

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

**Кваліфікаційна робота бакалавра
за спеціальністю 121 Інженерія програмного забезпечення**

на тему:

**РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЧИТУВАННЯ
ДАНИХ СИМВОЛЬНИХ ПРИСТРОЇВ ТА ЇХ ПЕРЕДАЧІ НА СЕРВЕР
ЧЕРЕЗ GPRS/GSM ЗВ'ЯЗОК**

Виконав студент 4-го курсу
Опанюк Микита Ігорович

(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Галкін Олександр Володимирович

(підпис)

Засвідчую, що в цій курсовій роботі
немає запозичень з праць інших авторів в
без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри інтелектуальних
програмних систем

« ____ » _____ 202_ р.,

протокол № ____
Завідувач кафедри

О. І. Провотар

(підпис)

РЕФЕРАТ

Обсяг роботи 54 сторінки, 9 ілюстрацій, 3 таблиці, 16 джерел посилань.

РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВИЧИТУВАННЯ ДАНИХ СИМВОЛЬНИХ ПРИСТРОЇВ ТА ЇХ ПЕРЕДАЧІ НА СЕРВЕР ЧЕРЕЗ GPRS/GSM ЗВ'ЯЗОК, ДОСЛІДЖЕННЯ ОПЕРАЦІЙНОЇ СИСТЕМИ LINUX, ІНСТРУМЕНТАРІЮ YOSTO PROJECT, GPRS/GSM ЗВ'ЯЗКУ, РЕАЛІЗАЦІЇ ДРАЙВЕРІВ, ВИКОРИСТАННЯ SSH, SCP ПРОТОКОЛІВ.

Об'єктом роботи є процес дослідження поведінки різних типів пристроїв та налаштування їх працездатності за допомогою драйверів, завдяки якому операційна система (Linux) має можливість отримувати інформацію для подальшого використання, а саме вичитування отриманих даних за допомогою програмного забезпечення для надсилання отриманої інформації на сервер. Предметом роботи є набір програмного забезпечення для операцій взаємодії цих функціональних блоків між собою.

Метою роботи є створення програмного забезпечення для налаштування та реалізації взаємодії функціональних блоків комп'ютера між собою та операційною системою для можливості отримання інформації з подальшим надсиленням на сервер.

Методи розроблення: комп'ютерне моделювання, методи роботи з різними типами пристроїв, інформаційних шин комп'ютера, розробка програмного продукту на основі сучасної операційної системи.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Eclipse IDE 4.11, мова програмування C, Yocto project ARM toolchain (набір необхідних пакетів програм для компіляції та генерації виконуваного коду з текстів програм на мові C).

Результати роботи: виконано загальний огляд електронних засобів, налагодження роботи та взаємодії електронних засобів з вбудовуваним одноплатним комп'ютером, на основі процесора архітектури ARM, BeagleBone

Black та модулів типу SIM800, проаналізовано переваги та недоліки використання різних версій модуля SIM800 та протоколів підключення між модулем та одноплатним комп'ютером, створено апаратну складову для реалізації безперебійної можливості функціонування без втручання людини, розроблено програмні продукти (драйвера, модулі, додаткове програмне забезпечення, bash-скрипти), які дозволяють наочно демонструвати процеси роботи та взаємодії різних апаратних засобів між собою, а також зчитувати інформацію з пристроїв в просторі користувача операційної системи, для подальшого надсилання даних на сервер.

За методами розробки та інструментальними засобами робота виконувалася сумісно з прикладами драйверів, реалізованих в операційній системі Linux а також з прикладами реалізації клієнт-серверного підключення на основі системного адміністрування комп'ютерних мереж.

Програмні продукти можуть застосовуватися в навчальному процесі університетського як для курсу системного програмування, так і для курсу системного адміністратора. Програмні продукти представляють із себе автоматизовану систему, що може бути використана для багатьох задач, які на пряму пов'язані з надсиланням інформації з віддаленого місцезнаходження пристрою.

ЗМІСТ

	С.
СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ	5
ВСТУП	7
РОЗДІЛ 1. АПАРАТНА СКЛАДОВА СИСТЕМИ	12
1.1 Поняття вбудовуваного одноплатного комп'ютера	12
1.2 Плата GPRS/GSM зв'язку SIM800	16
РОЗДІЛ 2. ОПЕРАЦІЙНА СИСТЕМА LINUX, РЕАЛІЗАЦІЯ ДРАЙВЕРІВ ПРИСТРОЇВ	19
2.1 ОС Linux	19
2.2 Проект Yocto для роботи з дистрибутивами Linux	21
2.3 Налаштування та компіляція ОС Linux та модулів на основі інструментарію Yocto	22
2.4 Драйвери символьних пристроїв Linux	25
РОЗДІЛ 3. РЕАЛІЗАЦІЯ КЛІЄНТ-СЕРВЕРНОГО ПІДКЛЮЧЕННЯ	32
3.1 PPP протокол підключення для використання SIM800	32
3.2 Реалізація клієнт-серверного підключення на основі SSH	36
ВИСНОВКИ	40
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ	41
ДОДАТОК А Реалізація драйвера матричної клавіатури	43
ДОДАТОК Б Реалізація драйвера датчика DHT11	46
ДОДАТОК В Функціонал клієнтської частини для надсилання даних на сервер	52

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

ОС — операційна система;

IoT — Internet of Things, Інтернет речей - концепція мережі, що складається із взаємозв'язаних фізичних пристроїв, які мають вбудовані датчики, а також програмне забезпечення, що дозволяє здійснювати передачу і обмін даними між фізичним світом і комп'ютерними системами в автоматичному режимі, за допомогою використання стандартних протоколів зв'язку;

IDE — Integrated development environment, інтегроване середовище розробки;

GSM — Global System for Mobile Communications, система мобільного зв'язку;

GPRS — General Packet Radio Service, загальний сервіс пакетної радіопередачі;

BBB — BeagleBone Black, вбудовуваний одноплатний комп'ютер;

eMMC — embedded MultiMediaCard, вбудована енергонезалежна система пам'яті;

API — Application programming interface, прикладний програмний інтерфейс;

GPIO — General-purpose input/output, інтерфейс введення/виведення загального призначення;

I2C — Inter-Integrated Circuit, послідовна шина даних для зв'язку інтегральних схем;

UART — universal asynchronous receiver/transmitter, універсальний асинхронний приймач/передавач;

USB — Universal serial bus, універсальна послідовна шина, призначена для з'єднання периферійних пристроїв комп'ютера;

DTS — Device tree source, представляє спеціальну структуру даних, що описує апаратні компоненти конкретного комп'ютера, так що ядро операційної системи може використовувати і керувати тими компонентами, включаючи процесор або процесор, пам'ять, шини і периферійні пристрої;

IRQ — Interrupt request, "запит на преривання". Так прийнято називати спеціальний сигнал, який повідомляє процесору про необхідність припинення

виконання поточної програми, зберігаючи її стан, і перехід до заздалегідь заданої адреси пам'яті для її обробки;

ARM — Advanced RISC Machine, поліпшена RISC машина. 32-бітна RISC архітектура процесорів;

RISC — Reduced Instruction Set Computing, обчислення зі скороченим набором команд. Архітектура процесорів зі скороченим набором команд;

SSH — Secure Shell, мережевий протокол рівня застосунків, що дозволяє проводити віддалене управління комп'ютером і тунелювання TCP-з'єднань;

SCP — secure copy, протокол для безпечного копіювання файлів між локальним та віддаленим хостом або між двома віддаленими хостами;

ВСТУП

Оцінка актуальності реалізації програмного забезпечення для вичитування даних символьних пристроїв та їх передачі на сервер через GPRS/GSM зв'язок. В наші дні панування та розвитку технологій людина створює нові пристрої для вимірювання, спостереження, зберігання інформації в першу чергу для покращення свого існування. Завдяки швидкому розвитку в комп'ютерній техніці та значним розширенням можливостей було створено як величезну кількість різновидів обчислювальних машин для виконання різних задач, так і допоміжних пристроїв для отримання певного роду інформації, що може бути використана людиною у своїх потребах.

Було створено чудове рішення - ОС, що допомагала нам вирішити такі проблеми, як: багатозадачність, можливість виконання різних операцій за одиницю часу, розподілення ресурсів комп'ютера, зберігання та обробка результатів, вирішення проблем конкурентності між задачами за ресурси комп'ютера і найголовніше – створення можливості узгодженої роботи між окремими фізичними модулями комп'ютера. Саме цю можливість узгодженості пристроїв створюють окремі модулі операційної системи – драйвера.

Проте драйвер може лише надати функціонал для взаємодії з пристроями. Саме для цього і існує в ОС простір користувача — для подальшого використання функціоналу для аналізу отриманих даних. Проте виникають ситуації, коли спеціалізованих комп'ютерів з додатковою периферією — значна кількість, крім того користувач не може мати прямого доступу до керування ними. Гарним прикладом рішення цієї проблеми — IoT, як приклад системи автоматизації процесів накопичування та аналізу інформації для подальшої передачі на віддалений сервер. Інтернет надає чудову можливість обмінюватись даними, тим самим сильно спрощуючи життя користувача. В наш час автоматизація та інтернет є найважливішими речами, без яких більшість із нас не можуть уявити своє існування.

Актуальність роботи та підстави для її виконання. Завдяки постійному розвитку різних сфер людської діяльності виникають нові потреби для вирішення сучасних задач, перш за все, найбільш оптимальним шляхом. Нові технології, їх масовість дають людству чудову можливість спростити своє життя - налагодження віддаленої роботи з різними пристроями, а головне – простим, “абстрагованим” способом використовувати новий функціонал без будь-яких потреб у додаткових знаннях та робочих умовах. Всі ці можливості нам надає програмне забезпечення - для реалізації взаємодії модулів комп’ютера між собою, отримання та передачі через інтернет інформації про стан модулів для подальшого аналізу та роботи з ними. Все це разом утворює систему **IoT**, яка в наш час є популярним рішенням віддаленого контролю над процесами.

Linux, як один з найкращих варіантів легко-адаптивного відкритого програмного забезпечення, буде продовжувати розвиватися та займати перші місця для компактних одноплатних комп’ютерів, серверів, мейнфреймів, смартфонів та багатьох інших пристроїв, а разом з ним буде рости потреба в покращенні та розробки як нових драйверів, так і створення нових стандартів та програмного забезпечення користувацького простору ОС.

Одним з варіантів виходу в мережу інтернет з мінімальними матеріальними витратами зі сторони реалізації апаратно-програмного рішення для отримання та передачі даних — це використання модуля GPRS/GSM зв'язку SIM800. Завдяки масовості, простоті у використанні, відкритому доступу до технічної інформації модуль дає великий набір можливостей для вирішення поставлених задач, а саме — отримання та передача інформації через інтернет.

