, підтримується тільки нумерація BCM GPIO.

Встановлюємо RPIO, якщо не встановлено:

Спочатку запускаємо команди:

```
sudo apt install python-dev
sudo apt install python-setuptools
sudo easy_install -U RPIO
```

Також можемо отримати RPIO з репозиторію Github:

```
git clone https://github.com/metachris/RPIO.git
cd RPIO
sudo python setup.py install
```

Або з Github, але без Git:

```
curl -L https://github.com/metachris/RPIO/archive/master.tar.gz |
tar -xz
cd RPIO-master
sudo python setup.py install
```

Приклад 2 – Використання PWM.Servo

Час субциклу (періоду) за замовчуванням 20 мс і ширина імпульсу за замовчуванням збільшується з кроком 10 мкс:

```
from RPIO import PWM
servo = PWM.Servo()

# Встановити серво на GPIO17 до 1200 µs (1.2 ms)
servo.set_servo(17, 1200)

# Встановити серво на GPIO17 до 2000 µs (2 ms)
servo.set_servo(17, 2000)

# Зупинити серво на GPIO17
servo.stop_servo(17)
```

Приклад 3 - Використання низькорівневих методів ШІМ

```
from RPIO import PWM
# Встановлення PWM і каналу DMA 0
PWM.setup()
PWM.init_channel(0)
# Додавання деяких імпульсів до субциклу
PWM.add_channel_pulse(0, 17, 0, 50)
PWM.add_channel_pulse(0, 17, 100, 50)

# Зупинка PWM для вказаного GPIO на каналі 0
PWM.clear_channel_gpio(0, 17)

# Зупинка всіх PWM і активності DMA
PWM.cleanup()
```

Приклад 4 – Програма керування серводвигуном через DMA

Напишемо сценарій, спочатку створивши файл servo_dma.py:

```
nano servo_dma.py
```

Текст сценарію:

```
import time
from RPIO import PWM
servo = PWM.Servo()
# Встановити servo на GPIO17 до 900 мкс (0.9 мс)
servo.set_servo(17, 900)
# Встановити servo на GPIO17 до 2000 мкс (2.0 мс)
#servo.set_servo(17, 2000)
try:
    while True:
        servo.set_servo(17, 750)
        print("Left")
        time.sleep(1)
```

```
        servo.set_servo(17, 1500)
        print("Center")
        time.sleep(1)
        print("Right")
        servo.set_servo(17, 2500)
        time.sleep(1)
except KeyboardInterrupt:
    # Clear servo on GPIO17
    servo.stop_servo(17)
```

Запустимо сценарій:

```
python ./servo_dma.py
```

Теперь сервопривід повинен працювати стабільно. Таким чином, для керування сервоприводами програмну реалізацію ШІМ бажано не використовувати.

7.2. Завдання

1. Розробіть сценарій для керування кутовими положеннями (повороту) сервоприводу відповідно до таблиці варіантів з кутами повороту в градусах.

Таблиця 7.1 – Варіанти завдань для кутів повороту сервоприводу

Варіант	Кут 1, град.	Кут 2, град.	Кут 3, град.	Кут 4, град.	Кут 5, град.
1	45	0	135	180	90
2	90	45	180	0	135
3	135	90	0	45	180
4	180	135	45	90	90
5	45	180	90	135	0
6	90	0	135	180	45
7	135	45	180	0	90
8	180	90	0	45	135
9	90	45	45	90	180
10	0	0	80	180	0

7.3. Зміст звіту

1. Сценарій на Python для керування сервоприводом з коментарями.

2. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

7.4. Контрольні питання

1. Які переваги сервоприводів при використанні їх у виконавчих механізмах комплексів?

2. Чому для керування сервоприводом ми не використовуємо додаткових компонент, наприклад, Н-моста?

Лабораторна робота 8
ВИКОРИСТАННЯ СЕНСОРІВ З ІНТЕРФЕЙСОМ I2C

Мета роботи: Навчитися використовувати обмін даними з мікро комп'ютером через інтерфейс I2C (Inter Integrated Circuit), який створювався як простий інтерфейс з мінімально кількістю ліній зв'язку і є досить поширеним.

Зміст. Розглядається використання інтерфейсу I2C, який використовує всього дві лінії для зв'язку ведучого пристрою з веденим: двонаправлену лінію даних SDA (Serial DAta) та лінію тактування SCL (Serial CLock) для синхронізації прийому і передачі даних.

8.1. Особливості застосування інтерфейсу I2C

Основний режим роботи інтерфейсу I2C — 100 кбіт/с і 10 кбіт/с в режимі роботи із зниженою швидкістю. Після введення стандарту 1992 року стало можливим підключення більшої кількості пристроїв на одну шину (за рахунок 10-бітної адресації), а також зросла швидкість обміну до 400 кбіт/с у швидкісному режимі (реалізовано в RPi). Відповідно, доступна кількість вузлів зросла до 1008. Версія стандарту 2.0, випущена в 1998 році, представила високошвидкісний режим роботи зі швидкістю до 3,4 Мбіт/с зі зниженим енергоспоживанням.

I2C використовується в таких додатках як, читання RTC (годинник реального часу), доступ до зовнішньої пам'яті EEPROM, в модулях таких сенсорів, як гіроскопи, магнітометри тощо.

Зв'язок пристроїв через інтерфейс I2C на рівні електричних сигналів здійснюється з урахуванням наступних 4 правил:

1. ВИСОКИЙ рівень на лінії даних SDA або на лінії тактування SCL не може бути встановлений безпосередньо пристроєм I2C. Всі пристрої I2C повинні мати виходи з відкритим колектором (або відкритим стоком). ВИСОКИЙ рівень напруги на лініях формується зовнішніми резисторами підтягування.

2. Інформація на лінії даних SDA зчитується лише при ВИСОКОМУ рівні на лінії тактування SCL.

3. Інформація на лінію даних SDA виставляється тільки при НИЗЬКОМУ рівні на лінії тактування SCL.

4. Якщо шина I2C в даний момент не використовується, то на лініях повинен бути встановлений ВИСОКИЙ рівень сигналу (логічна 1).

Основний формат кадру обміну через I2C складається з 11 біт. Спочатку посилається старт-біт, потім 8 біт даних, далі біт підтвердження прийому, потім стоп-біт. Форма подання старт і стоп-бітів порушує записане вище правило 3. Старт-біт визначається наявністю спадаючого фронту на лінії SDA при ВИСОКОМУ рівні сигналу на лінії SCL. Цю комбінацію сигналів прийнято також називати стартовим станом, або станом Start. Стоп-біт (або стан Stop) визначається наявністю наростаючого фронту на лінії SDA при ВИСОКОМУ рівні на лінії SCL. Стани Старт і Стоп генеруються ведучим пристроєм (рис. 8.1).

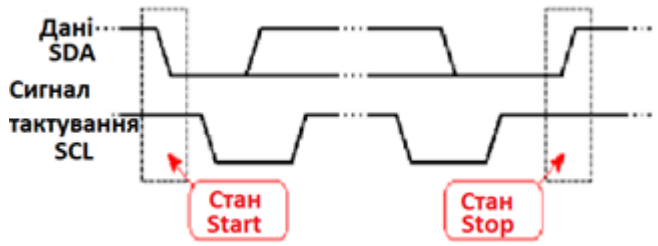


Рисунок 8.1 – Стани Start і Stop на шині I2C

Після формування стартового біта ведучий пристрій виставляє на лінію даних SDA 8 біт даних, починаючи зі старшого біта байта. Кожен біт супроводжується імпульсом синхронізації SCL. Потім приймач генерує біт підтвердження прийому ACK, встановлюючи лінію SDA в НИЗЬКИЙ стан на інтервалі дев'ятого імпульсу на лінії SCL.

Якщо приймач не впізнав прийняті дані, то генерується біт «непідтвердження» прийому NACK, просто не переводячи лінію SDA в НИЗЬКИЙ стан. В цьому випадку передавач закінчує поточний сеанс передачі

даних і генерує стоп-біт. Згідно алгоритму прикладної програми передавач спробує передати інформацію пізніше. На рис. 8.2 показані часові діаграми на лініях I2C, які супроводжують передачу та прийом байта 11010000 (передача успішна, біт підтвердження ACK сформований).

