НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2021 р.

Дипломний проект
на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: IoT пристрій на базі мікроконтролера STM32

Виконав (-ла): студент (-ка) __IV__ курсу, групи __ІО-72__
(шифр групи)

_____Музика Михайло Олександрович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____к.т.н. Ткаченко В.В._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н. Сімоненко В. П. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент
(підпис)

Київ – 2021 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальність 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2021 р.

–

ЗАВДАННЯ
на бакалаврський дипломний проект студента

_____ *Музики Михайла Олександровича*_____

1. Тема проекту IoT пристрій на базі мікроконтролера STM32
керівник проекту Ткаченко Валентина Василівна, доцент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 11 травня 2021 року №1139-с

2. Термін здачі студентом закінченого проекту _____*01 червня 2021 р.*

3. Вихідні дані до проекту *технічна документація. теоретичні дані*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис складових пристрою, актуальність розробки, поглиблене дослідження теорії використовуваних модулів, розробка принципової схеми, розробка алгоритму програми, розробка електричної функціональної схеми, розробка пристрою і прив’язування мобільного телефону.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень)
Принципова схема, схема алгоритму програми, електрично-функціональна схема.

6. Консультанта проекту, з вказівкою розділів проекту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проекту	Терміни виконання етапів проекту	Примітки
1.	*Затвердження теми проекту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем*	*25.03.2021-5.04.2021*	
5.	*Програмна реалізація системи*	*5.04.2021-15.04.2021*	
6.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
7.	*Захист програмного продукту*	*25.04.2021*	
8.	*Передзахист*	*23.05.2021*	
9.	*Захист*	*14.06.2021*	

Студент Музика Михайло

Керівник Валентина Ткаченко

Анотація

В бакалаврській дипломній роботі був реалізований IoT пристрій на базі мікроконтролера STM32 зі спряженням з ним мобільного телефону.

Система дозволяє запрошувати дані від плати, використовуючи технологію bluetooth. Програмний продукт складається із двох частин, мікроконтролера оснащеного датчиками вимірювання температури, освітленості та тиску та розробленої програми на мові програмування C.

Annotation

In the bachelor's thesis was implemented IoT device based on STM32 microcontroller with pairing with a mobile phone.

The system allows you to request data from the board using bluetooth technology. The software product consists of two parts, a microcontroller equipped with sensors for measuring temperature, light and pressure and a program developed in the C programming language.

ОПИС АЛЬБОМУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1			IoT пристрій на базі	3	
2			мікроконтролера STM32		
3	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
4			IoT пристрій на базі		
5			мікроконтролера STM32		
6	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	51	
7			IoT пристрій на базі		
8			мікроконтролера STM32.		
9	А4	ІАЛЦ.467200.004 Е1	Принципова схема.	1	
10			Процедура отримання інформації		
11			через bluetooth		
12	А4	ІАЛЦ.467200.005 Е2	Схема алгоритму програми	1	
13			Налаштування тактових частот		
14			для STM32		
15	А4	ІАЛЦ.467200.006 Е3	Схема електрична функціональна	1	
16	А4	ІАЛЦ.467200.007	Лістинг програми	8	
17					
18					
19					
20					
21					
22					
23					
24					

Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Музика М.О.			*IoT пристрій на базі мікроконтролера STM32* Відомість дипломного проєкту	Літ.	Арк. 1 / Акрушів
Перевір.		Ткаченко В.В.					
Н. Контр.		Сімоненко В.П.				КПІ ім. Ігоря Сікорського ФІОТ ІО-72	
Затверд.		Стіренко С.Г.					

25					
26					
27					

Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Музика М.О.			*IoT пристрій на базі мікроконтролера STM32* Відомість дипломного проєкту	Літ. / Арк. 1 / Акрушів
Перевір.		Ткаченко В.В.				
						КПІ ім. Ігоря Сікорського ФІОТ ІО-72
Н. Контр.		Сімоненко В.П.				
Затверд.		Стіренко С.Г.				

ТЕХНІЧНЕ ЗАВДАННЯ

Технічне завдання

Зміст

1.НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання поширюється на пристрій на основі STM32, та його програмне забезпечення.

Область застосування – побутова техніка, гідрометцентри.

2.ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3.МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Мета проекту полягає в розробці всіх необхідних схем, програмного забезпечення для мікроконтролера та в розробці самого мікроконтролера і спаренні його з мобільним пристроєм.

4. ДЖЕРЕЛА РОЗРОБКИ

4.1 Aghdasi, F., & Bhasin, A. (n.d.). DMA controller design using self-clocked methodology. 2004 IEEE Africon. 7th Africon Conference in Africa (IEEE Cat. No.04CH37590), PP.443-450.

4.2 Jayant Mankar. REVIEW OF I2C PROTOCOL / Jayant Mankar , Chaitali Darode , Komal Trivedi// E-ISSN: 2321−9637 International Journal of Research in Advent Technology. Volume 2, Issue 1, January 2014. − №3. − PP. 474-479.

4.3 Garima. Design & development of daughter board for Raspberry Pi to support Bluetooth communication using UART. / Garima, Agarwal, N., & Reddy// International Conference on Computing, Communication & Automation.- S. R. N. (2015). − 2015. − P.949-954.

4.4 A Review on Implementation of UART using Different Techniques / Ashwini D. Dhanadravye , Samrat S. Thorat // Ashwini D. Dhanadravye et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (1) , 2014, P. 394-396.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту, що розробляється

5.1.1 Для розробки IoT пристрою на основі STM32 потрібно уважно ознайомитися з технічними характеристиками данної плати.

5.1.2 Для розробки пристрою потрібні правильно підібрані модулі.