Мета й завдання роботи. Метою роботи є дослідити та підібрати електронні засоби, на основі яких потрібно реалізувати програмне забезпечення для налаштування взаємодії функціональних блоків комп’ютера між собою та операційною системою для подальшої передачі інформації користувачеві через глобальну мережу інтернет. Для досягнення цієї мети поставлено такі завдання:

- Дослідити різновиди одноплатних комп'ютерів та модулів GPRS/GSM зв'язку, провести порівняльну характеристику та реалізувати апаратну складову.
- Ознайомлення з ОС Linux, її дистрибутивами, особливостями та можливостями інструментарію Yocto Project для роботи з дистрибутивами.
- Дослідити різновиди допоміжних пристроїв, реалізувати набір драйверів під ОС Linux для аналіз та контролю статусу допоміжних пристроїв.
- Створити програмне забезпечення для отримання інформації з реалізованих драйверів та передачі даних віддаленому серверу на основі отриманих знань.

Об'єкт, методи й засоби розроблення. Об'єктом роботи є процес порівняння різних одноплатних комп'ютерів, дослідження поведінки різних допоміжних периферійних пристроїв, налаштування їх працездатності за допомогою драйверів, створення програмного забезпечення, завдяки якому ОС отримує доступ до апаратної складової комп'ютера для виконання операцій, налаштувань, вичитування інформації, а також взаємодії з модулем зв'язку SIM800 для подальшої передачі інформації через інтернет. Предметом роботи є набір програмного забезпечення для операцій взаємодії функціональних блоків між собою для отримання інформації та її передачі на сервер.

Методи розроблення: комп'ютерне моделювання, методи роботи з різними типами пристроїв, інформаційних шин комп'ютера, розробка програмного продукту на основі сучасної операційної системи, комп'ютерні мережі, системне адміністрування.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Eclipse IDE Version: Oxygen.3a Release (4.7.3a), мова програмування C, Yocto project (набір необхідних інструментів та пакетів програм для роботи з дистрибутивами ОС Linux, компіляції та генерації виконуваного коду з текстів програм на мові C), оболонка bash для реалізації допоміжних скриптів.

Результати роботи: виконано загальний огляд електронних засобів, налагодження роботи та взаємодії електронних засобів з одноплатним комп'ютером, на основі процесора архітектури ARM — BBB, та модуля GPRS/GSM зв'язку SIM800L, проаналізовано можливості ОС Linux, переваги та недоліки використання різних методів, структур даних, стандартів та інтерфейсів для отримання доступу до глобальної мережі інтернет, на базі дистрибутиву для BBB розроблено програмні продукти :

- драйвери матричної клавіатури та датчика DHT11;
- користувацька програма для передачі інформації про стан пристроїв на сервер на основі SSH підключення та SCP протоколу;
- допоміжні скрипти для налаштувань PPP підключення між BBB та SIM800L;

які дозволяють наочно продемонструвати процеси роботи та взаємодії різних апаратних засобів між собою, а також зчитувати інформацію з пристроїв на рівні операційної системи - як в просторі ядра, так і в просторі користувача для збереження інформації в файловій системі для подальшої передачі на віддалений сервер.

Взаємозв'язок з іншими роботами. За методами розробки та інструментальними засобами робота виконувалася сумісно з прикладами драйверів, реалізованих в операційній системі Linux.

Можливі сфери застосування. Програмні продукти можуть застосовуватися для створення вузькоспрямованих спеціалізованих систем, що можуть бути корисні:

- в аграрній сфері – отримання даних про стан ґрунту, повітря, вологи, погоди.
- для розробки охоронних, протипожежних систем, що базуються на використанні датчиків руху, світла, температури.

Програмні продукти можуть застосовуватися в навчальному процесі університетського курсу системного програмування під час вивчення модулів та драйверів операційної системи для роботи з апаратної складовою комп'ютера, а також в навчальному процесі курсу комп'ютерних мереж для вивчення методів налаштування доступу до мережі інтернет через GPRS/GSM зв'язок.

РОЗДІЛ 1.

АПАРАТНА СКЛАДОВА СИСТЕМИ

1.1 Поняття одноплатного комп'ютера

Одноплатний комп'ютер [1] - самодостатній комп'ютер, зібраний на одній друкованій платі, на якій встановлені мікропроцесор, оперативна пам'ять, системи введення-виведення і інші модулі, необхідні для функціонування комп'ютера. Одноплатні комп'ютери виготовляються в якості демонстраційних систем, систем для розробників або освіти, або для використання в ролі промислових або вбудованих комп'ютерів.

На відміну від традиційних персональних комп'ютерів форм-фактора «desktop», Одноплатні комп'ютери часто не вимагають установки якихось додаткових периферійних плат. Деякі одноплатні системи виготовлені у вигляді компактної плати з процесором і пам'яттю, що підключаються до backplane для розширення можливостей, наприклад, для збільшення кількості доступних роз'ємів.

Найчастіше ці рішення повинні бути захищені або дуже компактні, тому всі компоненти повинні розташовуватися на одній платі.

Така економія з одного боку робить все пристрій більш компактным і набагато дешевшим за рахунок використання системи на кристалі, з іншого боку, розширення можливостей - зміна процесора або пам'яті - утруднене, так як найчастіше ці компоненти напаяні на плату.

Найбільш популярними прикладами одноплатних комп'ютерів є:

а) Raspberry Pi;

б) BBB;

Обидва приклади базуються на системі на чипі архітектури ARM.

Архітектура процесора - це набір інструкцій, які можуть використовуватися при складанні програм і реалізовані на апаратному рівні за допомогою певних сполучень транзисторів процесора. Саме вони дозволяють

програмам взаємодіяти з апаратним забезпеченням і визначають яким чином будуть передаватися дані в пам'ять і зчитуватися звідти [1].

На даний момент існують два типи архітектур: CISC(x86) і RISC(ARM). Перша передбачає, що в процесорі будуть реалізовані інструкції на всі випадки життя, друга, RISC - ставить перед розробниками завдання створення процесора з набором мінімально необхідних для роботи команд. Інструкції RISC мають менший розмір і простіші.

ARM архітектура була представлена трохи пізніше за x86 - в 1985 році. Вона була розроблена відомою в Британії компанією Acorn, тоді ця архітектура називалася Arcon Risk Machine і належала до типу RISC, але потім була випущена її поліпшена версія, яка зараз і відома як ARM. При розробці цієї архітектури інженери ставили перед собою мету усунути всі недоліки x86 і створити абсолютно нову і максимально ефективну архітектуру. ARM чіпи отримали мінімальне енергоспоживання і низьку ціну, але мали низьку продуктивність роботи в порівнянні з x86. Проте за останні роки продуктивність процесорів ARM поліпшувалася. З огляду на це, з врахуванням низького енергоспоживання та простоти у виробництві, їх почали дуже широко застосовувати в мобільних чи вузькоспеціалізованих пристроях, таких як планшети, смартфони, медіа системи а також — одноплатні комп'ютери.

Raspberry Pi [2] — одноплатний комп'ютер, розроблений британським фондом Raspberry Pi Foundation. Його головне призначення — стимулювати вивчення у школах та вищих навчальних закладах як базового, так і спеціалізованого системного програмування, для набуття знань про апаратну складову та технологічні особливості, навичок роботи з різним типом внутрішньою чи зовнішньою периферією. чи з обмеженими ресурсами пристрою. Raspberry Pi Foundation пропонує широкий спектр різновидів розроблених під різні задачі в залежності від складності та потреб у використанні ресурсів: Raspberry Pi Zero, Raspberry Pi Zero W, Raspberry Pi 3A, Raspberry Pi 3B, Raspberry Pi 4 тощо. Raspberry Pi 4 пропонує найкращі показники з такими зручностями, як

подвійний мікро-вихід HDMI з можливістю 4K, до 4 ГБ оперативної пам'яті LPDDR4, а також більш швидка система на чипі.



Рисунок 1. Raspberry Pi

BBB [3] [16] — аналогічно як і Raspberry Pi представляє із себе мініатюрний комп'ютер для електронних проєктів, де одночасно потрібна і висока продуктивність, і широкі можливості для підключення периферії, так само зв'язок з мережею і інтернетом за допомогою ОС Linux. BBB - найновіший член родини BeagleBoard. Це дешевий, розширений BeagleBoard з низькою вартістю, використовуючи недорогий процесор архітектури ARM Sitara XAM3359AZCZ100 Cortex A8 від компанії Texas Instruments.

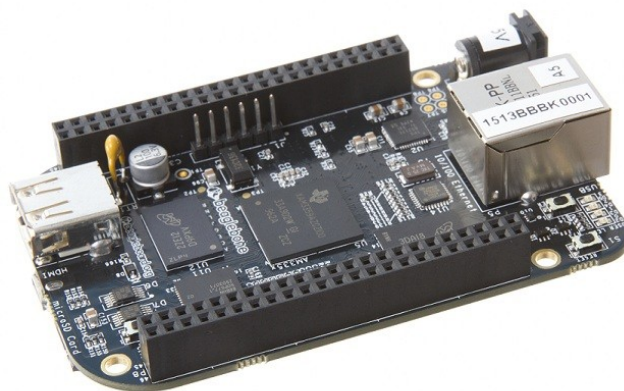


Рисунок 2. BeagleBone Black

Таб. 1. Порівняння одноплатних комп'ютерів

BeagleBone Black	Raspberry Pi 4
<ul style="list-style-type: none"> - Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz - 1GB, 2GB or 4GB LPDDR4 оперативної пам'яті - 2.4 GHz and 5.0 GHz IEEE 802.11ac wi-fi точки, Bluetooth 5.0 - Ethernet 1Гбіт/с 2 USB 3.0 порти; 2 USB 2.0 порти. Raspberry Pi стандартних 40 контактів GPIO 2 × micro-HDMI порт MIPI DSI порт для дисплея MIPI CSI порт для камери - Відео та стерео порти - MicroSD слот для завантаження операційної системи з зовнішнього носія - 5V DC вхід живлення через USB-C (3A) - 5V DC вхід живлення через GPIO (3A) - Живлення через Ethernet - Робочі температури: 0 – 50 градусів Цельсія 	<ul style="list-style-type: none"> - ARM Sitara XAM3359AZCZ100 Cortex A8, 1 Core (ARM v8). 32-bit SoC @ 1GHz - 512MB DDR3 оперативної пам'яті - 4GB 8-bit eMMC пам'яті для зберігання операційної системи - Ethernet 100Мбіт/с - Прискорювальний процесор з єдиною інструкцією - подвійний програмований блок в режимі реального часу - USB client порт для живлення та взаємодії з пристроєм - USB host порт - micro-HDMI порт - 2x 46 контактів GPIO - MicroSD слот для завантаження операційної системи з зовнішнього носія - 3V DC вихід живлення через GPIO - 5V DC вихід живлення через GPIO - 5V DC вхід для живлення пристрою (4 контакти на платі, 3A) - Робочі температури: 0 – 50 градусів Цельсія

BBB має трохи вищу ціну, аніж Raspberry Pi 4 (близько 60 доларів). Ціна за Raspberry Pi починається від 35 доларів за 1 Гб оперативної пам'яті, і до до 55 доларів за 4 Гб оперативної пам'яті. Raspberry Pi в цьому плані має переваги, особливо якщо враховувати достатню обчислювальну потужність, яку він надає своїми характеристиками. Навіть Raspberry Pi 3 B + за 35 доларів має кращі характеристики ніж BBB. Прот BBB - це справді міцна дошка, і зважаючи на її зосередженість на IoT, це розумна ціна.