Рисунок 8.2 – Кадри передачі та приймання даних через інтерфейс I2C

Тривалість фронтів при перемиканні в реальних системах не настільки мала, як показано на рис.8.2, оскільки реальні лінії мають розподілену ємність і до них підключені підтягуючі резистори. Чим більші значення ємності і опору підтягуючих резисторів, тим більші тривалості фронтів сигналів. Великий час наростання і спаду сигналу може привести до помилок при обміні даними між пристроями

При проєктуванні шини I2C необхідно враховувати кількість пристроїв, що підключаються, величини їх вхідної ємності і опорів підтягуючих резисторів. Деякі виробники напівпровідникових приладів пропонують спеціальні конвертори для збільшення кількості пристроїв, що підключаються до шини I2C.

Примітка. Треба враховувати, що максимальна сумарна ємність підключених до лінії I2C пристроїв не повинна перевищувати 400 пФ.

Протокол I2C передбачає можливість примусового зниження швидкості передачі даних веденим пристроєм в процесі самої передачі. Коли лінія SCL знаходиться в НИЗЬКОМУ стані, ведений пристрій може примусово продовжити цей стан на лінії, поки він не буде готовий прийняти наступний біт даних. Ця функція називається «розширенням тактових імпульсів».

При необхідності взаємодії декількох пристроїв по шині I2C у кожного підключеного до шини пристрою повинна бути своя адреса. Відповідно до специфікації протоколу I2C адреса може бути 7-ми або 10-бітовою.

Структура кадру обміну через I2C при передачі 7-бітної адреси: старт-біт, 7 біт адреси, біт напрямку передачі для наступного кадру R/W, біт підтвердження ACK. Перші 9 біт, включаючи старт-біт, генеруються ведучим пристроєм, а біт підтвердження прийому – веденим пристроєм, чия адреса збігається з адресою, яку виставив на лінію ведучий:

Рисунок 8.3 – Структура кадру обміну через I2C

Зазвичай 7-бітова адреса складається з двох частин: перші 4 біта визначають тип веденого пристрою, а останні 3 – номер пристрою зазначеного типу в системі. У табл. 8.1 наведені деякі типи пристроїв та їх спеціальні зарезервовані адреси при організації зв'язку через протокол I2C.

Таблиця 8.1 – Зарезервовані адреси при використанні протоколу I2C

Адреса	Читання/ Запис	Опис
0000000	0	Адрес загального визову
0000000	1	Старт-байт (допомагаємо повільним веденим пристроям визначити початок сеансу передачі даних)
0000001	x	Адреса CBUS
0000010	x	Резерв
0000011	x	Резерв
0000lxx	x	Високошвидкісний ведучий (до 3.4 Мбіт/с).
0010xxx	x	Синтезатори мови
0011xxx	x	Аудіоінтерфейси
0100xxx	x	Генератори звукових частот
0111xxx	x	LED/LCD-дисплеї
1000xxx	x	Відеоінтерфейси
1001xxx	x	Цифро-аналогові і аналого-цифрові перетворювачі
1010xxx	x	Пам'ять
1100xxx	x	ВЧ синтезатори
1101xxx	x	Годинники/календарі
11101xx	x	Резерв
11111xx	x	Режим 10-бітної адреси

Процедура адресації відбувається наступним чином:

1. Ведучий пристрій генерує стан Старт.

2. Ведучий пристрій виставляє на лінію адресу веденого пристрою.

3. В залежності від значення біта R/W, ведучий пристрій посилає дані веденому пристрою (якщо R/W = 0) або зчитує дані, виставлені на лінію веденим пристроєм (R/W = 1).

4. Як тільки всі дані надіслані, ведучий пристрій генерує на лінії стан Stop.

У специфікації протоколу описаний ще один режим, який називається «Повторний старт». В цьому режимі ведучий пристрій встановлює ще раз стан Start і передає один байт адреси без генерації стану Stop. Це буває корисно для зміни напрямку передачі даних.

8.2. Використання інтерфейсу I2C на Raspberry Pi

Використання інтерфейсу I2C на RPi розглянемо на прикладі цифрового компасу GY-271 (HMC5883L).

Магнітометр GY-271 (рис.8.4) використовується для вимірювання напрямку та величини магнітного поля Землі в дешевих цифрових компасах та магнітометрах. Вимірює значення магнітного поля Землі вздовж осей X, Y та Z від мілігауса до 8 гаус. Допомагає знайти напрямок руху пристрою. Для з'єднання використовує протокол I2C.

Рисунок 8.4 – Магнітометр (HMC5883L)

RPi має виводи інтерфейсу I2C, які показані на рис.8.5:

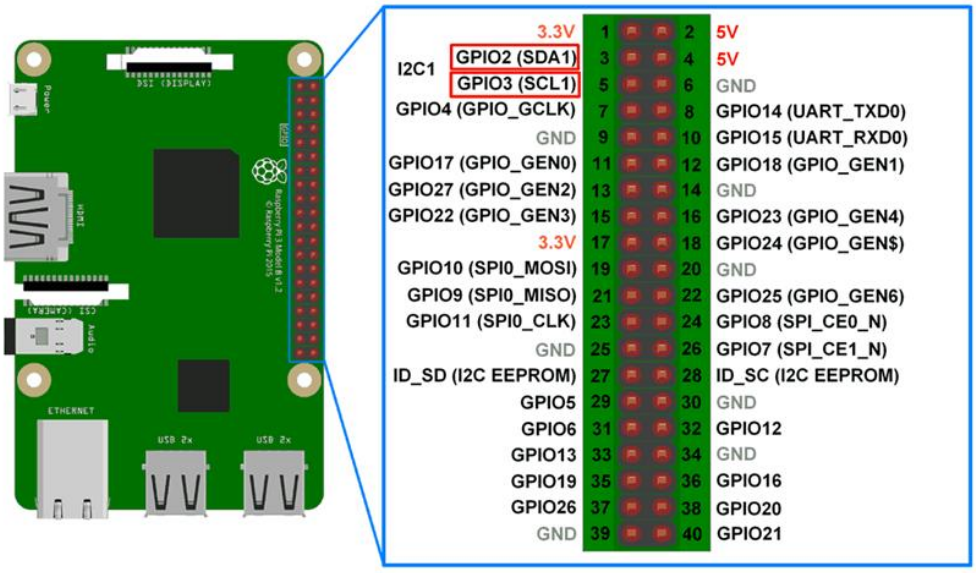


Рисунок 8.5 – Виводи інтерфейсу I2C на Raspberry Pi

Щоб отримати доступ до шини I2C в RPi, ми повинні виконати певні налаштування. Перш за все, треба дозволити інтерфейс I2C на RPi. Це можна зробити через меню, або використати термінал, ввівши команду:

```
sudo raspi-config
```

Вибираємо **Interfacing Configurations**, а потім **I2C**. Дозволяємо його і перезавантажуємо RPi.

Після перезавантаження RPi, перевіряємо, який режим використовує порт I2C, ввівши команду:

```
ls /dev/*i2c*
```

RPi повинен повернути назву порта i2c (рис.8.6):

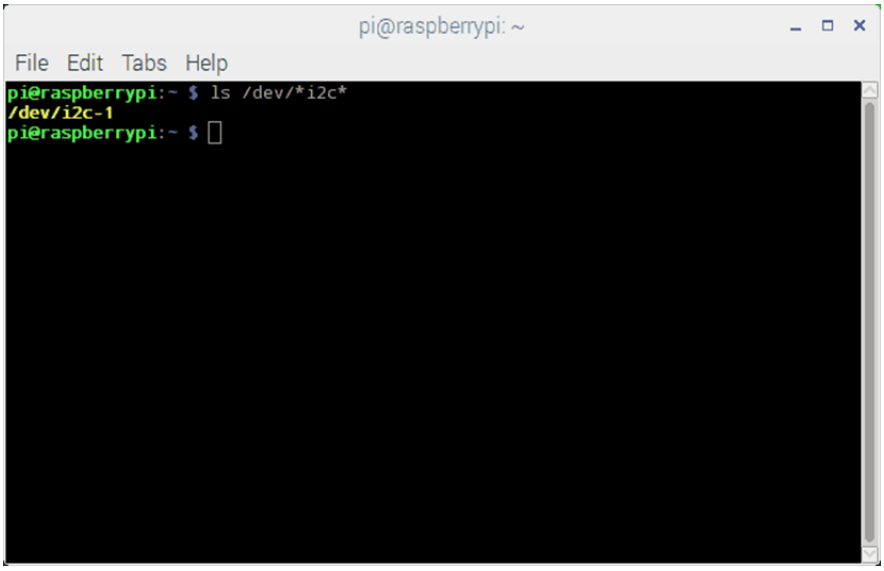


Рисунок 8.6 – Визначення режиму порта I2C

Вище показане використання інтерфейсу I2C в режимі i2c-1. Старі версії RPi можуть повертати режим i2c-0.