5.1.3 Для розробки пристрою потрібні правильно розроблені схеми: принципова, схема алгоритму програми та функціональна.

5.2. Вимоги до програмного забезпечення

Програмне забезпечення повинне бути надійним, щоб забезпечувати стабільні та точні виміри бажаних величин, а також щоб стабілізувати подачу запиту з мобільного пристрою, за допомогою bluetooth.

5.3. Вимоги до апаратної частини

- Процесор рівня Intel i5 і вище.
- Оперативна пам’ять не менше 500 МБ.
- Вільне місце на жорсткому диску не менше 100 МБ.
- Програматор для програмування плати.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	20.12.2019
Створення та узгодження технічного завдання	15.01.2020
Вивчення літературних джерел	27.01.2020
Розробка основних схем	15.04.2020
Розробка програмного алгоритму	01.05.2020
Створення програми та її налаштування на пристрої	15.05.2020
Оформлення документації дипломного проекту	06.06.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Система передавання даних для реалізації IOT на базі мережевих протоколів зв’язку»

Київ – 2021 року

Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Музика М.О.			*IoT пристрій на базі мікроконтролера STM32* Відомість дипломного проєкту	Літ.	Арк. 1 / Акрушів
Перевір.		Ткаченко В.В.					
						КПІ ім. Ігоря Сікорського ФІОТ ІО-72	
Н. Контр.		Сімоненко В.П.					
Затверд.		Стіренко С.Г.					

ЗМІСТ

Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Музика М.О.			*IoT пристрій на базі мікроконтролера STM32* Відомість дипломного проєкту	Літ.	Арк. 1 / Акрушів
Перевір.		Ткаченко В.В.					
						КПІ ім. Ігоря Сікорського ФІОТ ІО-72	
Н. Контр.		Сімоненко В.П.					
Затверд.		Стіренко С.Г.					

СПИСОК СКОРОЧЕНЬ

UART	Універсальний асинхронний приймач-передавач.
I2C	Інтегральна шина.
DMA	Прямий доступ до пам’яті.
CPU	Центральний процесор.
АЦП	Аналого-цифровий перетворювач.
ЦАП	Цифро-анаалоговий перетворювач.
ТЗ	Технічне завдання
ПЗ	Пояснювальна записка

ВСТУП

Настав той час, коли мікроконтролери використовуються всюди: від космічних ракет до кавоварок, від автомобілів до персональних комп'ютерів.

Пристрої на основі мікроконтролерів буквально стали незамінною частиною життя для багатьох. Кожен комп'ютер світу в своїй більшості складається з них. Але що ж таке мікроконтролер, якщо розглядати його, як окремий пристрій?

В цілому мікроконтролер сам по собі може виконувати лише елементарні завдання, які ж - залежить лише від призначення блоків, які є в ньому.

Коли вони вперше стали доступними, мікроконтролери використовували виключно асемблерну мову. Сьогодні мова програмування С є популярним варіантом. Інші поширені мови мікропроцесорів включають Python та JavaScript.

Мікроконтролери мають вхідні та вихідні штифти для реалізації периферійних функцій. Такі функції включають аналого-цифрові перетворювачі, контролери рідкокристалічного дисплея (РК), годинник реального часу (RTC), універсальний синхронний/асинхронний приймач-передавач (USART), таймери, універсальний асинхронний приймач-передавач (UART) та універсальну послідовну шину (USB). Датчики, що збирають дані, що стосуються, зокрема, вологості та температури, також часто прикріплені до мікроконтролерів.

Мікроконтролери використовуються в різних галузях промисловості та додатках, у тому числі в побуті та на підприємстві, автоматизації будівель, виробництві, робототехніці, автомобільній промисловості, освітленні, інтелектуальній енергетиці, промисловій автоматизації, розгортанні Інтернету речей(IoT).

Одним із дуже специфічних застосувань мікроконтролера є його використання як цифрового процесора сигналів. Часто вхідні аналогові сигнали надходять з певним рівнем шуму. У цьому контексті шум означає

неоднозначні значення, які неможливо легко перевести у стандартні цифрові значення. Мікроконтролер може використовувати свої АЦП і ЦАП для перетворення вхідного шумного аналогового сигналу в рівномірний вихідний цифровий сигнал.

Найпростіші мікроконтролери полегшують роботу електромеханічних систем, які можна знайти в побутових зручностях, таких як духовки, холодильники, тостери, мобільні пристрої, системи відеоігор, телевізори та системи поливу газону. Вони також поширені в офісних машинах, таких як копіювальні машини, сканери, факсимільні апарати та принтери, а також розумні лічильники, банкомати та системи безпеки.

Більш досконалі мікроконтролери виконують найважливіші функції в літаках, космічних апаратах, океанських суднах, транспортних засобах, системах медичного та життєзабезпечення, а також у роботах. У медичних сценаріях мікроконтролери можуть регулювати роботу штучного серця, нирок або інших органів. Вони також можуть сприяти функціонуванню протезів.

Різниця між мікроконтролерами та мікропроцесорами стала менш чіткою, оскільки щільність та складність мікросхем стали порівняно дешевими у виробництві, і, таким чином, мікроконтролери інтегрували більш "загальні комп'ютерні" типи функціональності. Загалом, можна сказати, що мікроконтролери корисно функціонують самі по собі, безпосередньо підключаючись до датчиків і виконавчих механізмів, де мікропроцесори призначені для максимального збільшення підключіть живлення до мікросхеми за допомогою внутрішньої шини (а не прямого вводу-виводу) для підтримки обладнання, такого як оперативна пам'ять та послідовний порт. Простіше кажучи, кавоварки використовують мікроконтролери; настільні комп’ютери використовують мікропроцесори.