Існує величезна кількість інформації у вигляді довідників та книжок, спільнот, які зацікавлені у використанні Від офіційного веб-сайту Raspberry Pi Foundation до підпорядкованих матеріалів, навчальних посібників та книг Raspberry Pi, доступний для вас запас ресурсів Raspberry Pi.

BBB, на фоні Raspberry Pi, виглядає більш скромно в плані об'ємів інформації, але все ще підтримує надійний сховище інформації. Зокрема, на BeagleBoard представлені процвітаючі дискусійні групи, навчальні посібники для початківців, та навіть повноцінні книги від розробників. Оскільки ви знайдете як офіційні ресурси спільноти BBB, так і сторонні пропозиції, це надзвичайно добре задокументована платформа.

Стосовно технічних характеристик - Raspberry Pi представляє більш потужні ресурси для виконання обчислень та збереження інформації. Хоча в BBB може не вистачити кількості портів USB, знайдених на Pi 4 та Pi 3 B +, у нього є USB для живлення та передачі даних, Ethernet та HDMI. Крім того, є цілих 92-контакти GPIO для реалізації підключення та взаємодії примітивної периферії, характерної для IoT проектів. На борту BBB має 32-розрядний мікроконтролер з подвійний програмований блок в режимі реального часу та 4 Гб пам'яті eMMC, що допомагає зберігати операційну систему на самому BBB та мати значно більший доступ до пам'яті, аніж Raspberry Pi. Таким чином, завдяки збільшенню вводу-виводу, BBB краще підходить для більш просунутих IoT проектів, які потребують підключення значної кількості сенсорів.

Кожен з вказаних одноплатних комп'ютерів має свої сильні сторони для конкретних задач. Raspberry Pi кращий з точки персонального комп'ютера чи сервера для обчислювальних задач, тоді як BBB більш зручний для проектів IoT напрямку.

1.2 Модуль GPRS/GSM зв'язку SIM800

GSM — глобальний стандарт цифрового мобільного стільникового зв'язку з розділенням каналів за часом і частоті. Розроблено під егідою Європейського інституту стандартизації електрозв'язку наприкінці 1980-х років.






GPRS [3] — надбудова над технологією мобільного зв'язку GSM, що здійснює пакетну передачу даних. GPRS дозволяє користувачеві мережі стільникового зв'язку здійснювати обмін даними з іншими пристроями в мережі GSM і із зовнішніми мережами, в тому числі Інтернет. GPRS передбачає тарифікацію за обсягом переданої / отриманої інформації, а не за часом, проведеним онлайн.

Мініатюрний модуль GSM/GPRS стільникового зв'язку на основі компонента **SIM800 [4][5]**, розробленого компанією SIMCom Wireless Solutions. Стандартний інтерфейс управління компонента SIM800, надає доступ до сервісів мереж GSM / GPRS 850/900/1800 / 1900МГц для відправки SMS-повідомлень, дзвінків та обміну цифровими даними GPRS.

Особливості SIM800:

- а) Чотирьохдіапазонний GSM/GPRS модуль, 850/900/1800/1900 МГц;
- б) Залежно від версії модему, інтерфейс USB для оновлення програмного забезпечення, UART, FM-radio, Bluetooth, PCM;
- в) Управління AT командами;
- г) Вбудований стек TCP/IP, UDP/IP, HTTP, FTP, Email, PING, MMC;

Таб. 2. Порівняння модулів SIM800

	 SIM800	 SIM800C	 SIM800H	 SIM800L	 SIM800F
Голосові виклики	+	+	+	+	+
Bluetooth 3.0	+	+	+	-	-
Вбудований FM-приймач:	-	-	+	+	-
CSD	+	-	+	+	-

В даній роботі буде використано модем SIM800 версії L, що базується на досить простій платі, головними особливостями якої є:

- 2 кріплення для антени (на базі SMA роз'єму, або піна NET);
- піни VCC — живлення, GND — земля;
- пін RST для запуску модему (замикання мінус — увімкнути, на плюс — вимкнути);
- піни RXD TXD, що передають дані на основі UART шини;
- додаткові піни для вводу/виводу звукового типу сигналу;
- пін DTR для реалізації в модемі режиму сну;
- пін RING для сповіщення про вхідні виклики;

Використовувати будемо лише NET, VCC, GND, RST, RXD та TXD для реалізації PPP протоколу, що дасть можливість обміну даних між BBB та модулем SIM800L.



Рисунок 3. Схема пінів SIM800

РОЗДІЛ 2.

ОПЕРАЦІЙНА СИСТЕМА LINUX, РЕАЛІЗАЦІЯ ДРАЙВЕРІВ ПРИСТРОЇВ

2.1 ОС Linux

Операційна система — це базовий комплекс програм, що виконує управління апаратною складовою комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організовує взаємодію з користувачем.

До **UNIX-подібних ОС** відноситься велика кількість операційних систем, котрі можна умовно поділити на три категорії — System V, BSD та Linux. Сама назва «UNIX» є торговою маркою, що належить «The Open Group», котра власне й ліцензує кожну конкретну ОС на предмет того, чи відповідає вона стандарту. Тому через ліцензійні чи інші неузгодження деякі ОС, котрі фактично є UNIX-подібними, не визнані такими офіційно.

Системи UNIX запускаються на великій кількості процесорних архітектур. Вони широко використовуються як серверні системи у бізнесі, як стільничні системи у академічному та інженерному середовищі. Тут популярні вільні варіанти UNIX, такі як Linux та БСД-системи. Окрім того, деякі з них останнім часом набувають широкого поширення в корпоративному середовищі, особливо це стосується орієнтованих на кінцевого користувача дистрибутивів Linux, в першу чергу Ubuntu, Mandriva, Red Hat Enterprise Linux та Suse. Linux також є популярною системою на стільницях розробників, системних адміністраторів та інших IT-спеціалістів.

Linux [6] (повна назва - GNU/Linux) - загальна назва UNIX-подібних операційних систем на основі однойменного ядра. Це один із найкращих прикладів розробки вільного (вільного) та відкритого (з відкритим кодом) програмного забезпечення (програмне забезпечення). Плюси операційної системи Linux:

- Безпека. Linux не може приховати свої недоліки. Його код переглядається багатьма експертами, котрі роблять свій внесок до кожного випуску нової версії ядра.
- Стабільність і надійність роботи.
- Модульність. Може включати тільки те, що система потребує, навіть під час роботи.
- Легко програмувати. Можливість вчитися на прикладі існуючого коду.
- Багато корисних ресурсів в інтернеті. Повна підтримка мережі.
- Портативність і апаратна підтримка. Запускається на більшості архітектур, через що набуває великої популярності на ринку смартфонів, планшетів.
- Масштабованість. Може працювати як на суперкомп'ютерах, так і на маленьких пристроях (достатньо 4 Мб оперативної пам'яті).

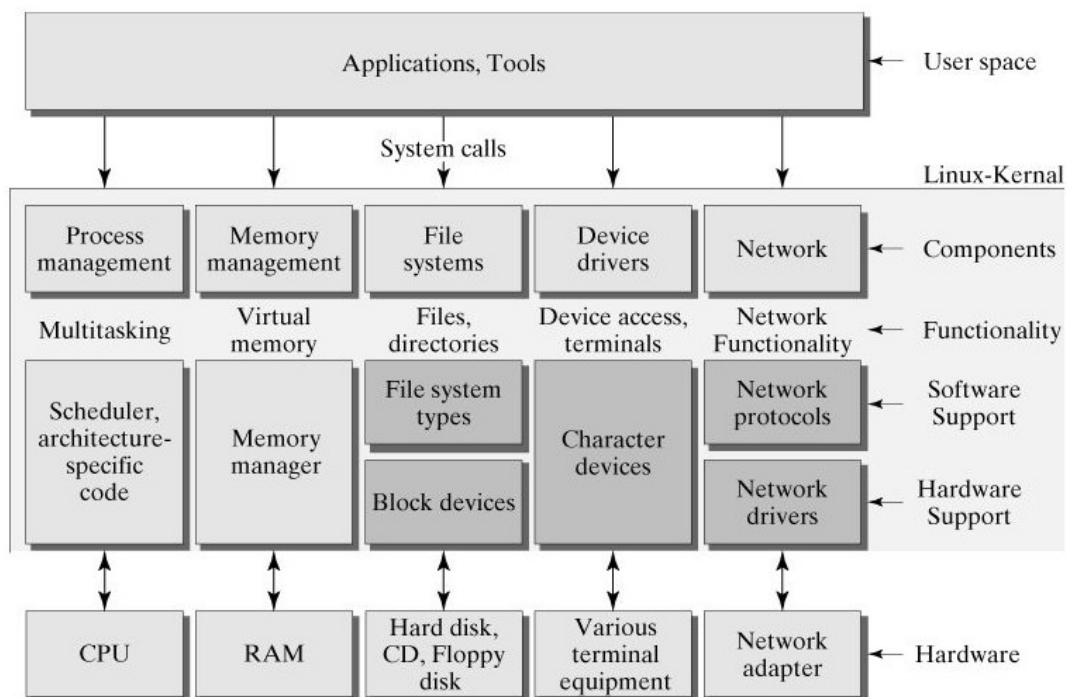


Рисунок 4. Архітектура ядра Linux. [7]

2.2 Проект Yocto для роботи з дистрибутивами Linux

Yocto Project [8] — робоча група організації Linux Foundation, яка розробляє і надає набір компонентів для створення власних дистрибутивів для вбудованих продуктів на базі різних апаратних архітектур, в тому числі ARM, PowerPC, MIPS, x86 і x86-64. Yocto не є окремим дистрибутивом, а надає розробникам вбудовуваних систем повний спектр рішень на базі існуючих готових компонентів, дозволяючи мінімізувати витрати на розробку прототипу системи, тим самим даючи можливість сфокусувати зусилля на процесі розробки та створення специфічних рішень та функціоналу для конкретного продукту. Пропонується кілька наборів для підтримки апаратних платформ для вбудованих платформ компаній Intel, NXP, Texas Instruments(як приклад BBB), Ubiquiti та інших.