Тепер підключимо магнітометр до порта I2C (рис.8.7):

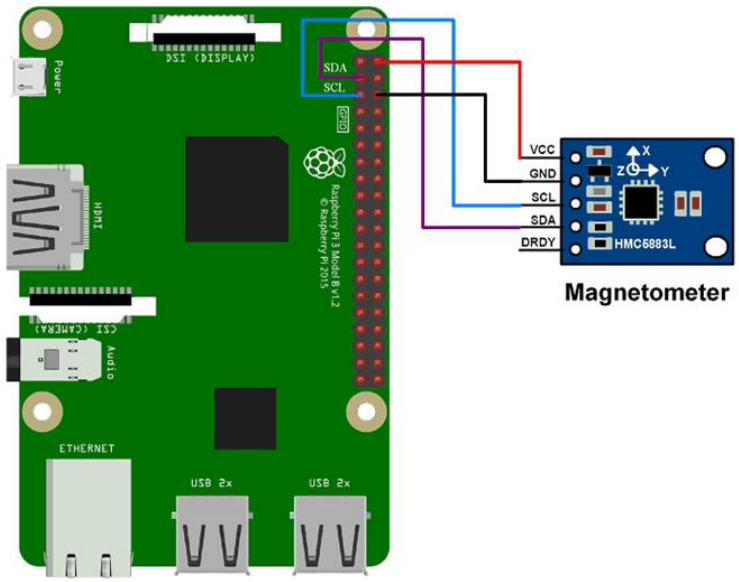


Рисунок 8.7 – Магнітометр з Raspberry Pi 3B

Далі, можемо протестувати/сканувати пристрої I2C, підключені до нашої плати RPi, попередньо встановивши `i2c-tools`. Ми можемо отримати `i2c-tools` за допомогою apt-менеджера пакетів. Використаємо таку команду в терміналі RPi:

```
sudo apt install -y i2c-tools
```

Скануємо порт I2C за допомогою команди:

```
sudo i2cdetect -y 1
```

У відповідь ми повинні отримати адресу пристрою. Наприклад, ми підключили GY-271 I2C до RPi і спробували виявити цей пристрій, як показано на рис.8.8:

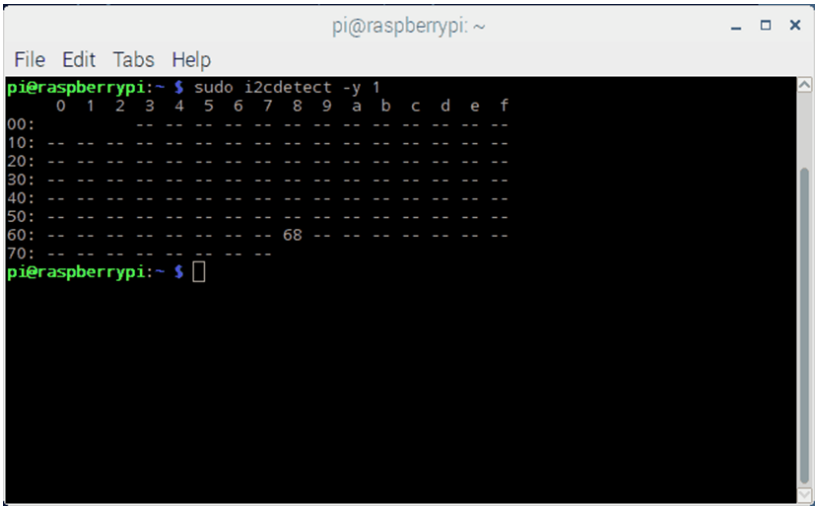


Рисунок 8.8 – Визначення адреси пристрою з інтерфейсом I2C

Команда `i2cdetect` сканує порт I2C, щоб отримати адресу пристрою, якщо він підключений. Якщо пристрій не підключений до порта I2C, то повернеться поле з (- -).

Ми можемо отримати або встановити дані пристрою I2C за допомогою команд `i2cget`, `i2cset`, тощо.

Наприклад,

```
sudo i2cget -y I2C_user_mode_Port address_of_device
Register_address
sudo i2cget -y 1 0x68 0x01
```

і отримаємо дані, що містяться в реєстрі з адресою 0x01.

Ми можемо отримати доступ до шини I2C на RPi, використовуючи SMBus. SMBus – це підмножина шини/інтерфейсу I2C. SMBus надає підтримку пристроям з інтерфейсом I2C. При написанні програми для доступу до пристрою на базі I2C використовуємо команди SMBus.

При розробці програм для зв'язку з RPi через інтерфейс I2C на Python також використовуємо пакет бібліотеки SMBus, який надає широку підтримку доступу до пристроїв I2C. Отже, ми повинні додати підтримку SMBus для Python за допомогою пакетного менеджера `apt`:

```
sudo apt install python-smbus
```

Програмування

• По-перше, ми повинні встановити реєстр конфігурації A для середнього значення з 8 вимірювань з швидкістю виведення даних за замовчуванням 15 Гц.

• Налаштувати Gain, використовуючи регістр конфігурації B, тут вибрано 0xA0. (можемо вибрати будь-який інший бажаний)

• Вибираємо в регістрі режиму режим безперервного вимірювання. Отже, значення регістра режиму буде рівне 0x00.

Після ініціалізації прочитаємо рядки значень регістрів X, Y та Z-осей.

Обчислюємо значення напрямку за допомогою такої формули:

$$Magnetic\ heading = atan2\left(\frac{y}{x}\right) \text{ (Radian)}$$

$$True\ heading = Magnetic\ heading + Declination\ Angle \quad \text{(Radian)}$$

Звернемо увагу, що необхідно врахувати магнітне відхилення для даного географічного місця. Відхилення можна знайти, скориставшись ресурсом http://magnetic-declination.com. Запустивши сценарій на сайті, отримаємо для м.Київ (рис.8.9):