Сучасні проекти великих дата-центрів, розумних будинків зазвичай побудовані на пристроях з мікроконтролерами та їх взаємодії. Саме через це, створення програмної частини коду для мікроконтролерів є, і завжди буде актуальною тематикою.

Мобільна революція дала нам щось більше, ніж телефонний досвід: крихітні, високоякісні, дешеві, низькоенергетичні, пов’язані між собою компоненти, а також нові процеси та промислові потужності, щоб зробити їх багато. Кожен смартфон - це колекція цікавого обладнання, більша частина якого розроблена спеціально для випадку використання мобільних пристроїв, але, витягнувши його з цього контексту, можна використовувати всілякі цікаві враження за межами мобільних телефонів.

Незабаром буде важко придбати новий продукт без якогось пов'язаного розуму. Управління приладом з мобільного телефону стане стандартною функцією, а всюдисущі датчики та управління перетворять промислову автоматизацію, будівлі та нашу взаємодію з навколишнім світом. Ця вся підключена апаратна трансформація обіцяє стати найбільшою частиною обчислювальної революції.

Його живлення значною мірою буде новим класом мікроконтролерів, що надає низку вбудованих переваг, серед яких: ціна, функціональність / продуктивність та вартість.

В той час як прикладні процесори на SBC з часом мають тенденцію до великого енергоспоживання, мікроконтролери фактично налаштовані на підвищення енергоефективності. Наприклад, мікросхема STM32F7 вдвічі потужніша, але використовує половину енергії свого попередника STM32F4.

Ця енергоефективність означає, що мікроконтролери, навіть із датчиками, можуть роками працювати на маленькій батареї або необмежено довго, додаючи невеликий сонячний елемент.

І хоча не кожен випадок використання IoT вимагає необхідності працювати від акумулятора, енергоефективність мікроконтролера приносить дивіденди іншими способами.

По-перше, завдяки використанню частки потужності, яку роблять одноплатні комп'ютери, їх загальна вартість володіння значно нижча; економія, яка часто помножується на кілька установок пристроїв. І все це заощадження енергії означає, що мікроконтролери працюють набагато

прохолодніше, ніж процесори додатків, що не тільки зменшує складність охолодження в конструкціях, але і корисно в місцях, де важке охолодження, наприклад, у космосі.

Це невелике споживання електроенергії також означає, що вони можуть живитися від альтернативних джерел, ніж від настінної вилки, таких як технологія збирання енергії. Це означає, що їх можна розміщувати практично де завгодно, значно розширюючи їх адреси використання та виконуючи обіцянки розміщення IoT практично скрізь.

Дуже мало випадків використання IoT, коли процес заявки або SBC насправді необхідний для їх підтримки, значною мірою тому, що мало завдань функціонально обмежує обчислювальна потужність мікроконтролера.

Наприклад, Machine Vision неймовірно добре працює на мікроконтролерах. У фантастичному дописі Піта Уордена «Чому майбутнє машинного навчання крихітне» він стверджує, що мікроконтролери насправді кращі в машинному навчанні, ніж зовнішні архітектури, знайдені в SBC. І він повинен знати; він очолює команду TensorFlow в Google, яка спеціалізується на мобільних та вбудованих програмах машинного навчання.

Інтерфейси користувача (UI) - це ще одна причина, яку іноді наводять для використання процесу застосування для IoT, але, по правді кажучи, більшість сучасних мікроконтролерів мають вбудоване графічне прискорення 2D. І існує цілий ряд бібліотек графіки, націлених на мікроконтролер.

Одне з небагатьох місць, де одноплатний комп'ютер насправді перемагає мікроконтролер, - це 3D-графіка та вихід HDMI. Але майже за визначенням існує мало випадків використання IoT, які вимагають таких функцій кіоску. При спробі провести межу навколо того, що таке IoT, кіоски не легко приходять на розум. А завдяки 2D-графічному прискоренню, підтримці вхідного сенсорного екрану та широкій підтримці дисплеїв, які самі по собі не є комп'ютерами (будь-який дисплей із входом HDMI, швидше за все, є самим комп'ютером), а замість цього, товарні компоненти, мікроконтролери справді сяють.

Хоча переваг мікроконтролерів багато, є одна видатна особливість, яка спонукає їх стати найважливішою частиною революції IoT: ціна. 32-розрядні мікроконтролери стартують приблизно за 1 долар кожен у виробничих кількостях, і навіть такий флагманський мікроконтролер, як STM32F7 (топовий ARM F7 від ST), коштує менше 10 доларів. І новий різновид мікроконтролера від Espressif Systems, ESP32 має вбудований Wi-Fi та Blutetooth Low-Energy (BLE) і коштує близько 2 доларів.

Ви можете придбати або побудувати від 5 до 10 пристроїв IoT на основі мікроконтролера за ту ж ціну, що і один пристрій SoC! А на нижньому кінці спектру мікроконтролера розрив у ціні стає ще більш очевидним.

Революція пов’язаних речей вже на шляху, і вона буде живитися мільярдами крихітних підключених мікроконтролерів. Вони охоплюють майже всі можливі випадки використання IoT, багато з яких не можуть бути практично підтримані будь-якою іншою архітектурою мікросхем, і вони роблять це з часткою від загальної вартості володіння. І хоча оснащення мікроконтролерів може в основному все ще бути примітивним, поле швидко змінюється у відповідь на зростаючий попит на рішення для мікроконтролерів.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1Мікроконтролери та їх опис

Мікроконтролер - це компактна інтегральна схема, призначена для управління конкретною операцією у вбудованій системі. Типовий мікроконтролер включає в себе процесор, пам'ять та периферію вводу - виводу (введення / виведення) на одному чіпі.[1]

Іноді їх називають вбудованим контролером або блоком мікроконтролера (MCU), мікроконтролери серед інших пристроїв зустрічаються в транспортних засобах, роботах, офісних машинах, медичних пристроях, мобільних радіостанціях, торгових автоматах та побутовій техніці. Вони є по суті простими мініатюрними персональними комп’ютерами (ПК), призначеними для управління дрібними функціями більшого компонента, без складної інтерфейсної операційної системи (ОС).