До складу платформи входить інструментарій розробника, система складання, набір програмних інтерфейсів і колекція мета-пакетів. Набір метаданих і компонентів складання підтримується спільно з проектом OpenEmbedded. Наявний набір компіляторів, задіяний GCC. Існує додатковий функціонал на зразок Cross-Prelink, що дозволяє істотно прискорити завантаження програм, пов'язаних з великою кількістю бібліотек. Для спрощення розробки застосунків для платформ на базі Yocto існують плагіни для роботи з популярними IDE — для середовища розробки Eclipse і для Anjuta IDE, які підтримують розгортання проектів на віддалених системах, зневадження, аналіз коду, крос-компіляцію і використання емулятора QEMU.

Для складання задіяна система **Poky [9]**, що є відгалуженням від OpenEmbedded Build System і дозволяє об'єднати в рамках дистрибутиву розрізнені застосунки. Пакунки поширюються у форматі Red Hat Package Manager v5. Для контролю за інфраструктурою складання використовується програмне забезпечення Swabber, для виконання привілейованих операцій задіяний Pseudo, для організації автоматизованого тестування використовуються технології

Shoeleather Lab. Передбачена можливість генерації SDK, оптимізованого для продуктів, побудованих на базі Yocto.

2.3 Налаштування та компіляція ОС Linux та модулів на основі інструментарію Yocto

Так як робота виконана на основі одноплатного комп'ютера BBB, відповідно на прикладі BBB розглянемо налаштування та компіляцію ОС Linux та модулів під BBB на основі інструментарію Yocto.

Yocto використовує **мета-шари** [10] (meta-layers) для визначення конфігурації для складання системи. У кожному мета-шарі знаходяться рецепти, класи та файли конфігурації, які підтримують основний інструмент побудови python, bitbake.

BitBake [11] - це загальний механізм виконання завдань, який дозволяє ефективно та паралельно виконувати завдання оболонки та Python, працюючи в межах складних обмежень залежності між завданнями. Концептуально BitBake в деяких аспектах схожий на GNU Make, але має суттєві відмінності:

- а) BitBake виконує завдання відповідно до наданих метаданих, що створюють завдання. Метадані зберігаються у файлах рецептів (.bb), конфігурації (.conf) та класу (.bbclass) та надають BitBake інструкції щодо виконання завдань та залежностей між цими завданнями.
- б) BitBake включає в себе бібліотеку виборців для отримання вихідного коду з різних місць, таких як системи управління джерелами або веб-сайти.
- в) Інструкції для кожного блоку, який потрібно побудувати (наприклад, програмне забезпечення) відомі як файли рецептів і містять всю інформацію про пристрій (залежності, розташування вихідних файлів, контрольні суми, опис тощо).
- г) BitBake включає абстракцію клієнта/сервера і може використовуватися з командного рядка або використовуватись як сервіс по XMLRPC і має кілька різних інтерфейсів користувача.

Аналогічно для BBB існує свій мета-шар. Шар **meta-bbb** генерує деякі базові системи з пакетами, що підтримують такі мови програмування, як C, C++, Qt5, Perl та Python, що сильно розширює коло можливостей для подальшої роботи з meta-bbb.

Всі дії відбуваються на основі ОС Ubuntu 18.04 64-bit. Перш ніж розпочати роботу з Yocto, потрібно завантажити пакети, на зразок:

- a) build-essential;*
- б) chrpath;*
- в) diffstat;*
- г) gawk;*
- r) libncurses5-dev;*
- д) python3-distutils;*
- e) texinfo;*

Виконати налаштування:

```
~$ sudo dpkg-reconfigure dash
```

Для роботи з Yocto потрібен не лише основний репозиторій, а також вищезгадані мета-шари, що будуть доповнювати потрібним інструментарієм:

```
~$ git clone -b dunfell git://git.yoctoproject.org/poky.git poky-dunfell
```

Окремо завантажуюмо додаткові мета-шаблони, які дозволять отримати інструментарій для роботи з ОС Linux, для роботи з графічною оболонкою ОС, для захищеного доступу до інтернету та для налаштувань ОС Linux під потреби BBB.

```
~$ cd poky-dunfell
```

```
~/poky-dunfell$ git clone -b dunfell git://git.openembedded.org/meta-openembedded
```

```
~/poky-dunfell$ git clone -b dunfell https://github.com/meta-qt5/meta-qt5.git
```

```
~/poky-dunfell$ git clone -b dunfell git://git.yoctoproject.org/meta-security.git
```

```
~$ mkdir ~/bbb
```

```
~$ cd ~/bbb
```

```
~/bbb$ git clone -b dunfell git://github.com/jumpnow/meta-bbb
```

Наступний етап — ініціалізація build-директорію, де будуть зберігатися результати виконання команд bitbake. Для вказаного шляху файлової системи (**~/bbb/build/conf**) виконується налаштування середовища змінних, що використовуються командою bitbake для отримання інформації про характеристики платформи, під яку компілюється сконфігурована ОС Linux:

```
~$ source poky-dunfell/oe-init-build-env ~/bbb/build
```

Як результат виконаної команди — отримано 2 типів файлу, в яких зберігається інформація про налаштування процесу виконання команди bitbake (**build/conf/local.conf**) та інформація про мета-шаблони, що повинні бути включені у вказаний процес (**build/conf/bblayers.conf**). Після того, як будуть внесені зміни в **bblayers.conf**, а саме — додано нові мета-шари, що були завантажені, виконуємо команду:

```
~/bbb/build$ bitbake console-image
```

Результат цієї команди — бінарні файли (на зразок u-boot.img, MLO, uEnv.txt, rootfs.img, boot.img та інші), які ми в подальшому можемо використовувати для роботи з BBB.

А також, крім бінарів, ми отримали SDK для крос-компіляції програм користувацького простору:

```
~/bbb/build/tmp/deploy/sdk$ ls
```

```
poky-glibc-x86_64-meta-toolchain-cortexa8hf-neon-beaglebone-toolchain-3.0.2.host.manifest
```

```
poky-glibc-x86_64-meta-toolchain-cortexa8hf-neon-beaglebone-toolchain-3.0.2.sh
```

```
poky-glibc-x86_64-meta-toolchain-cortexa8hf-neon-beaglebone-toolchain-3.0.2.target.manifest
```

```
poky-glibc-x86_64-meta-toolchain-cortexa8hf-neon-beaglebone-toolchain-3.0.2.testdata.json
```

Саме *.sh скрипт із вказаного листу допоможе нам створити простір змінних для роботи з крос-компіляцією на основі GCC під архітектуру ARM.

2.4 Драйвери символьних пристроїв Linux [12]

У ОС Linux пристрої доступні користувачеві через спеціальні файли пристроїв. Ці файли згруповані в каталог `/dev`, з використанням системних викликів на зразок відкриття, читання, запис, закриття, `lseek`, `mmap` і т.д. Системні виклики перенаправляються ОС на драйвер, що пов'язаний з фізичним пристроєм. Драйвер пристрою - це компонент ядра (зазвичай модуль), який взаємодіє з пристроєм.

У світі UNIX існують дві категорії файлів пристроїв і, таким чином, драйвери пристроїв: символьні і блочні. Цей розподіл здійснюється за швидкістю, обсягом і способом організації даних, що передаються від пристрою до системи, і навпаки. В цій роботі використовуються лише символьні пристрої для реалізації драйверів.

До першої категорії відносять повільні пристрої, які керують невеликою кількістю даних, а доступ до даних не вимагає частого пошуку запитів. Прикладами є такі пристрої, як клавіатура, миша, послідовний порт, звукова карта, джойстик. Загалом, операції з цими пристроями (читання, запис та інші) виконуються послідовно байтом по байту.

Коротка характеристика символьного пристрою:

- Доступ до пристроїв здійснюється через імена у файловій системі;
- Ці імена називаються спеціальними файлами або файлами пристроїв або просто вузлами дерева файлової системи (зберігаються в `/dev`);
- Ядро Linux представляє символьні та блокові пристрої як пари чисел `<major>`: `<minor>`.
- Деякі `major` номери зарезервовані для певних драйверів символьних пристроїв;
- Інші `major` номери динамічно призначаються драйверу пристрою під час завантаження Linux (інформація з DTS) або при завантаженні драйвера;
- Пристрої одного і того ж `major` номера належать до одного класу;

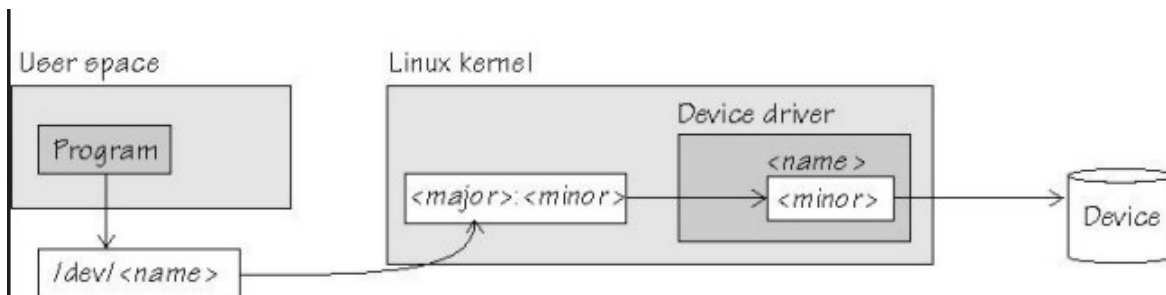


Рисунок 5. Схема взаємодії простору користувача з драйвером пристрою в проторі ядра Linux

Для символьних пристроїв існує структура, що зберігає вказівники на можливі методи, які може використовувати користувач для взаємодії з пристроєм:

- llseek: викликається, коли VFS потрібно перемістити індекс позиції файлу;
- read: викликається read (2) і пов'язаними системними викликами;
- read_iter: можливо асинхронне читання з iov_iter як призначення;
- write: викликається записом (2) і пов'язаними системними викликами;
- write_iter: можливо асинхронне записування з iov_iter як джерело;
- iterate: викликається, коли VFS потрібно прочитати вміст каталогу;
- poll: викликається VFS, коли процес хоче перевірити, чи є активність на; цьому файлі, і (за бажанням) перейти в режим сну, поки не буде вказаної в драйвері діяльності. Викликається системними викликами select (2) і poll (2);
- unlocked_ioctl: викликається системним викликом ioctl (2);
- compat_ioctl: викликається системним викликом ioctl (2), коли 32-бітні системні виклики використовуються на 64-бітових ядрах;
- mmap: викликається системним викликом mmap (2);
- open: викликається VFS при відкритті inode;
- flush: викликається системним викликом close (2) для видалення файлу;
- release: викликається після закриття останньої посилання на відкритий файл;
- fsync: викликається системним викликом fsync (2);
- fasync: викликається системним викликом fcntl (2), коли для файлу включений асинхронний (неблокуючий) режим;

Більшість з цих операцій не реалізовано у вказаних далі прикладах, а тому ми позначаємо відповідний вказівник на метод макросом на зразок `no_llseek`

Умови коректної реалізації структури драйвера:

Існують певні загальні правила написання драйверів під ОС Linux, що представляють із себе обмеження та нормативи, яких повинен дотримуватись кожен системний розробник:

- Драйвер повинен бути незалежним від платформи. Це означає, що в коді драйвера не повинно бути зафіксовано значення номерів GPIO, по яким відбувається підключення та отримання інформації;

- Дані пристрою отримуються з дерева пристроїв (DTC);

- Прив'язка драйверів – це автоматичний процес асоціації пристрою з відповідним драйвером, що працює з цим пристроєм;

GPIO — інтерфейс для зв'язку між компонентами комп'ютерної системи, наприклад, мікропроцесором і різними периферійними пристроями. Контакти GPIO можуть діяти і як входи, і як виходи, і це, як правило, підлягає налаштуванню. GPIO контакти часто групуються в порти. [13]

GPIO контакти не мають спеціального призначення і зазвичай залишаються невикористаними. Ідея полягає в тому, що іноді системному інтегратору для побудови повної системи, яка використовує чип, може виявитися корисним мати кілька додаткових ліній цифрового управління. З них можна організувати додаткові схеми, які інакше довелося б створювати з нуля.