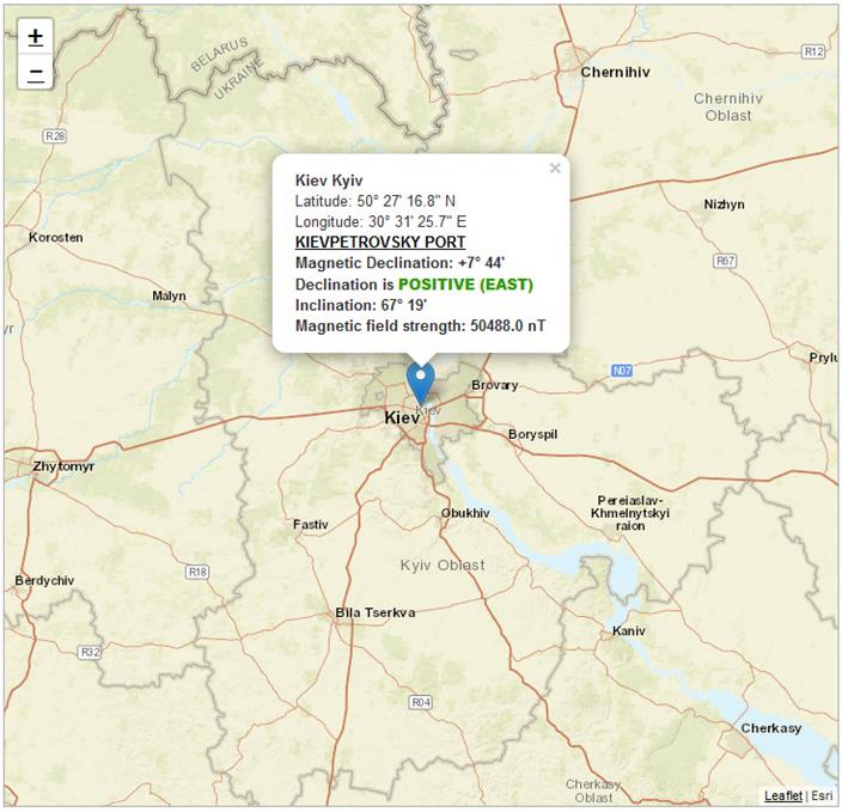


Рисунок 8.9 – Магнітне відхилення для даного географічного місця

Сценарій на Python:

```
"""
    Знаходження напрямку з GY-271 за допомогою Raspberry Pi і
Python
    http://www.electronicwings.com
"""
```

```
import smbus            #Імпорт модуля SMBus інтерфейсу I2C
from time import sleep  #Імпорт sleep
import math

#деякі регістри та їх адреси
Register_A     = 0              #Адреса регістра конфігурації A
Register_B     = 0x01           #Адреса регістра конфігурації B
Register_mode  = 0x02           #Адреса регістра режиму

X_axis_H    = 0x03              #Адреса регістра старшого байта
даних X-oci
Z_axis_H    = 0x05              #Адреса регістра старшого байта
даних Z-oci
Y_axis_H    = 0x07              #Адреса регістра старшого байта
даних Y-oci
declination = 0.135             #Кут відхилення для даного місця в
радіанах
pi          = 3.14159265359     #Значення pi

def Magnetometer_Init():
        #Запис в регістр конфігурації A
        bus.write_byte_data(Device_Address, Register_A, 0x70)

        #Запис в регістр конфігурації B для gain
        bus.write_byte_data(Device_Address, Register_B, 0xa0)

        #Запис в регістр режиму для вибору режиму
        bus.write_byte_data(Device_Address, Register_mode, 0)

def read_raw_data(addr):

        #Читання рядка 16-бітового значення
        high = bus.read_byte_data(Device_Address, addr)
        low = bus.read_byte_data(Device_Address, addr+1)
```

```
        #Об’єднання старшого і молодшого байтів
        value = ((high << 8) | low)

        #Отримання знаку значення з модуля
        if(value > 32768):
                value = value - 65536
        return value

bus = smbus.SMBus(1)    # або bus = smbus.SMBus(0) для старих
версій плати
Device_Address = 0x68  # Адреса магнітометра

Magnetometer_Init()    # Ініціалізація магнітометра

print (" Reading Heading Angle")

while True:
        #Читання рядка значення акселерометра
        x = read_raw_data(X_axis_H)
        z = read_raw_data(Z_axis_H)
        y = read_raw_data(Y_axis_H)

        heading = math.atan2(y, x) + declination

        #Через відхилення перевіряємо чи не більше >360 градусів
        if(heading > 2*pi):
                heading = heading - 2*pi

        #Перевірка знаку
        if(heading < 0):
                heading = heading + 2*pi

        #Перетворення в градуси
        heading_angle = int(heading * 180/pi)
```

```
print ("Heading Angle = %d " %heading_angle)
sleep(1)
```

Вирішення проблем

• Якщо результати команди `i2cdetect` видають помилку, то або не встановлено `i2c-tools`, або нам потрібно використати режим 0, а не 1.

• Якщо пристрій не виявлений і ви не бачите адреси, то або ми неправильно підключили пристрій, або не включений інтерфейс I2C.

• Ще раз перевіряємо з'єднання і перезавантажимо RPi.

У рідкісних випадках може бути несправний пристрій, але частіше більш ймовірно, що має місце одна з перерахованих вище причин.

8.3. Завдання

1. Отримати віддалено напрямок, використавши підключений до RPi модуль GY-271.

Примітка. Модулі на кожному з лабораторних стендів повернуті в різні сторони. Також щоденно змінюється їх орієнтація, тому після виконання ЛР надсилайте протокол без затримки, щоб записані в них дані відповідали реальним значенням на відповідну дату.

8.4. Зміст звіту

1. Сценарій на Python для визначення напрямку з коментарями.

2. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

8.5. Контрольні питання

1. Чому необхідно враховувати магнітне відхилення для даного географічного місця?

2. Як організувати підключення кількох сенсорів з інтерфейсом I2C до мікрокомп'ютера?

Лабораторна робота 9

ВИКОРИСТАННЯ UART НА RASPBERRY PI ЗА ДОПОМОГОЮ PYTHON

Мета роботи: Вивчити використання UART (Universal Asynchronous Receiver/Transmitter – універсального асинхронного приймача/передавача) – протоколу послідовного зв'язку, в якому дані передаються послідовно, тобто побітно. При такому зв'язку за один раз передається байт даних.

Зміст. Розглядається налаштування UART на RPi та його використання на прикладі модуля GPS.

9.1. Налаштування UART на Rapberry Pi

UART зазвичай використовується в RPi як зручний спосіб керування через GPIO або для доступу до завантажувальних повідомлень ядра з послідовної консолі (включено за замовчуванням). Він також може бути використаний як спосіб для взаємодії з Arduino, завантаженого ATmega, ESP8266 та інших мікроконтролерів з RPi. Треба бути обережними з рівнями логіки між пристроями, бо, наприклад, RPi має 3,3 В, а Arduino – 5 В. Підключивши їх разом ми можете їх пошкодити.

RPi має два вбудованих UART:

- **PL011 UART**
- **mini UART**

PL011 UART - це UART на базі ARM. Він має кращу пропускну здатність, ніж mini UART

Хоча RPi має два UART, але ми маємо лише одну пару виводів TXD і RXD для роботи.

UART PL011 є основним UART для моделей без функції Bluetooth і зв'язаний безпосередньо з консоллю Linux. Це означає, що через цей UART ми можемо надсилати команди Linux з нашого ПК на RPi.

З іншого боку, mini UART є Linux консоллю UART для моделей з Bluetooth, такими як Raspberry Pi 3 і Raspberry Pi Zero W. Для цих моделей UART PL011 зв'язаний безпосередньо з модулем Bluetooth.

UART PL011 більш надійний, ніж міні UART, оскільки останній має менші за обсягом FIFO, не може контролювати потік, а швидкість передачі даних залежить від тактової частоти GPU. Оскільки частота ядра графічного процесора змінюється, то змінюється і частота UART, що, в свою чергу, змінює швидкість передачі даних через UART. Це робить mini UART нестабільним і може призвести до втрати даних або їх пошкодження. Щоб зробити mini UART стабільним, треба змінити основну частоту. Також mini UART не підтримує перевірку парності.

Для кращого та ефективного зв'язку рекомендується використовуйте PL011 UART замість mini UART.

Для використання послідовного зв'язку треба включити UART RPi. В іншому випадку ми не зможемо спілкуватися через послідовний порт, оскільки порти UART використовуються для виведення консолі Linux та модуля Bluetooth.

В RPi у вікні терміналу введемо наступну команду, щоб дозволити UART:

```
sudo raspi-config
```

Виберемо -> Interfacing Options (рис.9.1):

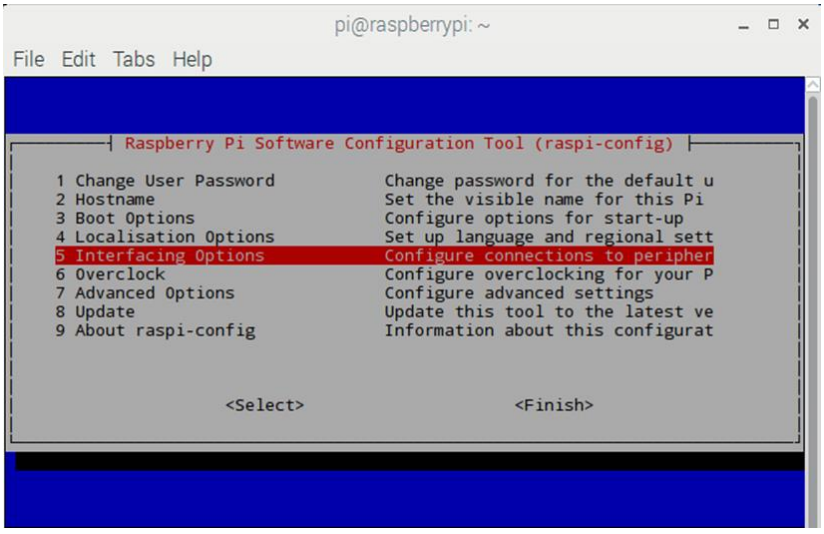


Рисунок 9.1 – Вибір вкладки інтерфейсів

Після вибору меню Interfacing, виберемо параметр Serial, щоб дозволити UART (рис.9.2):