Як працюють мікроконтролери? Мікроконтролер вбудований всередину системи для управління особливою функцією в пристрої. Він робить це, інтерпретуючи дані, які отримує від периферійних пристроїв вводу-виводу, використовуючи центральний процесор. Тимчасова інформація, яку отримує мікроконтролер, зберігається в його пам'яті даних, де процесор отримує до неї доступ і використовує інструкції, що зберігаються в його програмі пам'яті, для розшифровки та застосування вхідних даних. Потім він використовує свої периферійні пристрої вводу-виводу для зв’язку та здійснення відповідних дій.

Мікроконтролери використовуються в широкому діапазоні систем і пристроїв. У пристроях часто використовується кілька мікроконтролерів, які працюють разом для вирішення відповідних завдань.[3]

Наприклад, в машині може бути багато мікроконтролерів, які керують різними окремими системами, такими як антиблокувальна система гальмування, контроль тяги, впорскування палива або контроль підвіски. Всі мікроконтролери взаємодіють між собою, щоб повідомити правильні дії. Деякі

можуть спілкуватися з більш складним центральним комп'ютером у машині, а інші можуть спілкуватися лише з іншими мікроконтролерами. Вони надсилають та отримують дані за допомогою периферійних пристроїв вводу-виводу та обробляють ці дані для виконання своїх призначених завдань.

Основними елементами мікроконтролера є:

- Процесор (CPU) - процесор можна сприймати як мозок пристрою. Він обробляє та реагує на різні інструкції, що керують функцією мікроконтролера. Це передбачає виконання основних арифметичних, логічних та операцій вводу-виводу. Він також виконує операції передачі даних, які передають команди іншим компонентам більшої вбудованої системи.
- Пам'ять - пам'ять мікроконтролера використовується для зберігання даних, які отримує процесор, і використовує їх для реагування на вказівки, які він запрограмував виконувати.

Мікроконтролер має два основних типи пам'яті:

- Пам'ять програм, яка зберігає довгострокову інформацію про інструкції, які виконує центральний процесор. Пам'ять програм - це енергонезалежна пам'ять, тобто вона зберігає інформацію з часом, не потребуючи джерела живлення.
- Пам'ять даних, яка потрібна для тимчасового зберігання даних під час виконання інструкцій. Пам'ять даних є нестабільною, тобто дані, які вона зберігає, є тимчасовими і підтримуються лише в тому випадку, якщо пристрій підключено до джерела живлення.[2]

Периферійні пристрої вводу-виводу - пристрої введення та виведення є інтерфейсом процесора до зовнішнього світу. Вхідні порти отримують інформацію та передають її процесору у вигляді двійкових даних. Процесор отримує ці дані і надсилає необхідні інструкції вихідним пристроям, які виконують завдання, зовнішні для мікроконтролера.

Хоча процесор, пам’ять та периферія вводу-виводу є визначальними елементами мікропроцесора, є й інші елементи, які часто включаються. Сам термін периферія вводу / виводу просто стосується допоміжних компонентів,

які взаємодіють з пам'яттю та процесором. Є багато допоміжних компонентів, які можна класифікувати як периферійні пристрої. Наявність певного прояву периферійного вводу-виводу є елементом мікропроцесора, оскільки вони є механізмом, за допомогою якого застосовується процесор.

Інші допоміжні елементи мікроконтролера включають:

- Аналого-цифровий перетворювач (АЦП) - це схема, яка перетворює аналогові сигнали в цифрові. Це дозволяє процесору в центрі мікроконтролера взаємодіяти із зовнішніми аналоговими пристроями, такими як датчики.
- Цифрово-аналоговий перетворювач (ЦАП) - виконує зворотну функцію АЦП і дозволяє процесору в центрі мікроконтролера передавати свої вихідні сигнали зовнішнім аналоговим компонентам.
- Системна шина - це з'єднувальний провід, який з'єднує всі компоненти мікроконтролера разом.
- Послідовний порт - послідовний порт є одним із прикладів порту вводу-виводу, що дозволяє мікроконтролеру підключатися до зовнішніх компонентів. Він має функцію, подібну до USB або паралельного порту, але відрізняється способом обміну бітами.

Особливості мікроконтролера. Процесор мікроконтролера залежить від програми. Варіанти варіюються від простих 4-розрядних, 8-розрядних або 16-розрядних процесорів до більш складних 32-розрядних або 64-розрядних процесорів. Мікроконтролери мають змогу використовувати енергозалежні типи пам’яті, такі як оперативна пам’ять (RAM) та енергонезалежні типи пам’яті - сюди входить флеш-пам’ять, стирається програмована пам’ять лише для читання (EPROM) та програмована лише для читання пам’ять (EEPROM).[5]

Як правило, мікроконтролери призначені для зручного використання без додаткових обчислювальних компонентів, оскільки вони розроблені з достатньою вбудованою пам'яттю, а також пропонують висновки для

загальних операцій вводу-виводу, завдяки чому вони можуть безпосередньо взаємодіяти з датчиками та іншими компонентами.