Адреса портів пристроїв, що використовують переривання GPIO, повідомляються центральному процесору (CPU), і лише тоді процесор зможе їх використовувати.

Cape Expansion Headers

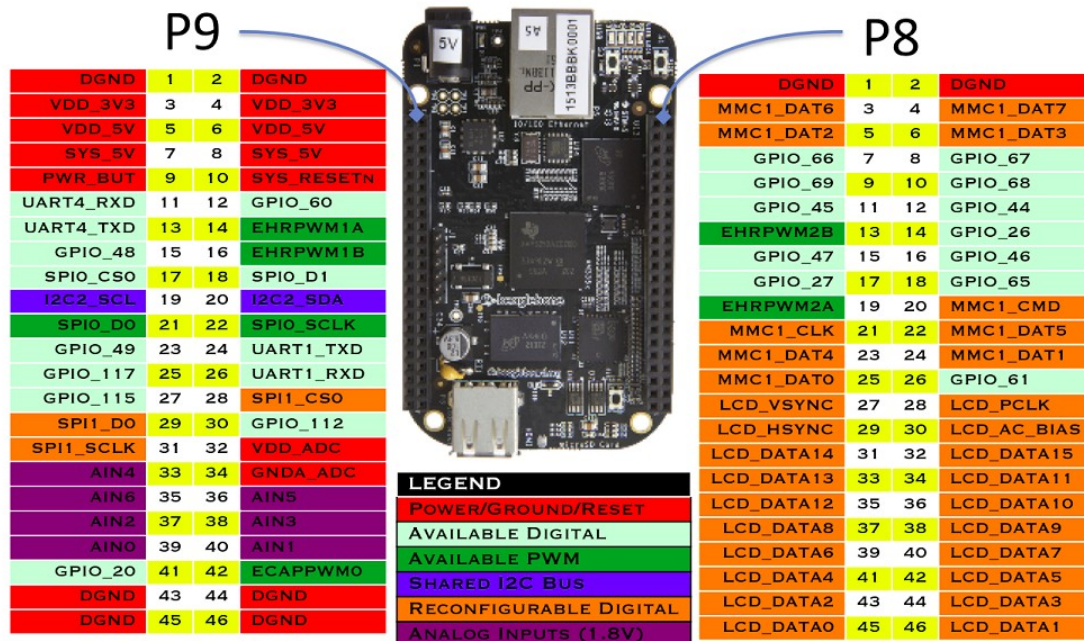


Рисунок 6. Схема GPIO для BBB [14]

Приклади реалізації драйверів, використаних у системі IoT:

Драйвер матричної клавіатури.

Опишемо основні принципи реалізації драйвера:

Всього вісім GPIO ліній може бути використано для роботи клавіатури. Для повноцінної роботи клавіатури, в драйвері реалізовано функціонал у вигляді трьох етапів:

Ініціалізація:

- виділення 4 ліній на сканування та 4 ліній на читання;
- встановлення всіх ліній сканування та читання на режим “вхід” (input);
- налаштування часу очікування на лініях читання (між 2 сигналами переривання на 1 лінії);

Опитування (цей процес починається, коли натискається будь-яка клавіша і створюється переривання):

- встановлення всіх ліній сканування на вхід (input);

- встановлення однієї лінії сканування в режим “виходу” (output);
- циклічне повторення для наступної лінії сканування;

Пошук переривання:

- Перевірка стану всіх ліній читання (відповідно до попереднього пункту, коли ми встановлюємо послідовно лінії сканування в output і зчитуємо їх значення стану, порівнюючи зі станом в попередньому циклі обробки переривання);
- Виявлення натиснутої кнопки (стан, порівняно з попередньою ітерацією – змінився);

Таблиця 3. GPIO-лінії BBB, що були використані для підключення

GPIO line	Pin name	BBB pin
gpio0_26	gpmc_ad10	P8.14
gpio0_27	gpmc_ad11	P8.17
gpio1_12	gpmc_ad12	P8.12
gpio1_13	gpmc_ad13	P8.11
gpio1_14	gpmc_ad14	P8.16
gpio1_15	gpmc_ad15	P8.15
gpio1_17	gpmc_a1	P9.23
gpio1_29	gpmc_csn0	P8.26

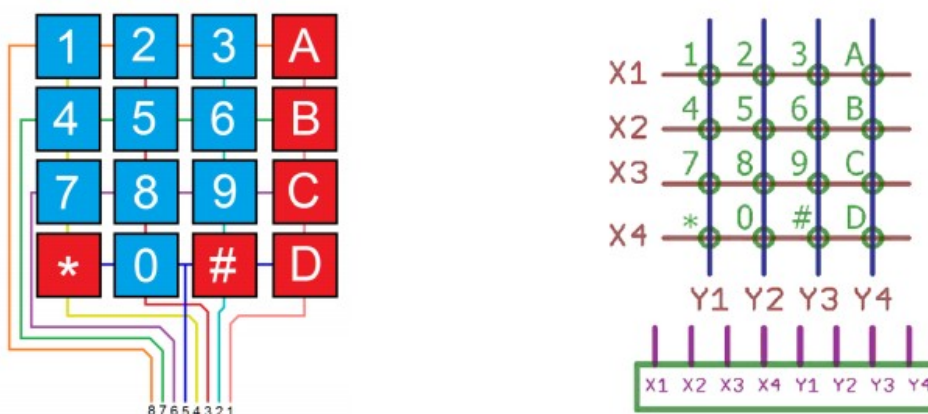


Рисунок 6. Загальний вигляд матричної клавіатури

DHT драйвер.

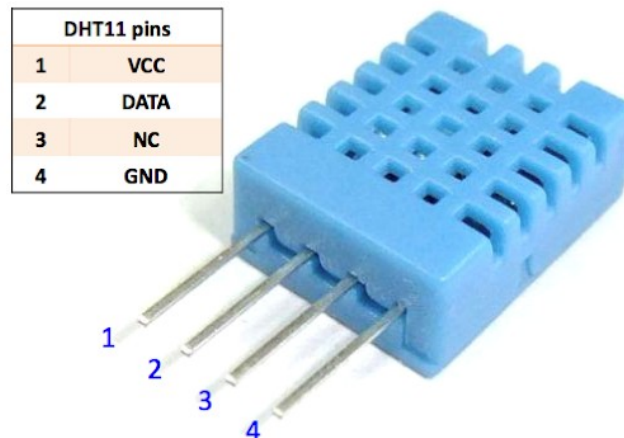


Рисунок 7. DHT11 датчик

DHT11 датчик [14, 15] — один із гарних прикладів датчиків вологості та температури повітря, який часто використовують у навчальних цілях, реалізаціях проектів розумного дому, аграрних автоматизованих проектів тощо. Датчик складається з двох частин - ємнісного датчика температури і гігрометра. Перший використовується для вимірювання температури, другий - для вологості повітря. Чіп, що знаходиться в середині корпусу, може виконувати аналого-цифрові перетворення і видавати цифровий сигнал, який зчитується та аналізується за допомогою ВВВ. DHT11 виводить калібрований цифровий сигнал. Він використовує ексклюзивну техніку збору цифрових сигналів і технологію зондування вологості, що забезпечує йому надійність та стабільність.

Невеликий розмір, мала ціна, низьке споживання електроенергії, мінімальні відхилення результатів дозволяють DHT11 бути зручним у будь-яких суворих випадках застосування.

Драйвер базується на тому, щоб у правильні проміжки часу вичитувати закодовані біти інформації, для подальшого декодування.

Біти даних кодуються як тривалість імпульсу у рядку даних:

- 0 біт: 22-30 нс, зазвичай 26 нс
- 1 біт: 68-75 нс, зазвичай 70 нс

Потрібно враховувати, що кабелі, по яким відбувається передача інформації від DHT11 до ВВВ, також має вплив на імпульси. Чим довший кабель — тим менше тривалість імпульсу, і як результат — тим важче його зареєструвати на боці ВВВ. Декодування імпульсів залежить від часової роздільної здатності системи (системного годинника). Найбільш оптимальні діапазони: від 34 нс і до 30 нс, або якщо роздільна здатність менше ніж 23 нс.

В драйвері реалізовано 2 основні функції, що відповідають за вичитування даних з самого датчика та декодування цих даних по принципу накопичування закодованих бітів, із яких декодується інформація по зміщенню про температуру, вологість та контрольна сума:

- *static int dht11_read_raw()* - функція, що перевіряє коректність пропускну здатності системного годинника, для подальшої ініціалізації отримання бітів даних, їх накопичування, та передачі, по зміщенню в буфері даних, на дешифрування.
- *dht11_decode()* - по вказаному зміщенню витягує 84 біти (з врахуванням похибки відхилення в плані періодичності імпульсів) із буфера для декодування значення температури (16 біт) , вологості (16 біт), контрольна сума (8 біт).

РОЗДІЛ 3.

РЕАЛІЗАЦІЯ КЛІЄНТ-СЕРВЕРНОГО ПІДКЛЮЧЕННЯ

3.1 PPP протокол підключення для використання SIM800

PPP — Point-to-Point protocol, проток точка-точка, канального рівня мережевої моделі OSI. Це механізм для створення і запуску IP — Internet Protocol і інших мережевих протоколів на послідовних лініях зв'язку, таких як прямий послідовний зв'язок по нуль-модемному кабелю, Ethernet, модемний зв'язок по телефонних лініях, радіозв'язок, зв'язок через GSM/3G/4G-модем. PPP дає нам чудову можливість реалізації передачі даних на сервер через модем SIM800.

Для того, що почати роботу з PPP на BBB, треба внести зміни в конфігурацію ядра ОС Linux, а також включити в дистрибутив пакет “ppp”. Тим самим буде внесено підтримку PPP протоколу.