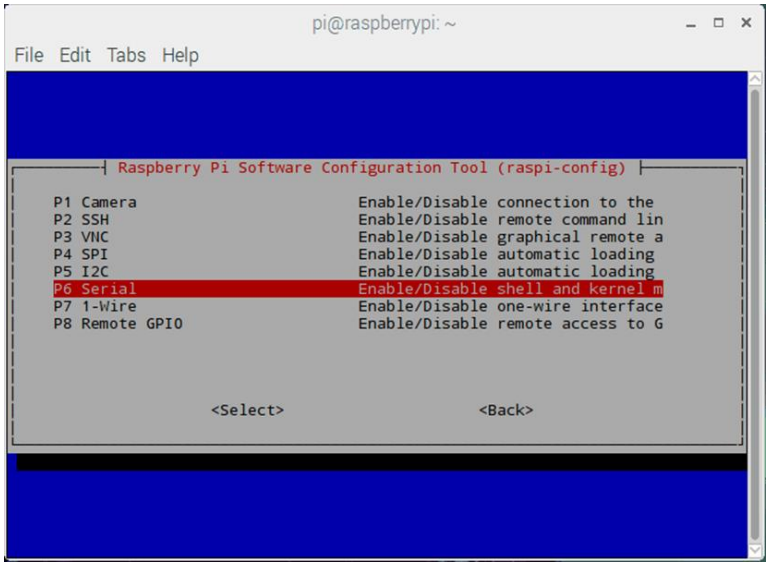


Рисунок 9.2 – Дозвіл послідовного порта

Дозволити використання UART можна також через вкладку Interfaces меню параметрів у графічному інтерфейсі RPi. Виводи Raspberry Pi 3 UART показані на рис.9.3:

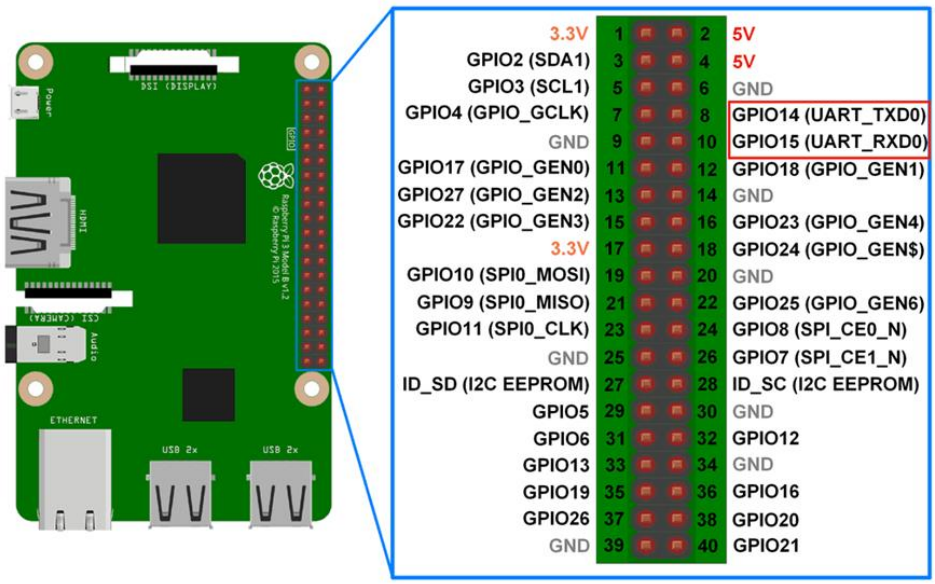


Рисунок 9.3 – Виводи UART на платі Raspberry Pi

UART виводи мають номери: в режимі BCM 14, 15, а в режимі WiringPi 15, 16.

Ще раз звернемо увагу, що виводи RPi використовують рівні логіки 3,3 В, тому ми не можемо підключити їх безпосередньо до пристроїв, які використовують 5В: Arduino UNO або ПК.

Для доступу до mini UART в Raspberry Pi 3 присвоюється порт `ttyS0`. А для доступу до PL011 в Raspberry Pi 3 буде присвоєний порт `ttyAMA0`. В попередніх моделях RPi доступний тільки порт `ttyAMA0`.

Апаратний порт mini UART, тобто GPIO14 (TXD) та GPIO15 (RXD) відомий як `serial0`, тоді як інший порт UART, який за замовчуванням підключений до модуля Bluetooth, відомий як `serial1`. Ці імена створюються як псевдоніми для переносимості версій RPi. Псевдоніми створюються в Raspbian. Отже, ми можемо замінити `ttyS0` або `ttyAMA0` на `serial0`.

Коли використовуємо `serial0` як UART-порт замість `ttyS0` або `ttyAMA0`, то програма, написана для Raspberry Pi 3, також буде працювати на старих моделях RPi.

Для кращої продуктивності послідовного зв'язку на GPIO14 та GPIO15 необхідно використовувати порт `ttyAMA0`, який підключений до модуля Bluetooth. Для використання цього порту ми повинні спочатку обміняти порти UART, тобто отримати `ttyAMA0` на GPIO14 та GPIO15, а `ttyS0` (mini UART) в модулі Bluetooth.

Обмін послідовними портами можна здійснити за допомогою міні-UART (`ttyS0`) для модуля Bluetooth через оверлей пристрою, тобто `pi3-miniuart-bt`. Або це можна зробити, повністю відключивши Bluetooth через заборону пристрою, тобто `pi3-disable-bt`.

Щоб поміняти порт UART, відкриємо файл `config.txt` командою:

```
sudo nano /boot/config.txt
```

а потім додамо в кінці файлу рядок, який показано нижче.

```
dtoverlay = pi3-miniuart-bt
```

або

```
dtoverlay = pi3-disable-bt
```

Після додавання рядка збережемо зміни у файлі та перезавантажимо систему. Ми можемо перевірити нове відображення послідовних портів за допомогою команди

```
ls -l /dev
```

Карта UART для `/dev/ttyS0` і `/dev/ttyAMA0` показана на рис9.4:

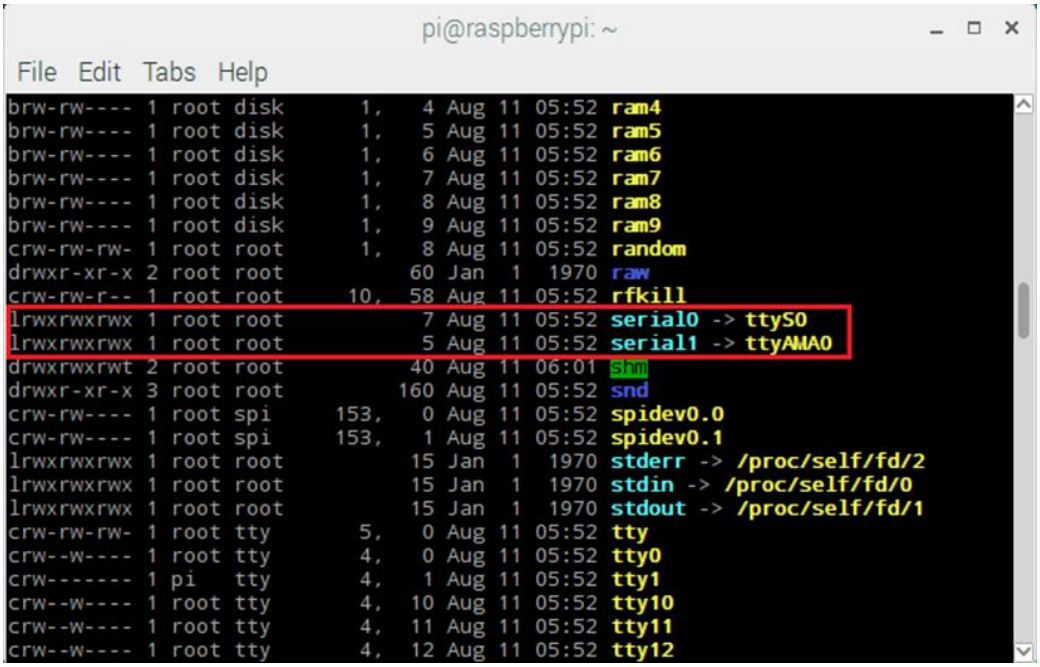


Рисунок 9.4 – Карта UART

9.2. Підключення модуля GPS до RPi через UART

Існує два способи підключення UART/послідовних пристроїв до Raspberry Pi: більш простий спосіб і трохи важчий шлях.