Архітектура мікроконтролера може базуватися на архітектурі Гарварда або архітектурі фон Неймана, обидва пропонуючи різні методи обміну даними між процесором і пам'яттю. В архітектурі Гарварда шина даних та інструкція є окремими, що дозволяє одночасно передавати дані. В архітектурі фон Неймана одна шина використовується як для даних, так і для інструкцій.[4]

Процесори мікроконтролера можуть базуватися на складних обчисленнях наборів команд (CISC) або скорочених обчисленнях наборів команд (RISC). CISC, як правило, має близько 80 інструкцій, тоді як RISC має близько 30, а також більше режимів адресації, 12-24 порівняно з 3-5 RISC. Хоча CISC може бути простішим у впровадженні та більш ефективним використанням пам'яті, він може погіршити продуктивність через більшу кількість тактових циклів, необхідних для виконання інструкцій. RISC, який робить більший акцент на програмному забезпеченні, часто забезпечує кращу продуктивність, ніж процесори CISC, які роблять більший акцент на апаратному забезпеченні завдяки спрощеному набору інструкцій і, отже, збільшеній простоті дизайну, але через акцент, який він робить на програмному забезпеченні, програмне забезпечення може бути більш складним. Який ISC використовується, залежить від програми.

1.2 Мета та цілі дипломної роботи

Метою моєї дипломної роботи є ознайомлення з принципами роботи мікроконтролера на базі STM32 та його взаємодією з периферійними пристроями та створення на основі нього пристрою, який виконуватиме функції годинника, термометра, барометра та датчика освітлення, і який також зможе здійснювати обмін даними зі спряженим мобільним телефоном за використання технології bluetooth.

Для досягнення цілі потрібно:

- ознайомитися з властивостями і специфікацією даного мікроконтролера;
- розробити організаційну схему пристрою та розрахувати характеристики усіх елементів;
- створити протокол передачі даних для налаштування комунікації зі спряженим мобільним телефоном;
- створити програмне забезпечення;
- безпосередньо розробити сам прилад.

Об'єктом мого дослідження є робота мікроконтролера разом з його взаємодією з периферією за допомогою вбудованими в ньому інтерфейсами зв'язку.

Предметом мого дослідження є мікроконтролер stm32f103c6t8, його взаємодія з датчиками температури, тиску, освітлення, дисплеєм, а також з вбудованим модулем бездротової комунікації в основі якого технологія bluetooth, з допомогою інтерфейсів UART, I2C та GPIO.

Під час виконання даної роботи я використовував такі *методи дослідження*:

- абстрагування: під час написання програмної частини роботи периферійні пристрої і зовнішні інтерфейси розглядалися як абстрактні об'єкти, взаємодія з якими відбувається за участі програмних функцій, які були створені раніше;
- моделювання: для визначення опору резисторів, які необхідні для належного функціонування шини I2C і дисплею була створена модель електричного кола, частиною якої будуть прилади, згадані вище;
- спостереження: в процесі виконання даної роботи були проведені потрібні спостереження за стабільністю та коректністю виконання програми з використанням дисплею, а також програматора, який надає персональному комп'ютеру повну інформацію про коректність станів регістрів даного приладу.

Практичне значення отриманих результатів. В роботі поступово описано методологію створення приладу на основі мікроконтролера stm32f103c6t8 з використанням периферійних датчиків і дисплею, а також розробку програмної частини забезпечення для його функціонування та спарення з мобільним пристроєм з використанням технології bluetooth.

Висновки до розділу 1

Мікроконтролери з часу їх створення стали невід'ємною частиною нашого життя, а їх сполучення, в вигляді сучасних девайсів стали нашими життєвими супутниками. Вони можуть виконувати різні задачі, вести обмін інформацією, збирати та висвітлювати дані тощо. І саме системи з використанням мікроконтролерів можна зустріти де завгодно.

В результаті виконання досліджень, що пов'язані з компонентами мого проекту, ми визначили основні теоритичні засади складових частин приладу, а саме, те що плата STM32F103 приєднується до датчиків тиску, температури, освітленості через шину даних I2C, пов'язаний з Bluetooth-модулем через інтерфейс UART, а також виводить зображення на циферблат.

Також в моєму проекті STM32 приєднана до чотирьох кнопок, які відповідають за перемикання функцій плати.

Технологія Bluetooth дозволяє нам отримати дані, які виміряла система, на підключений телефон, що додає більше комфорту в використання подібного функціоналу.

А завдяки DMA-контролеру, швидкість виконання операцій збільшується, тому що він може отримувати доступ до центральної шини, незалежно від процесора.

Розділ 2. СТВОРЕННЯ ПРИЦИПОВОЇ СХЕМИ

2.1 Опис використаних технологій

2.1.1 Характеристики відладочної плати

Сімейство 32-розрядних мікроконтролерів STM32 на основі процесора Arm ® Cortex ®-M розроблено, щоб запропонувати новим рівням свободи користувачам MCU. Він пропонує продукти, що поєднують дуже високу продуктивність, можливості в режимі реального часу, цифрову обробку сигналу, роботу з низьким енергоспоживанням / низькою напругою та підключення, зберігаючи при цьому повну інтеграцію та простоту розробки.

Неперевершений асортимент мікроконтролерів STM32, заснований на стандартному ядрі, постачається з широким вибором інструментів та програмного забезпечення для підтримки розробки проектів, що робить це сімейство продуктів ідеальними як для невеликих проектів, так і для наскрізних платформ.