Для конфігурації ядра достатньо перейти в **~/bbb/build** та виконати команду:

~/bbb/build\$ bitbake virtual/kernel -c menuconfig

Після чого буде отримано доступ до menuconfig, в якому потрібно вибрати наступні опції:

```
Device drivers --->
  [*] Network device support --->
    <M> PPP (point-to-point protocol) support
      <M> PPP BSD-Compress compression
      <M> PPP Deflate compression
      <*> PPP filtering
      <M> PPP MPPE compression (encryption)
      <*> PPP multilink support
      <M> PPP over Ethernet
      <M> PPP support for async serial ports
      <M> PPP support for sync tty ports
```

Рисунок 8. Конфігурація підтримки PPP в menuconfig

Зберігаємо конфігурацію.

Далі потрібно про-модифікувати **local.conf**, додавши в кінець файлу:

IMAGE_INSTALL_append = "ppp"

Після чого пере-збираємо дистрибутив і перевіряємо в історії виконання збірки наявність ppp пакету:

```
~/bbb/build$ cat buildhistory/images/beaglebone/glibc/core-image-base/installed-packages.txt | \ grep "ppp_"
```

Тепер в дистрибутиві є підтримка PPP.

Підключення між SIM800 та BBB реалізовано через 4 піни — живлення, “земля”, 2 піни для двонаправленої передачі даних. Для цього було використано піни:

- UART4_RXD (11, лінія P9);
- UART4_TXD (13, лінія P9);
- GND (1, P9);

Додатково: роль живлення для обох плат виконує Літій-іоні батареї сумарного об’єму 6600 мАг.

Саме UART4 шина буде надавати можливість реалізації PPP підключення.

Для цього у файловій системі BBB потрібно створити конфігураційний файл: “**/etc/ppp/options-mobile**” з вказаними значеннями:

```
UART4
921600
lock
crtscts
modem
passive
novj
defaultroute
noipdefault
usepeerdns
noauth
hide-password
persist
holdoff 10
maxfail 0
debug
```

Відповідно повинні бути проініціалізовані вказані файли в директорії **/etc/ppp/peers:**

```
mobile-noauth
provider -> mobile-noauth
```

Опис файлу **/etc/ppp/peers/mobile-auth:**

file /etc/ppp/options-mobile

connect "/usr/sbin/chat -v -t15 -f /etc/ppp/chatscripts/mobile-modem.chat"

Тепер перейдемо до основного скрипту: **/etc/ppp/chatscripts/mobile-modem.chat**, який веде діалог з модемом і належним чином виставляє інші критерії та налаштування для вибору APN, GPRS / 3G та PIN-коду. Цей скрипт інтерпретується обмеженим (але досить потужним) інструментом чату, що входить до стандартного пакету ppp.

Вказаним вище скрипт використовує додаткові файли на зразок:

/etc/ppp/chatscripts/apn.es.vodafone

/etc/ppp/chatscripts/pin.CODE

/etc/ppp/chatscripts/pin.NONE

/etc/ppp/chatscripts/mode.3G-only

/etc/ppp/chatscripts/mode.3G-pref

/etc/ppp/chatscripts/mode.GPRS-only

/etc/ppp/chatscripts/mode.GPRS-pref

/etc/ppp/chatscripts/mode.NONE

Вибір налаштувань залежить як від оператора зв'язку, так і від можливостей самого модему. Після того, як налаштування були виконані — виконання команд `pon/roff` відповідно дають нам керуванням демоном `pppd`. Після увімкнення можна перевірити результат виконання команд `pon`, переглянувши інформацію в **/var/log/messages**:

- Відповідно ініціалізація PPP інтерфейсу для обміну даних з SIM800L:

Jun 6 04:37:51 beaglebone daemon.debug pppd[1272]: Script /usr/sbin/chat -v -t15 -f /etc/ppp/chatscripts/mobile-modem.chat finished (pid 1273), status = 0x0

Jun 6 04:37:51 beaglebone daemon.info pppd[1272]: Serial connection established.

Jun 6 04:37:51 beaglebone daemon.debug pppd[1272]: using channel 1

Jun 6 04:37:51 beaglebone daemon.info pppd[1272]: Using interface ppp0

Jun 6 04:37:51 beaglebone daemon.notice pppd[1272]: Connect: ppp0 <--> /dev/ttyO4

- Налаштування адресації (отримання IP адреси) та перевірка доступу в інтернет:

Jun 6 04:37:52 beaglebone user.info kernel: [905.408115] PPP BSD Compression module registered

Jun 6 04:37:52 beaglebone user.info kernel: [905.445241] PPP Deflate Compression module registered

Jun 6 04:37:52 beaglebone daemon.debug pppd[1272]: sent [CCP ConfReq id=0x1 <deflate 15> <deflate(old#) 15> <bsd v1 15>]

Jun 6 04:37:52 beaglebone daemon.debug pppd[1272]: sent [IPCP ConfReq id=0x1 <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns2 0.0.0.0>]

Jun 6 04:37:52 beaglebone daemon.debug pppd[1272]: rcvd [IPCP ConfReq id=0x1 <addr 192.168.254.254>]

Jun 6 04:37:52 beaglebone daemon.debug pppd[1272]: sent [IPCP ConfAck id=0x1 <addr 192.168.254.254>]

Jun 6 04:37:52 beaglebone daemon.debug pppd[1272]: rcvd [LCP ProtRej id=0x0 80 fd 01 01 00 0f 1a 04 78 00 18 04 78 00 15 03 2f]

Jun 6 04:37:52 beaglebone daemon.debug pppd[1272]: Protocol-Reject for 'Compression Control Protocol' (0x80fd) received

Jun 6 04:37:53 beaglebone daemon.debug pppd[1272]: rcvd [IPCP ConfNak id=0x1 <addr 10.157.84.241> <ms-dns1 193.41.60.16> <ms-dns2 193.41.60.15>]

Jun 6 04:37:53 beaglebone daemon.debug pppd[1272]: sent [IPCP ConfReq id=0x2 <addr 10.157.84.241> <ms-dns1 193.41.60.16> <ms-dns2 193.41.60.15>]

Jun 6 04:37:53 beaglebone daemon.debug pppd[1272]: rcvd [IPCP ConfAck id=0x2 <addr 10.157.84.241> <ms-dns1 193.41.60.16> <ms-dns2 193.41.60.15>]

Jun 6 04:37:53 beaglebone daemon.notice pppd[1272]: local IP address 10.157.84.241

Jun 6 04:37:53 beaglebone daemon.notice pppd[1272]: remote IP address 192.168.254.254

Jun 6 04:37:53 beaglebone daemon.notice pppd[1272]: primary DNS address 193.41.60.16

Jun 6 04:37:53 beaglebone daemon.notice pppd[1272]: secondary DNS address 193.41.60.15

Jun 6 04:37:53 beaglebone daemon.debug pppd[1272]: Script /etc/ppp/ip-up started (pid 1280)

Jun 6 04:37:53 beaglebone daemon.debug pppd[1272]: Script /etc/ppp/ip-up finished (pid 1280), status = 0x0

Jun 6 04:37:54 beaglebone daemon.info ntpd[1218]: Listen normally on 2 ppp0 10.157.84.241:123

Лише після отриманих повідомлень можна вважати, що PPP протокол працює злагоджено, завдяки чому BBB має доступ до інтернету.

3.2 Реалізація клієнт-серверного підключення на основі SSH

SSH — мережевий протокол рівня застосунків, що надає можливість проведення віддаленого управління комп'ютера або тунелювання TCP - з'єднань для передачі файлів.

Основні функції, доступні при використанні SSH-протоколу:

- а) Передача будь-яких даних через захищене SSH-з'єднання, включаючи стиснення даних для подальшого шифрування.
- б) X11 Forwarding - механізм, що дозволяє запускати програми UNIX / Linux-сервера у вигляді графічної оболонки.
- в) Переадресація портів - передача шифрованого трафіку між портами різних машин.

Безпека SSH-з'єднання забезпечується:

- а) шифруванням даних одним з існуючих алгоритмів;
- б) аутентифікацією сервера і клієнта одним з декількох доступних методів;

- в) наявністю додаткових функцій протоколу, спрямованих на попередження різних атак

Аутентифікація сервера дає захист від:

- а) злому шляхом підміни IP-адрес (IP-spoofing), коли один віддалений хост посилає пакети від імені іншого віддаленого хоста;
- б) підміни DNS-записів (DNS-spoofing), коли підміняється запис на DNS-сервері, в результаті чого з'єднання встановлюється з хостом, який вказаний в підміненої записи, замість необхідного;
- в) перехоплення відкритих паролів і інших даних, що передаються у відкритому вигляді через встановлений з'єднання;

Для реалізації SSH підключення використовується OpenSSH — це провідний інструмент підключення для віддаленого входу з протоколом SSH. OpenSSH пропонує великий набір можливостей безпечної тунелювання, кілька методів аутентифікації та складні параметри конфігурації. sshd (OpenSSH Daemon or Server) — програма-демон для клієнта ssh. Це безкоштовний і відкритий ssh-сервер. ssh замінює небезпечні rlogin і rsh і забезпечує безпечну зашифровану зв'язок між двома ненадійними вузлами через небезпечну мережу, таку як Інтернет.

Для реалізації серверної частини скористаємось вільно-доступним ресурсом SDF.ORG. SDF, або Super Dimension Fortress - це неприбуткова компанія, що надає доступ до оболонки UNIX в Інтернеті. Він працює в постійній діяльності з 1987 року як некомерційний соціальний клуб. SDF являється багатофункціональним постачальником різного типу утиліт, програм, інформації та документації, який обслуговує членів у всьому світі.