Продемонструємо підключення обома способами на прикладі використання модуля GPS (рис.9.5), але в лабораторній роботі буде використаний другий спосіб.

Рисунок 9.5 – Модуль GPS з антеною

Простий шлях – використання зовнішнього конвертора USB-UART

Безсумнівно, найпростіший спосіб додати послідовний порт - використати кабель з конвертором USB - UART. Кабель просто підключають до порта USB, як для підключення RPi до ПК. На іншому кінці кабелю є проводи або роз'єм, які забезпечують живлення, землю, RX, TX та, можливо, інші контрольні функції чи додаткові пристрої. Замість кабелю можна використати плату-конвертор на мікросхемах CP2102 або CP2104.

Виконаємо підключення модуля GPS таким способом (рис.9.6):

- GPS Vin до USB 5В або 3В (червоний провід на кабелі USB)
- GPS Ground до USB Ground (чорний провід)
- GPS RX до USB TX (зелений провід)
- GPS TX до USB RX (білий провід)

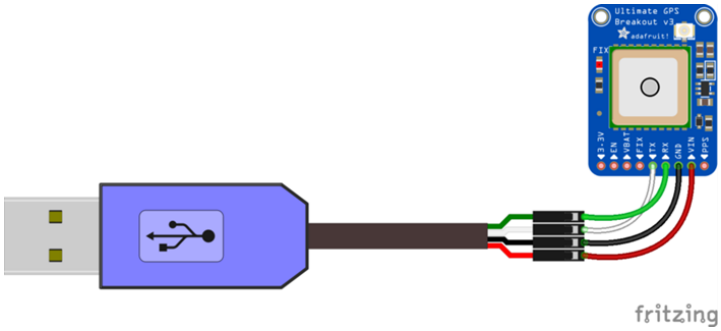


Рис.9.6 – Кабель з конвертором USB – UART та модуль GPS

Після підключення адаптера USB нам потрібно з'ясувати, яке ім'я отримав послідовний порт. Знайти ім'я можна шляхом відключення-підключення до USB, а потім набравши команду:

```
dmesg | tail -10
```

(або просто `dmesg`) і шукати текст, схожий на показаний нижче (рис.9.7):

```
pi@devpi:~ $ dmesg | tail -10
[  601.391424] usb 1-1.2: Manufacturer: FTDI
[  601.391432] usb 1-1.2: SerialNumber: AL00FP25
[  601.440489] usbcore: registered new interface driver usbserial
[  601.440554] usbcore: registered new interface driver usbserial_generic
[  601.440609] usbserial: USB Serial support registered for generic
[  601.455895] usbcore: registered new interface driver ftdi_sio
[  601.455970] usbserial: USB Serial support registered for FTDI USB Serial Device
[  601.456248] ftdi_sio 1-1.2:1.0: FTDI USB Serial Device converter detected
[  601.456383] usb 1-1.2: Detected FT232RL
[  601.459259] usb 1-1.2: FTDI USB Serial Device converter now attached to ttyUSB0
```

Рисунок 9.7 – Перевірка імені послідовного порта

Внизу побачимо назву приєднаного пристрою, в даному випадку ім’я `ttyUSB0`, тобто наш пристрій послідовного порта доступний на `/dev/ttyUSB0`.

Використання вбудованого UART

Якщо не хочемо підключати зовнішнє устаткування до RPi, то можемо використати вбудований UART на виводах RX/TX, хоча досвід використання зазначеного модуля GPS через конвертор показує більшу чутливість при прийомі сигналів.

Не забуваємо, що якщо ми це зробимо, то втратимо доступ до послідовної консолі, тому, якщо використовуєте кабель PiUART або консольний кабель чи HAT, який дозволяє підключатись безпосередньо до консолі, це більше не спрацює, і нам доведеться використовувати HDMI+клавіатуру або ssh-метод запуску команд.

Це не є проблемою, адже консоль послідовного входу більше не доступна за замовчуванням на Raspbian, але про це варто попередити.

Після перезавантаження можемо використовувати вбудований UART за допомогою `/dev/ttyS0`.

Під’єднаємо GPS таким чином (рис.9.8):

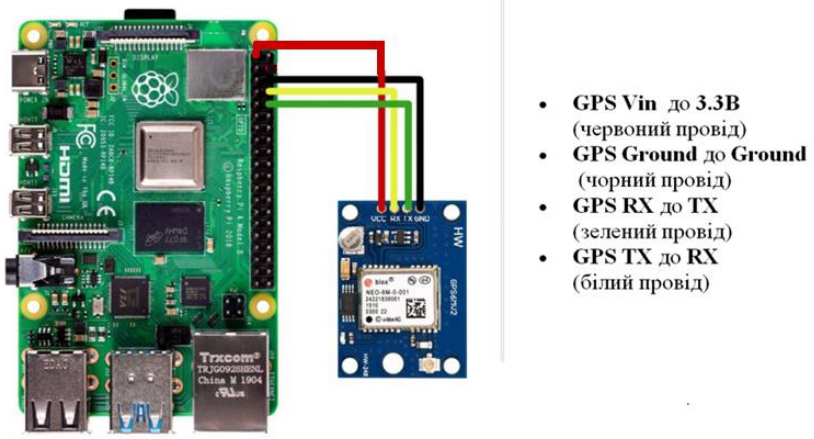


Рисунок 9.8 – Під’єднання модуля GPS до Raspberry Pi 4

На модулі два світлодіода - зелений (вмикається при подачі живлення), синій (вмикається при подачі живлення, починає моргати після встановлення зв’язку з супутниками).

Увага! GPS сигнал сам по собі дуже слабкий, тому в приміщенні антену треба розмістити як можна ближче до вікна і направити на небо (якщо знаходимося в приміщенні).

Тепер, якщо все правильно підключено, набрати в консолі команду

```
cat /dev/ttyAMA0
```

і у виводі побачимо повідомлення у форматі NMEA, які GPS приймач передає на RPi.

Встановлення необхідного програмного забезпечення

Встановлюємо `gpsd` і `gpsd-client`:

```
sudo apt install gpsd gpsd-clients
```

`gpsd` - служба, яка приймає дані у форматі NMEA від GPS модуля і "розшарює" доступ до ним для стороннього програмного забезпечення. Один з варіантів отримання даних від `gpsd` – це отримання их їх від сокет-сервера, який піднімає `gpsd` на порту 2947 (за замовчуванням), у JSON форматі. Значна частина навігаційного ПЗ вимагає наявності `gpsd`.

Для отримання додаткової інформації про програму можемо ввести:

```
man gpsd
```

Після встановлення переконаємося, що можемо отримувати дані від модуля GPS. Для цього виведемо дані, які він надсилає нам через послідовний порт:

```
cat /dev/serial0
```

Спочатку результат повинен виглядати приблизно так (рис.9.9):

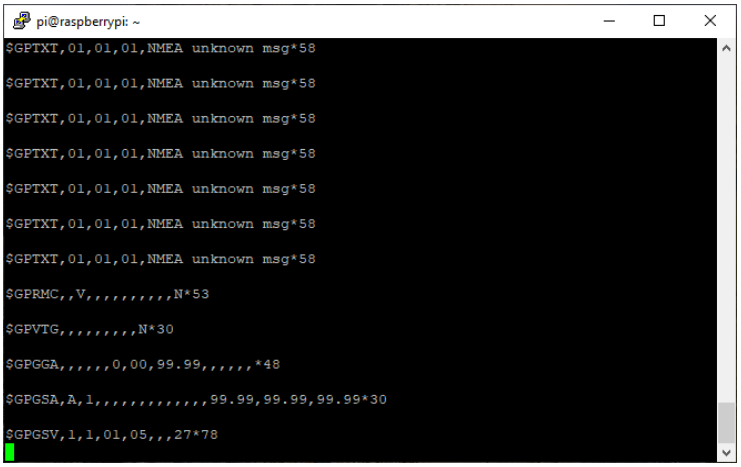


Рисунок 9.9 – Початок передачі даних через послідовний порт

Не має значення, які дані отримуємо на даний момент, якщо щось отримуємо. Якщо ж порт негайно закривається або RPi взагалі не отримує даних, перевіряємо, чи правильно ми підключили модуль. Звертаємо увагу, що ми повинні мати змогу запустити цю команду не будучи суперкористувачем. Якщо не можемо це зробити, то додаємо користувача pi до групи діалогового вікна:

```
sudo adduser pi dialout
```