Запроваджено низькопотужну, високоефективну систему збору даних про потужність на базі STM32 та описано принцип роботи системи та її програмне та апаратне забезпечення. STM32 містить безліч функціональних модулів. Без зовнішніх мікросхем розширення система може використовувати власний АЦП STM32 для виконання багатоканального синхронного аналого-цифрового перетворення вхідних сигналів, використовувати гнучкий статичний контролер пам'яті FSMC для розширення даних зберігання NAND FLASH та використовувати STM32 Розширений стандартний інтерфейс зв'язку реалізує Віддалений зв'язок RS485 на основі протоколу MODBUS, який долає недоліки традиційних збирачів даних, таких як обмежений простір для зберігання даних та інтерфейс зв'язку, низька точність та низька продуктивність у режимі реального часу. Фактична робота показує, що продуктивність в режимі реального часу та надійність даних про потужність,

зібраних цією системою, значно покращуються, і система має переваги низької вартості, невеликих розмірів та доброзичливої взаємодії людини з комп'ютером.

Є багато причин для використання послідовного дизайну інтерфейсу

Багато поширених периферійних пристроїв вбудованої системи, таких як аналого-цифрові та цифрово-аналогові конвектори, РК-дисплеї та датчики температури підтримують послідовний інтерфейс.

Послідовний інтерфейс дозволяє процесорам спілкуватися без потреби у спільній пам'яті та проблем, які вони можуть створити. Існують послідовні протоколи зв'язку, такі як UART, CAN, USB, SPI, Inter IC. [6]

USB використовує мультиплексор для зв'язку з іншими пристроями. Використовуються лише в протоколах I2C та CAN програмна адресація. Але лише I2C дуже простий у проектуванні та простий в обслуговуванні.

Таблиця 2.1 – Переваги та недоліки інтерфейсів обміну даними.[7]

	UART	CAN	SPI	I^2C
Плюси	Поширений, простий	Безпечний, швидкий	Швидкий, дешевий, універсальний	Швидкий, універсальний ,дешевий
Мінуси	Обмежена функціональність, зв’язок точка-точка	Комплексний, автомобільно орієнтований	Немає фіксованого стандарту	Обмежена кількість компонентів на 1 шині

2.1.2 Інтерфейс UART

UART (universal asynchronous receiver/transmitter) використовує дві лінії з одним напрямком – для прийому і передачі даних. Існує багато протоколів для передачі даних, самим поширеним з них є RS-232. Для комунікації двох пристроїв між собою за допомогою UART вони мають бути з однаковою

швидкістю прийому-передачі даних. UART зачасту використовується разом з іншими комунікаційними стандартами, такими як EIA RS-232.[8]

“UART” означає Універсальний асинхронний приймач-передавач. Це апаратне периферійне обладнання, яке знаходиться всередині мікроконтролера. Функція UART полягає у перетворенні вхідних та вихідних даних у послідовний двійковий потік. 8-бітові послідовні дані, отримані від периферійного пристрою, перетворюються в паралельну форму за допомогою послідовного в паралельне перетворення, а паралельні дані, отримані від центрального процесора, за допомогою послідовного в паралельне перетворення. Ці дані представлені в модулюючій формі і передаються із заданою швидкістю передачі.

Зазвичай це окрема мікросхема, або частина мікросхеми, що застосовується для з’єднання через комп’ютерний або периферійний послідовний порт. Сучасні мікросхеми випускаются з можливістю комунікації в синхронному режимі, подібні прилади носять назву USART. USART чіпи зазвичай мають два режими: синхронний та асинхронний.

Модуль послідовного зв'язку UART розділений на три підмодулі: генератор швидкості передачі, приймач модуль і модуль передавача.

Швидкість передачі даних - це насправді дільник частоти, який може розраховуватися відповідно до тактової частоти системи та бажаної швидкості передачі даних.

Функція генератора швидкості передачі даних, що виробляють сигнал локального тактового сигналу, який набагато вище, ніж швидкість передачі даних для контролю прийому та передачі UART.

Приймач виконує послідовно-паралельне перетворення на асинхронний кадр даних, отриманий від послідовного введення даних.

Модуль передавача перетворює байти в послідовні біти відповідно до основного формату кадру, отриманого від центрального процесора.

Для того, щоб синхронізувати асинхронні послідовні дані та забезпечити цілісність даних, додані біти запуску, паритету та зупинки до послідовних даних.

Оскільки системи проектування без повної перевірки відкриті до підвищеної можливості виходу з ладу та пропуску, існує можливість забезпечити помилку передачі даних. Наразі UART вдосконалюється завдяки внутрішній діагностиці, можливості введення вбудованого самотестування (BIST).[9]

Мікросхема UART із вбудованим вбудованим самотестуванням (BIST) архітектура з використанням технології FPGA мінімізувала вимоги до тестувальника та процедури випробувань, починаючи з тестування рівня логічної схеми до рівня поля і, таким чином, покращується надійність продукту.

Щоб знати роботу UART, потрібно зрозуміти основні функції послідовного зв'язку. Передавач і приймач використовують стартовий біт, стоп-біт і параметри синхронізації для синхронізації між собою. Вихідні дані мають паралельну форму. Наприклад, у нас є 4-розрядні дані, для перетворення їх у послідовну форму нам потрібен паралельний послідовний перетворювач. Як правило, D-триггери або засувки використовуються для проектування перетворювачів.

Передача даних через UART:

- В першу чергу передається початковий біт (0).
- Далі проводится передача інформаційних бітів (зазвичай 8).
- Опісля передається біт парності (опціонально).
- Завершує передачу даних передача стоп-сигналу (0) з довжиною 1, 1.5 або 2 біта.

Для швидкого зв'язку використовуються такі протоколи, як SPI (послідовний периферійний інтерфейс) та USB (Universal Serial Bus). Коли швидкісна передача даних не потрібна, використовується UART. Це дешевий

комунікаційний пристрій з одним передавачем / приймачем. Для передачі даних потрібен один провід, а для прийому - інший.