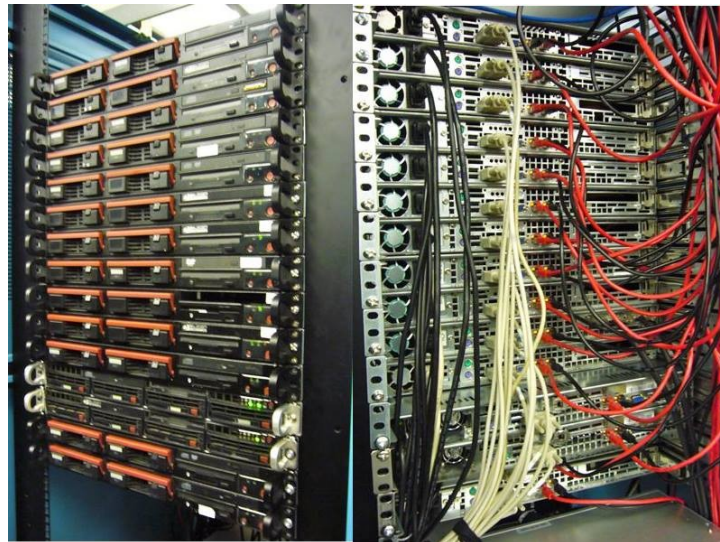


Рисунок 9. Сердце SDF

Після проходження процедури реєстрації на вказаному ресурсі можна виконати підключення для перевірки доступу до SSH серверу, наданого для ваших потреб:

```
root@beaglebone:~# ssh nikita1opanyuk@tty.sdf.org
```

```
The authenticity of host 'tty.sdf.org (205.166.94.9)' can't be established.
```

```
ED25519 key fingerprint is:
```

```
SHA256:ZjwbO7AU8rHJExYrmZS2LqGZ7WfdoELfMrF54W92PYA.
```

```
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added 'tty.sdf.org,205.166.94.9' (ED25519) to the list of known hosts.
```

```
Connection closed by 205.166.94.9 port 22
```

```
root@beaglebone:~#:$ ssh nikita1opanyuk@tty.sdf.org
```

```
nikita1opanyuk@tty.sdf.org's password:
```

```
*
```

```
* Wed Jun 3 23:41:54 UTC 2020
```

```
*
```

```
* For webmail authentication pair, type 'webmail'
```

```
*
```

```
[ 'nikita1opanyuk' will expire in 665 days - Please 'validate' your account soon ]  
Please press your BACKSPACE key:
```

Тим самим переконавшись в доступі на сервер, можна використовувати програмне забезпечення **client_reader**. Так як зі сторони клієнта, в дистрибутиві BBB, присутня підтримка ssh server and client, відповідно реалізовано застосунок, який має такий основний функціонал:

- ***int read_information_from_dev(int *fd, unsigned int dev_cnt, char *info);*** — функція, яка отримує на вхід масив файлових дескрипторів драйверів символьних пристроїв для вчитування даних про статус пристроїв та масив info для збереження вчитаної інформації;
- ***int scp_info_file(ssh_session session, ssh_scp scp, char *info);*** — функція, яка (на базі використання SCP протоколу) отримує на вхід проініціалізовану сесію SSH та SCP сесію для створення запису строки info у файл, що знаходиться на віддаленому сервері;
- ***int ssh_session_init(ssh_session cur_ssh_session);*** — функція, ініціалізує SSH сесію між BBB та сервером *nikitaIopanyuk@tty.sdf.org*;
- ***int scp_write_init(ssh_session session, ssh_scp scp_session);*** — функція, яка ініціалізує SCP процес з параметрами запису на сервер на базі проініціалізованої сесії SSH;

Завдяки вказаному функціоналу на сервері відбувається створення директорії з файлом info.txt, в який з певною періодичністю відбувається запис вчитаної з файлових дескрипторів інформації про статус підключених до BBB пристроїв.

ВИСНОВКИ

Так як метою роботи є створення IoT програмного забезпечення для налаштування та реалізації взаємодії функціональних блоків комп'ютера між собою та операційною системою. для можливості отримання інформації з пристроїв для подальшої передачі користувачеві, а тому як результат роботи:

- було розглянуто причини використання саме операційної системи Linux, її плюси та мінуси, а також особливості та можливості інструментарію Yocto Project для роботи з дистрибутивами ОС Linux;
- реалізовано приклади драйверів (драйвер кнопки, драйвер матричної клавіатури, драйвер DHT датчика) на основні символічних пристроїв, як варіант драйверів для отримання користувацьким простором інформації;
- виконано налаштування дерева приладів ОС Linux для коректної роботи драйверів;
- створено додаткове програмне забезпечення та набір скриптів для налаштування взаємодії основної плати з модулем SIM800L для отримання доступу до інтернету та надсиланням даних на SDF сервер через SSH підключення;

Все це – лише мала частина можливостей реалізації драйверів, їх взаємодії, та методів для передачі даних через мережу Інтернет. IoT напрямок реалізацій програмного забезпечення під операційну систему Linux буде продовжувати свій розвиток, покращуючи методи для роботи різних типів пристроїв, спрощуючи інтерфейси для роботи з різними типами підключень, створюючи нові інтерфейси користувача для взаємодії з драйверами – більш оптимальні та зручні, використовуючи більш надійні протоколи передачі даних через мережу.

Разом з цим буде рости потреба в спеціалістах-розробниках як і системної частини, так і програмного забезпечення користувацького простору. Завдяки цьому досить актуально розглядати цей напрямок програмування серед навчальних програм факультетів, спеціалізованих в комп'ютерних науках.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. "Implementing High Performance Embedded Computing Hardware" / [авт. тексту предс. Trenton Systems] Trenton Systems, Inc., 2016. 13–15 с.
2. Gareth Halfacree. "Raspberry Pi User Guide", 4th Edition: довідник основ. хар. пристр. для користувачів / Gareth Halfacree, Eben Upton – К.: Raspberry Pi & BBC Micro, 2016. – 293 с.
3. Олексій Лукацький. СІТ форум, онлайн бібліотека [Електронний ресурс] : журнал "Мобільні системи" / Олексій Лукацький – К.: Науково-інженерне підприємство "Інформзахист", №2, 2018. Режим доступу до журн. : http://citforum.ru/security/articles/sec_GPRS/
4. Sim Com. "SIM800H/L Hardware Design V2.01" : PDF документація про основні особл. модуля SIM800H/L / Sim Com – К.: компанія SIM Tech, 2014. – 72 с.
5. Кравченко Виктор. codius, збірник статей [Електронний ресурс] : GSM-модуль SIM800L: повний мануал / Кравченко Виктор, 2018. Режим доступу до статті : http://codius.ru/articles/GSM_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_SIM800L_%D1%87%D0%B0%D1%81%D1%82%D1%8C_1
6. Роберт Лав. Ядро Linux, Опис процесів розробки, 3-те видання: вичерпне керівн. про проектування і реалізації ядра Linux / Роберт Лав – К.: Pearson Education, пер. з англ. - М.: ООО "І.Д. Віл'ямс" 2013. – 496 с.
7. Asish Vara. Engine Garage, An EE World Online Resource. [Електронний ресурс] : Інформація про архітектуру ядра Linux // 2016. Режим доступу до статті: <https://www.engineersgarage.com/tutorials/introduction-linux-part-715>
8. Офіційний сайт Yocto Project [Електронний ресурс] : Оф. сайт для ознайомлення, обміну інф. і т.д. – К.: Yocto Project, a Linux Foundation Collaborative Project. 2020. Режим доступу до ресурсу: <https://www.yoctoproject.org/>
9. Оф. сайт-репозиторій для Yocto Project. [Електронний ресурс] : Оф. репозиторій для ознайомлення із інструментом Poky. – К.: Yocto Project, a Linux Foundation

Collaborative Project. 2020. Режим доступу до реп.:

<https://git.yoctoproject.org/cgit/cgit.cgi/poky/>

10. Otavio Salvador. “Embedded Linux Development using Yocto Project”, Second Edition : PDF довідник для вивчення потужностей та можливостей Yocto Project / Otavio Salvador, Daiane Angolini – К.: Packt — 2014. 150 с.

11. Yocto Project, a Linux Foundation Collaborative Project. [Електронний ресурс] : / Richard Purdie, Chris Larson, and Phil Blundell // Довідник по використ. Bitbake / 2012. Режим доступу до ресурсу:

<https://www.yoctoproject.org/docs/1.6/bitbake-user-manual/bitbake-user-manual.html>

12. “Linux API”. [Електронний ресурс] : Документація по Linux API симв. пристроїв. Режим доступу до ресурсу:

https://linux-kernel-labs.github.io/refs/heads/master/labs/device_drivers.html

13. “Components 101”. [Електронний ресурс] : Документація по осн. хар. BBB, схематика. Режим Доступу до блогу:

<https://components101.com/microcontrollers/beaglebone-black-pinout-datasheet>

14. “Digital-output relative humidity & temperature sensor/module – DHT11” : PDF документ. Заг інф. про датчик DHT11 / 2020. 3с.

15. Adafruit learning system. “DHT Humidity Sensing on Raspberry Pi or Beaglebone Black with GDocs Logging” : PDF документ. загальні приклади використання датчику DHT11. Adafruit learning system – К.: Adafruit Industries. 2019. 20.с

16. Credentiality Blog [Електронний ресурс] : Інформація про роз'єм P8 P9 плати BeagleBone Black, 2015. Режим доступу до блогу: <http://credentiality2.blogspot.com/2015/09/beaglebone-pru-gpio-example.html>

ДОДАТОК А

Реалізація драйвера матричної клавіатури

Основна частина драйверу, що відповідає за три етапи аналізу отриманого переривання:

```
static void matrix_keypad_scan(struct work_struct *work) {
/* макрос container_of витягує потрібну нам для обробки інформацію */
    struct matrix_keypad *keypad =
        container_of(work, struct matrix_keypad, dwork.work);
    int row, col;
    unsigned int i;
    int *new_cols_state =
        (int*)kzalloc(keypad->num_col_gpios*sizeof(int),GFP_KERNEL);
/* Деактивація ліній сканування – переводимо в режим input */
    activate_all_cols(keypad, 0);
    memset(new_cols_state, 0, sizeof(int) * keypad->num_col_gpios);
/* Активуючи по черзі лінії сканування, перевіряємо чи є зміни відповідно на
лініях для читання. Якщо зміни є – зберігаємо це з врахуванням того яка саме
лінія для читання змінила свій стан */
    for (col = 0; col < keypad->num_col_gpios; col++) {
        /* Перехід лінії сканування в output для перевірки стану */
        activate_col(keypad, col, 1);
        /* Після відновлення роботи відповідної лінії сканування –
зчитуємо стан ліній зчитування */
        for (row = 0; row < keypad->num_row_gpios; row++)
            new_cols_state[col] |=
                row_asserted(keypad, row) ? (1 << row) : 0;
        /* Переводимо лінію сканування в режим input */
    }
}
```

```

        activate_col(keypad, col, 0);
    }
    /* Порівняння нового стану клавіатури (а саме стану ліній сканування) з
    попереднім станом – якщо ж присутні зміни – виводимо яка клавіша була
    натиснута */
    for (col = 0; col < keypad->num_col_gpios; col++) {
        /* Перевірка зміни відповідної лінії сканування */
        u32 changing = new_cols_state[col] ^ keypad->cols_state[col];
        /* Якщо змін немає – продовжуємо з нової ітерації */
        if (changing == 0)
            continue;
        /* Якщо зміни є – перевіряємо на якій саме лінії зчитування */
        for (row = 0; row < keypad->num_row_gpios; row++) {
            if ((changing & (1 << row)) == 0)
                continue;
            check_what_output(row, col);
        }
    }
    /* Зберігаємо новий стан клавіатури */
    memcpy(keypad->cols_state, new_cols_state,
           sizeof(int) * keypad->num_col_gpios);
    /* Після завершення перевірки на зміну стану – відновлення всіх GPIO лінії
    сканування в режим input */
    activate_all_cols(keypad, 1);
    /* Змінення статусу обробки стану клавіатури, відновлення переривання для
    наступних ітерацій */
    spin_lock_irq(&keypad->lock);
    keypad->scan_pending = 0;