Зчитування даних про позицію

Тепер, нарешті, настав час визначити місцеположення RPi. Введемо наступну команду, щоб зупинити службу `gpsd`, яка запустилася автоматично, коли раніше встановлювали `gpsd`. Ми повинні зробити це, оскільки параметри за замовчуванням для RPi не правильні:

```
sudo systemctl stop gpsd.socket
```

Звернемо увагу, що нам доведеться вводити цю команду щоразу під час завантаження системи. Ми також можемо повністю його відключити:

```
sudo systemctl disable gpsd.socket
```

Запускаємо новий примірник `gpsd`, який перенаправляє дані правильного послідовного порта в сокет:

```
sudo gpsd /dev/serial0 -F /var/run/gpsd.sock
```

А потім можемо виконати будь-яку з наступних двох команд для відображення даних GPS:

```
sudo gpsmon
sudo cgps -s
```

Спочатку ми повинні побачити щось подібне (рис.9.10):

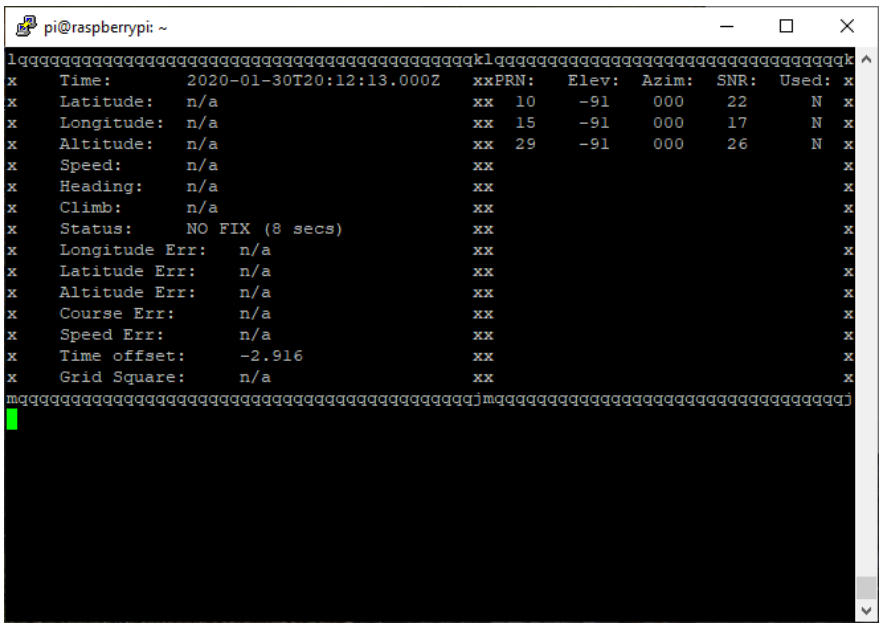


Рисунок 9.10 – Початок отримання даних із супутників

Однак, через деякий час модуль повинен буде зібрати достатньо даних для відображення місцеположення (рис.9.11):

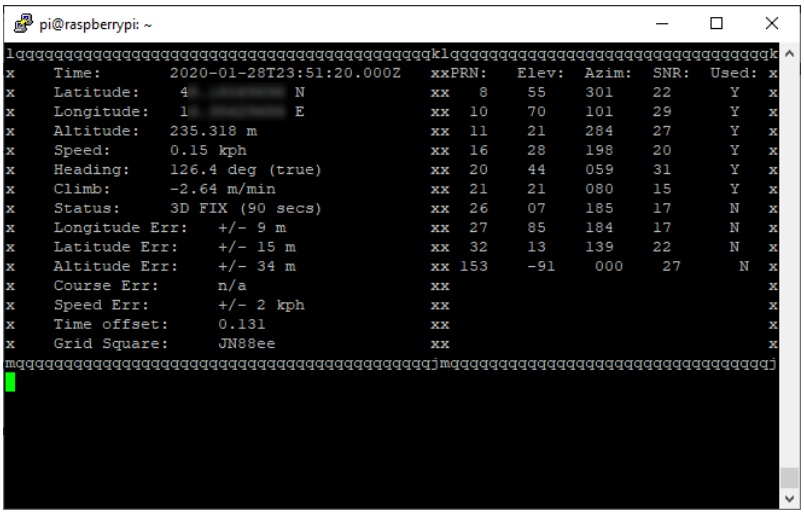


Рисунок 9.11 – Зібрані дані для відображення

Звертаємо увагу, що може знадобитися до 30 хвилин, поки модуль зможе визначити наше положення під час першого завантаження, особливо якщо перебуваєте в приміщенні. Спробуємо направити антену на вікно або, ще краще, розташуйте весь модуль поруч з вікном чи на відкритому повітрі.

Спробуємо виконати таку команду, якщо під час запуску `gpsmon` з’являється помилка, а після запуску `cgps` немає результатів:

```
sudo systemctl stop serial-getty@serial0.service
```

Тоді ми також зможемо використовувати `gpsmon` (рис.9.12):

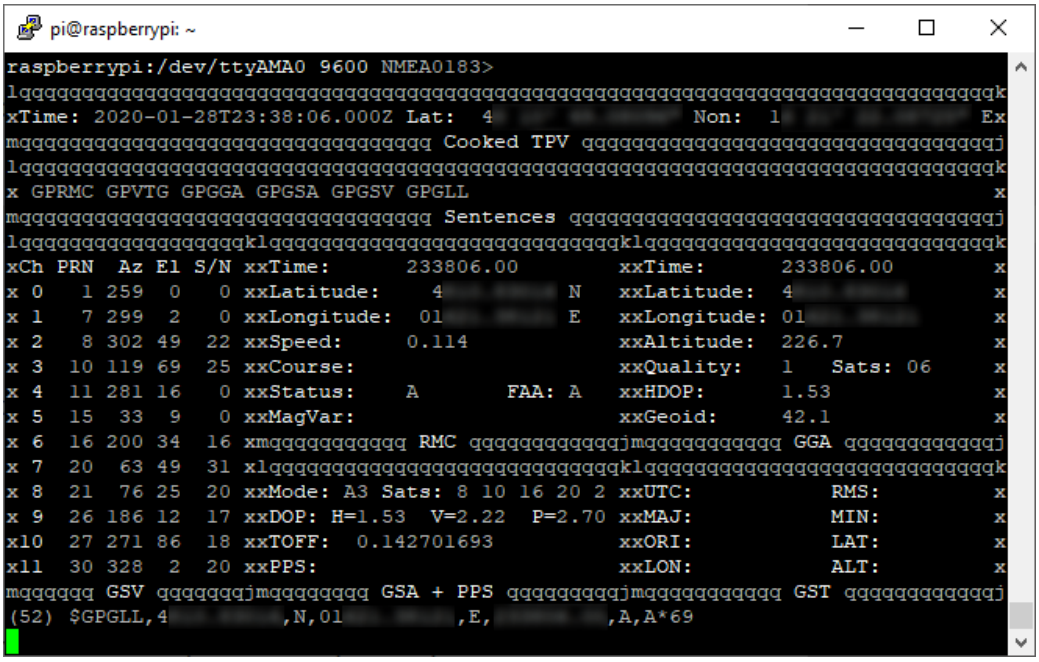


Рисунок 9.12 – Використання `gpsmon`

Приклад

Давайте організуємо взаємодію GPS-модуля з RPi і будемо отримувати GPS-інформацію. Можемо підключити модуль GPS до RPi за допомогою Python та C (WiringPi). Для взаємодії модуля GPS підключимо модуль GPS до RPi, як показано на рис.9.8.

Використання Python

Отримаємо інформацію про широту, довготу та час із рядка NMEA GPGGA, отриманого від модуля GPS за допомогою Python і виведемо її на консоль (термінал). Використовуючи ці широту та довготу, знайдемо поточну позицію на карті Google.