Він може взаємодіяти з ПК (персональним комп'ютером) за допомогою перетворювача RS232-TTL або перетворювача USB-TTL. Загальне між RS232 та UART полягає в тому, що їм обом не потрібен годинник для передачі та прийому даних. Кадр Uart складається з 1 стартового біта, 1 або 2 стопових біта та біта парності для послідовної передачі даних.

2.1.3 Bluetooth та його використання

Технологія Bluetooth - це технологія бездротового зв'язку короткого діапазону, яка замінює кабелі, що з'єднують електронні пристрої, що дозволяє людині вести телефонну розмову через гарнітуру, використовувати бездротову мишу та синхронізувати інформацію з мобільного телефону на ПК, використовуючи одне і те ж ядро системи.

Радіочастотний приймач Bluetooth (або фізичний рівень) працює в неліцензійному діапазоні ISM з центром 2,4 ГГц (той самий діапазон частот, що використовується мікрохвильовкою та Wi-Fi). Основна система використовує прийомопередавач із змінюваними частотами для боротьби з перешкодами та вицвітанням.[10]

Пристрої Bluetooth керуються за допомогою радіочастотної топології, відомої як "зіркова топологія". Група пристроїв, синхронізованих таким чином, утворює пікомережу, яка може містити одного ведучого та до семи активних ведених пристроїв, з додатковими веденими пристроями, які не беруть активної участі в мережі. (Даний пристрій може також бути частиною одного або декількох піконетів, або як ведучий, або як ведений.) У пікомережі фізичний радіоканал спільно використовується групою пристроїв, які синхронізовані із загальним тактовим сигналом та перескоком частоти з головним пристроєм, що забезпечує посилання на синхронізацію.

Скажімо, головним пристроєм є ваш мобільний телефон. Всі інші пристрої у вашій пікомережі відомі як підлеглі. Це може включати вашу гарнітуру, GPS-приймач, MP3-плеєр, стереосистему тощо.

Пристрої в пікомережі використовують певний шаблон стрибка частоти, який алгоритмічно визначається головним пристроєм. Основною схемою стрибків є псевдовипадкове упорядкування 79 частот в діапазоні ISM. Шаблон стрибка може бути адаптований для виключення частини частот, які використовуються пристроями, що заважають. Техніка адаптивного стрибка покращує співіснування технології Bluetooth із статичними (непрохідними) системами ISM, такими як мережі Wi-Fi, коли вони розташовані поблизу пікомережі.

Фізичний канал (або бездротовий зв'язок) поділяється на одиниці часу, відомі як слоти. Дані передаються між пристроями з підтримкою Bluetooth у пакетах, які розташовані в цих слотах. Перехід частоти відбувається між передачею або прийомом пакетів, тому пакети, що складають одну передачу, можуть надсилатися на різних частотах в діапазоні ISM.

Фізичний канал також використовується як транспорт для одного або декількох логічних посилань, які підтримують синхронний і асинхронний трафік, а також широкомовний трафік. Кожен тип посилань має певне використання. Наприклад, синхронний трафік використовується для передачі аудіоданих “вільні руки”, тоді як асинхронний трафік може нести інші форми даних, які витримують більшу мінливість у термінах доставки, наприклад, друк файлу або синхронізація календаря між телефоном та комп’ютером.

Однією із складностей, часто пов’язаних з бездротовими технологіями, є процес підключення бездротових пристроїв. Користувачі звикли до процесу підключення дротових пристроїв, підключивши один кінець кабелю до одного пристрою, а інший кінець - до додаткового пристрою.

Надійний та захищений зв’язок між двома або більше пристроями вимагає дротового з’єднання. Бездротовий зв'язок, такий як - Bluetooth, WI-Fi,

ZigBee тощо забезпечує гнучке та недороге рішення для віддалених програм. Доступна велика кількість недорогих апаратних платформ, таких як Raspberry Pi, Arduino, плати mbed тощо, які не забезпечують вбудований бездротовий модуль, але оснащені портами UART, I2C для проектування та розвитку Інтернету речей (IoT) та вбудованих додатків. Розробники стикаються з труднощами використання такого недорогого обладнання для бездротового зв'язку для реалізації програм.

Технологія Bluetooth використовує принципи "запиту" та "сканування запиту". Скануючі пристрої слухають на відомих частотах пристрої, які активно запитують. Коли надходить запит, скануючий пристрій надсилає відповідь з інформацією, необхідною для запитуючого пристрою для визначення та відображення природи пристрою, який розпізнав свій сигнал.

Скажімо, ви хочете бездротовим способом надрукувати зображення зі свого мобільного телефону на сусідній принтер. У цьому випадку ви переходите до зображення на телефоні та вибираєте друк як опцію надсилання цього зображення. Телефон розпочне пошук пристроїв у цьому районі. Принтер (скануючий пристрій) відповість на запит і, як результат, з'явиться на телефоні як доступний друкарський пристрій. Відповідаючи, принтер готовий прийняти підключення. Коли ви вибираєте бездротовий принтер Bluetooth, процес друку починається шляхом встановлення з'єднань на послідовно вищих рівнях стека протоколу Bluetooth, які в цьому випадку керують функцією друку.

Як і будь-яка успішна технологія, вся ця складність триває без того, щоб користувач усвідомлював щось більше, ніж завдання, яке він або вона намагається виконати, наприклад, підключення пристроїв та розмову в режимі гучного зв'язку чи прослуховування високоякісної стерео музики на бездротових навушниках.