```

/ Відновлення переривань на лініях зчитування */*

```
for (i = 0; i < keypad->num_row_gpios; i++)
    enable_irq(keypad->irq_rows[i]);
kfree(new_cols_state);
```

Вказана функція використовується являється обробником для черги задач:

INIT_DELAYED_WORK(&keypad->dwork, matrix_keypad_scan);

Яка заповнюється у випадку виникнення будь яких зареєстрованих переривань на лініях зчитування:

```
queue_delayed_work(keypad->work_queue, &keypad->dwork,  
    msec_to_jiffies(QUEUE_DELAY_MS));
```

ДОДАТОК Б

Реалізація драйвера датчика DHT11

Основна частина драйверу DHT11, що відповідає за вчитування та декодування отриманих даних з пристрої:

/ допоміжна функція для конвертування масиву бітів в байт */*

static unsigned char dht11_decode_byte(char *bits);

/ основна функція декодування розділених за часом даних, що зберігаються в масиві переходів сигналів dht11->edges, для отримання інформації про температуру, вологість та контрольної суми, яка допомагає нам переконатись, що дані не були втрачені на шляху до BBB */*

static int dht11_decode(struct dht11 *dht11, int offset)

{

int i, t;

char bits[DHT11_BITS_PER_READ];

unsigned char temp_int, temp_dec, hum_int, hum_dec, checksum;

/ З врахуванням роздільної здатності системного годинника, часом, коли було вчитано сигнали, та отриманих даних з лінії DATA, перевіряється відсутність порушень в плані синхронізації та заповнюється масив бітів результату вчитування на основі порівняння сусідніх сигналів */*

for (i = 0; i < DHT11_BITS_PER_READ; ++i) {

t = dht11->edges[offset + 2 * i + 2].ts -

dht11->edges[offset + 2 * i + 1].ts;

if (!dht11->edges[offset + 2 * i + 1].value) {

dev_dbg(dht11->dev,

"lost synchronisation at edge %d\n",

offset + 2 * i + 1);

return -EIO;

}

bits[i] = t > DHT11_THRESHOLD;

```
}
```

/* Розділення отриманого масиву бітів на окремі результати для декодування, перевірки контрольної суми та заповнення результату для передачі користувачеві через дескриптор ПО (Industrial input-output, різновид API для роботи з символьними пристроями) пристрою */

```
hum_int = dht11_decode_byte(bits);
hum_dec = dht11_decode_byte(&bits[8]);
temp_int = dht11_decode_byte(&bits[16]);
temp_dec = dht11_decode_byte(&bits[24]);
checksum = dht11_decode_byte(&bits[32]);

if (((hum_int + hum_dec + temp_int + temp_dec) & 0xff) != checksum) {
    dev_dbg(dht11->dev, "invalid checksum\n");
    return -EIO;
}

dht11->timestamp = ktime_get_boottime_ns();
if (hum_int < 4) { /* DHT22: 100000 = (3*256+232)*100 */
    dht11->temperature = (((temp_int & 0x7f) << 8) + temp_dec) *
                        ((temp_int & 0x80) ? -100 : 100);
    dht11->humidity = ((hum_int << 8) + hum_dec) * 100;
} else if (temp_dec == 0 && hum_dec == 0) { /* DHT11 */
    dht11->temperature = temp_int * 1000;
    dht11->humidity = hum_int * 1000;
} else {
    dev_err(dht11->dev,
            "Don't know how to decode data: %d %d %d %d\n",
            hum_int, hum_dec, temp_int, temp_dec);
    return -EIO;
}
```

```

    return 0;
}

/* Обробник переривань, який заповнює масив переходів сигналу в врахування
системного часу до тих пір, поки не буде вчитано достатньої кількості сигналу
для процесу декодування */
static irqreturn_t dht11_handle_irq(int irq, void *data)
{
    struct iio_dev *iio = data;
    struct dht11 *dht11 = iio_priv(iio);

    if (dht11->num_edges < DHT11_EDGES_PER_READ && dht11->num_edges
    >= 0) {
        dht11->edges[dht11->num_edges].ts = ktime_get_boottime_ns();
        dht11->edges[dht11->num_edges++].value =
            gpiod_get_value(dht11->gpiod);
        /* Перевірка того скільки переходів сигналу було вчитано */
        if (dht11->num_edges >= DHT11_EDGES_PER_READ)
            complete(&dht11->completion);
    }

    return IRQ_HANDLED;
}

/* Основна функція, що перевіряє відповідно роздільну здатність системного
таймера, ініціалізує обробник переривань для вчитування переходів сигналу (як
знизу-вверх, так і зверху-вниз), на базі отриманого масиву переходів робить
декодування і у випадку успішного виконання повертає користувачеві (через
звернення, як приклад, командою read() до файлового дескриптора драйвера)
відповідні показники температури та вологості */

```



```

static int dht11_read_raw(struct iio_dev *iio_dev,
                        const struct iio_chan_spec *chan,
                        int *val, int *val2, long m)
{
    struct dht11 *dht11 = iio_priv(iio_dev);
    int ret, timeres, offset;

    /* Перевірка роздільної здатності таймера */
    mutex_lock(&dht11->lock);
    if ((dht11->timestamp + DHT11_DATA_VALID_TIME <
ktime_get_boottime_ns()) {
        timeres = ktime_get_resolution_ns();
        dev_dbg(dht11->dev, "current timeresolution: %dns\n", timeres);
        if (timeres > DHT11_MIN_TIMERES) {
            dev_err(dht11->dev, "timeresolution %dns too low\n",
                timeres);
            ret = -EAGAIN;
            goto err;
        }
        if (timeres > DHT11_AMBIG_LOW && timeres <
DHT11_AMBIG_HIGH)
            dev_warn(dht11->dev,
                "timeresolution: %dns - decoding ambiguous\n",
                timeres);

        /* Ініціалізація структури завершення виконання роботи обробника
переривань */
        reinit_completion(&dht11->completion);

        /* Ініціалізація GPIO лінії для отримання даних з DHT11 драйвера */
        dht11->num_edges = 0;

```

```

ret = gpiod_direction_output(dht11->gpiod, 0);
if (ret)
    goto err;
usleep_range(DHT11_START_TRANSMISSION_MIN,
             DHT11_START_TRANSMISSION_MAX);
ret = gpiod_direction_input(dht11->gpiod);
if (ret)
    goto err;

ret = request_irq(dht11->irq, dht11_handle_irq,
                 IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING,
                 iio_dev->name, iio_dev);
if (ret)
    goto err;

/* Очікування завершення вичитування інформації */
ret = wait_for_completion_killable_timeout(&dht11->completion,
                                           HZ);

free_irq(dht11->irq, iio_dev);
if (ret == 0 && dht11->num_edges < DHT11_EDGES_PER_READ - 1) {
    dev_err(dht11->dev, "Only %d signal edges detected\n",
           dht11->num_edges);
    ret = -ETIMEDOUT;
}
if (ret < 0)
    goto err;

/* Декодування отриманої інформації */
offset = DHT11_EDGES_PREAMBLE +
         dht11->num_edges - DHT11_EDGES_PER_READ;

```

```

    for (; offset >= 0; --offset) {
        ret = dht11_decode(dht11, offset);
        if (!ret)
            break;
    }

    if (ret)
        goto err;
}

/* Передача користувачеві, в залежності від того з яким значенням
параметру const struct iio_chan_spec *chan було виконано звернення користувача,
отриманої інформації */
ret = IIO_VAL_INT;
if (chan->type == IIO_TEMP)
    *val = dht11->temperature;
else if (chan->type == IIO_HUMIDITYRELATIVE)
    *val = dht11->humidity;
else
    ret = -EINVAL;
err:
dht11->num_edges = -1;
mutex_unlock(&dht11->lock);
return ret;
}

```

ДОДАТОК В

Функціонал клієнтської частини для надсилання даних на сервер

```

int ssh_session_init(ssh_session cur_ssh_session) {
    int rc = SSH_OK;
    char *info = NULL;
    /* Створення нової сесії SSH */
    cur_ssh_session = ssh_new();
    if (NULL == cur_ssh_session)
        exit(-1);
    /* Налаштування сесії SSH для підключення до SDF сервера */
    ssh_options_set(cur_ssh_session, SSH_OPTIONS_HOST,
        "nikitaIopanyuk@tty.sdf.org");
    /* Підключення до сесії SSH */
    rc = ssh_connect(my_ssh_session);
    if (SSH_OK != rc) {
        fprintf(stderr, "Error connecting to localhost: %s\n",
            ssh_get_error(my_ssh_session));
        exit(-1);
    }
    return rc;
}

int scp_write_init(ssh_session session, ssh_scp scp_session) {
    ssh_scp scp;
    int rc = SSH_OK;
    /* Декодування отриманої інформації */
    scp = ssh_scp_new(session,
        SSH_SCP_WRITE | SSH_SCP_RECURSIVE, ".");
    if (NULL == scp) {
        printf("Error allocating scp session: %s\n",

```

```

        ssh_get_error(session));
    return SSH_ERROR;
}

/* Декодування отриманої інформації */
rc = ssh_scp_init(scp);
if (SSH_OK != rc) {
    printf("Error initializing scp session: %s\n",
        ssh_get_error(session));
    ssh_scp_free(scp);
    return rc;
}
return rc;
}

int scp_info_file(ssh_session session, ssh_scp scp, char *info) {
    int rc = SSH_OK;
    int length = strlen(info);

    /* Створення віддаленої директорії на сервері SSH для зберігання файлу */
    rc = ssh_scp_push_directory(scp, "info_from_client", S_IRWXU);
    if (SSH_OK != rc) {
        printf("Can't create remote directory: %s\n",
            ssh_get_error(session));
        return rc;
    }

    /* Створення віддаленого файлу */
    rc = ssh_scp_push_file(scp, "info.txt",
        length, S_IRUSR | S_IWUSR);
    if (SSH_OK != rc) {
        printf("Can't open remote file: %s\n",
            ssh_get_error(session));
    }
}

```

```

        return rc;
    }

    /* Виконання запису на віддалений файл масиву info */
    rc = ssh_scp_write(scp, info, length);
    if (SSH_OK != rc) {
        printf("Can't write to remote file: %s\n",
            ssh_get_error(session));
        return rc;
    }
    return rc;
}

int read_information_from_dev(int *fd, unsigned int dev_cnt, char *info) {
    int rc = EXIT_FAILURE;
    char *buffer = (char *)calloc(BUF_SIZE, sizeof(char));
    /* вчитуємо з кожного пристрою дані та записуємо в масив info */
    for (unsigned int i = 0; i < COUNT; ++i) {
        for (unsigned int j = 0; j < dev_cnt; j++) {
            memset(buffer, 0, BUF_SIZE);
            rc = read(fd[j], buffer, BUF_SIZE);
            if (rc == -1) {
                perror("read() error");
            } else {
                printf("read: %s\n", buffer);
            }
            strcat(info, buffer);
        }
    }
    return rc;
}

```