Програма на Python:

```
'''
GPS взаємодія з Raspberry Pi за допомогою Pyhton
http://www.electronicwings.com
'''
import serial                #імпорт пакета для послідовного порта
from time import sleep
```

```
import webbrowser           #імпорт пакета для відкривання
посилання в браузері
import sys                  #імпорт пакета sys

def GPS_Info():
    global NMEA_buff
    global lat_in_degrees
    global long_in_degrees
    nmea_time = []
    nmea_latitude = []
    nmea_longitude = []
    nmea_time = NMEA_buff[0]      #витягнути час із рядка GPGGA
    nmea_latitude = NMEA_buff[1]  #витягнути широту з рядка GPGGA
    nmea_longitude = NMEA_buff[3] #витягнути довготу з рядка GPGGA

    print("NMEA Time: ", nmea_time,'\n')
    print ("NMEA Latitude:", nmea_latitude,"NMEA Longitude:",
nmea_longitude,'\n')

    lat = float(nmea_latitude)      #перетворити рядок у float для
обчислення
    longi = float(nmea_longitude)   #перетворити рядок у float для
обчислення

    lat_in_degrees = convert_to_degrees(lat)    #отримати широту в
градусах у десятковому форматі
    long_in_degrees = convert_to_degrees(longi) #отримати довготу
в градусах у десятковому форматі

#перетворити необроблений рядок NMEA в градуси у десятковому
форматі
def convert_to_degrees(raw_value):
    decimal_value = raw_value/100.00
    degrees = int(decimal_value)
    mm_mmmm = (decimal_value - int(decimal_value))/0.6
```

```
    position = degrees + mm_mmmm
    position = "%.4f" %(position)
    return position
gpgga_info = "$GPGGA,"
ser = serial.Serial ("/dev/ttyS0")       #Відкрити порт зі
швидкістю передачі даних
GPGGA_buffer = 0
NMEA_buff = 0
lat_in_degrees = 0
long_in_degrees = 0

try:
    while True:
        received_data = (str)(ser.readline())        #прочитати
прийнятий рядок NMEA
        GPGGA_data_available = received_data.find(gpgga_info)
#перевірка для рядка NMEA GPGGA
        if (GPGGA_data_available>0):
            GPGGA_buffer = received_data.split("$GPGGA,",1)[1]
#зберегти дані, що надходять після рядка "$GPGGA,"
            NMEA_buff = (GPGGA_buffer.split(','))
#зберегти дані, розділені комами, в буфері
            GPS_Info()
#отримати час, широту, довготу

            print("lat in degrees:", lat_in_degrees," long in
degree: ", long_in_degrees, '\n')
            map_link = 'http://maps.google.com/?q=' +
lat_in_degrees + ',' + long_in_degrees       #створити посилання
для розміщення місця на карті Google
            print("<<<<<<<<press ctrl+c to plot location on google
maps>>>>>>\n")               #натиснути ctrl+c, щоб нанести графік
на карту і вийти
            print("-----------------------------------------------
-------------\n")
```

```
except KeyboardInterrupt:
    webbrowser.open(map_link)        #відкрити інформацію про
поточну позицію на карті Google
    sys.exit(0)
```

Вивід на Python IDE (рис.9.13):

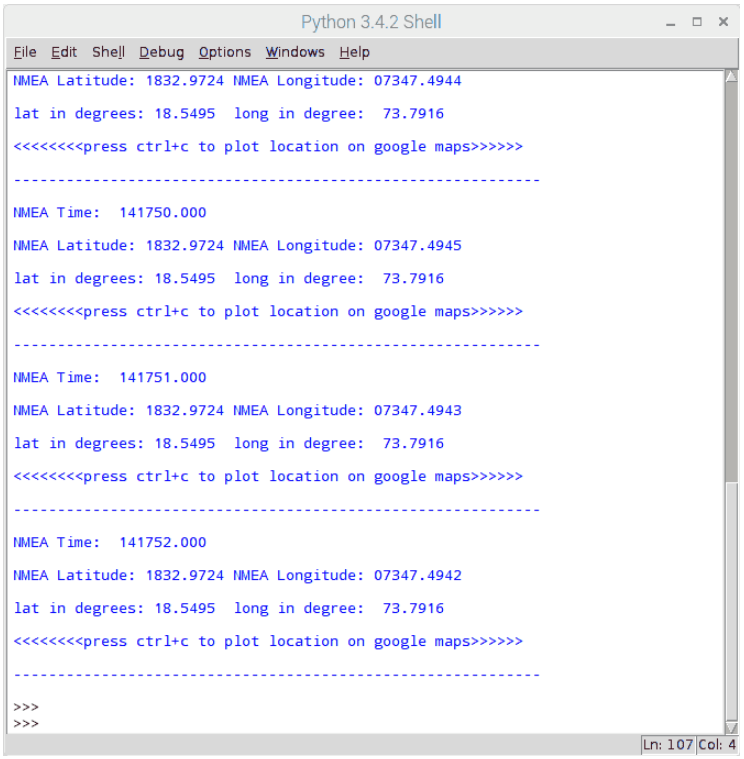


Рисунок 9.13 – Дані NMEA

Виведення розташування на карті Google

Щоб показати своє місце розташування на карті Google, нам треба звернутися за посиланням URL для карти Google. Ми можемо використати наступне посилання для відкриття карти Google із нашими витягнутими довготою та широтою:

```
http://maps.google.com/?q= <широта>, <довгота>
```

9.2. Завдання

1. Встановити необхідне ПЗ та запустити зібрану систему визначення місця положення.

2. В протоколі навести скріншоти результатів виконання роботи, в тому числі, визначене місцеположення RPi на карті Google.

9.3. Зміст звіту

1. Сценарій на Python для визначення координат місцезнаходження з коментарями.

2. Скріншоти результатів роботи сценарію.

3. Висновки за результатами виконання роботи.

Звіт в електронному вигляді (бажано у форматі pdf) завантажити у відповідну папку в Moodle.

9.4. Контрольні питання

1. Як збільшити чутливість GPS-приймача?

2. Як доповнити сценарій визначення координат, щоб місцезнаходження відразу відображалось на карті Google?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова література:

1. Могильний С.Б. Мікрокомп’ютер Raspberry Pi – інструмент дослідника. - К.: Талком, 2014. – 340 с. (Електронна версія http://isearch.kiev.ua/uk/book/1850-microcomputer-raspberry-pi-tool-researcher)
2. Яковенко А.В. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 «Комп’ютерні науки». – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.
3. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.
4. Копей В.Б. Мова програмування Python для інженерів і науковців: Навчальний посібник. Івано-Франківськ : ІФНТУНГ, 2019. 274с.
5. Креневич А.П. Python у прикладах і задачах. Частина 1. Структурне програмування. Навчальний посібник із дисципліни "Інформатика та програмування" – К.: ВПЦ "Київський Університет", 2017. – 206 с.

Додаткова література:

1. Simon Monk. Raspberry Pi Cookbook. - O’REILLY, 2016. – 510 p.
2. Stewart Watkiss. Learn Electronics with Raspberry Pi. – Apress, 2016. – 300 p.
3. Alex Bradbury, Ben Everard. Learning Python with Raspberry Pi. - Wiley, 2013. – 288 p.
4. Tim Cox. Raspberry Pi Cookbook for Python Programmers. - Packt Publishing, 2014. – 402 p.
5. John C. Shovic. Raspberry Pi IoT Projects: Prototyping Experiments for Makers. - Washington, USA.: Liberty Lake, 2016, - 253 p.
6. Eben Upton, Gareth Halfacree. John. Raspberry Pi® User Guide, 4th Edition. - Chichester, West Sussex, United Kingdom.: Wiley & Sons Ltd, 2016. - 315 p.

7. Sams Teach Yourself Python Programming for Raspberry Pi in 24 Hours, Second Edition, Pearson Education, Inc., 2016. — 1760 с.

Інформаційні ресурси Інтернету:

1. Сайт Академії Mikrotik:
   - https://mikrotik.kpi.ua/index.php/courses-list/category-raspberry
   - https://mikrotik.kpi.ua/index.php/courses-list/category-python
2. Персональний сайт викладача: - http://isearch.kiev.ua/
3. Сайт дистанційного навчання на платформі Moodle Академії Mikrotik: - http://iot.kpi.ua/lms/
4. Платформа дистанційного навчання «Сікорський»: - https://www.sikorsky-distance.org/