2.1.4 Опис шини даних I2C

I2C (Inter-Integrated Circuit) - послідовна шина даних, яка використовується для зв’язку інтегральних схем. Дана шина в принципі своєї роботи використовує дві лінії, кожна з яких має два напрями - лінію тактування (SCL) і лінію даних (SDA).[11]

Стандартна швидкість передачі даних - 100 кбіт/с, але новітні стандарти дають можливість розвивати швидкість передачі до 3.4 Мбіт/с.

Принцип роботи:

- процедура обміну починається з того, що вузол, який ініціює передачу даних генерує сигнал "старт" (перехід сигналу лінії SDA з високого стану в низький при високому рівні на SCL-лінії), також цей перехід сприймається всіма пристроями, які підключені до шини, як сигнал, що характеризує початок процедури обміну;
- генерація синхросигналу завжди є обов’язком ведучого, тобто кожен ведучий генерує свій власний сигнал синхронізації, при пересиланні даних по шині;
- передача даних завершується генерацією керуючим пристроєм сигналу "стоп" (переходу сигналу лінії SDA з низького стану в високий при високому рівні на SCL-лінії);
- стани “стоп” і “старт” завжди виробляються ведучим, також, вважають, що шина зайнята після виявлення стану “старт”, а стає звільненою через деякий час після виявлення стану “стоп”;
- переданим бітом вважають сигнал лінії SDA відразу після переходу стану лінії SCL з низького до високого. Зміна сигналу лінії SDA повинна відбуватися при низькому рівні сигналу SCL;
- після кожного байту підлеглим пристроєм генерується додатковий біт підтвердження (0);
- також пристрої можуть затримувати відповідь та при цьому тримати сигнал лінії SCL на низькому рівні.

Має 7-бітовий адресний простір (з розширенням до 10 в нових версіях). Загальна кількість пристроїв, які підключені до шини, обмежена адресним простором та максимальною ємністю лінії в 400 пФ. Кожен окремий пристрій на шині має свою унікальну адресу. Врахувавши 16 зарезервованих адрес, всього є 112 унікальних адрес для даних пристроїв. Адреса кожного пристрою може складатися із двох частин: статичної і динамічної, ця особливість дає нам змогу підключення до однієї шини декількох пристроїв одного типу. Динамічна частина має змогу визначатися з допомогою сигналів, які під'єднані до деяких виводів пристрою, з'єднання певних виводів одного пристрою між собою, або ж поданням сигналу на вивід з аналоговим способом задавання адреси.

Саме першим байтом виконується передача адреси пристрою, до якого потрібно звернутися (7 біт), та біт читання/запису (0/1). Після цього пристрій, адресу якого надав перший байт, вступає в комунікацію, а інші чекають сигналу "стоп". Наступним кроком, зазвичай, передають адресу реєстру (розмір залежить від пристрою) та дані, запис яких потрібний.

Перевагами шини I2C є наявність лише двох ліній та можливість підключення декількох пристроїв до однієї шини. Так як такі мікросхеми підключаються до шини без будь-яких додаткових ланцюгів, з'являється можливість модифікації та модернізації системи, за допомогою підключення, або відключення периферійних пристроїв від шини.

Однією із переваг, також є відсутність потреби в розробці шинних інтерфейсів, так як шина вже інтегрована в мікросхеми, а блоки на функціональній схемі відповідають мікросхемам та перехід схеми від функціональної до принципової відбувається за короткий проміжок часу.

Інтегрованість адресації пристроїв та протоколу передачі даних, дозволяють системі бути повністю програмно обумовленою. А мікросхеми можуть бути додані або видалені з системи, не впливаючи на інші мікросхеми, що підключені до шини.

Одним із найбільших плюсів даної шини є проста діагностика помилок і їх налагодження, порушення в роботі шини можуть бути відслідковані зразу після їх здійснення. Звідси випливає, що час, витрачений на розробку програмного забезпечення може бути знижений, за рахунок використання бібліотеки повторно використовуваних програмних модулів.

У будь-який час головний пристрій може розпочати транзакцію, витягнувши SDA низьким, а SCL високим (унікальна особлива умова, яку інші пристрої I2C розпізнають як початок головної передачі).

Ведений пристрій прослуховує наступні 7 послідовних бітів адреси, щоб перевірити, чи відповідає вона його власній адресі (кожен I2C повинен мати унікальну адресу). Якщо він розпізнає наступні 7 бітів як адресу від головного пристрою, тоді він буде споживати наступний біт R / W.

Це біт читання / запису, який говорить веденому приймати дані або генерувати дані в наступних транзакціях.

У той час, коли повинен бути сформований сигнал ACK, головний пристрій звільняє лінію SDA і відкритий зливний вихід піднімається високо. Це дозволяє веденому пристрою генерувати сигнал ACK, витягуючи його низьким рівнем (лише на той конкретний бітовий період). Головний пристрій контролює шину I2C за цим сигналом від веденого пристрою.

Інтерфейс I2C використовує дві двонаправлені лінії, що означає, що будь-який пристрій може керувати будь-якою лінією. В одній головній системі головний пристрій керує годинником більшу частину часу - головний відповідає за годинник, але раби можуть впливати на нього, щоб уповільнити його .

Два дроти повинні вестись як відкриті колекторні / зливні виходи і повинні бути витягнуті високо, використовуючи по одному резистору (це один резистор на лінію I2C, тобто для даних і годинника) - це реалізує “функцію дротового NOR” - будь-який пристрій, що тягне провід, низький, призводить до того, що всі пристрої бачать низьке логічне значення - для високого логічного значення всі пристрої повинні припинити введення дроту.

Якщо ви використовуєте I2C, ви не можете вставити на шину будь-які інші (не I2C) пристрої, оскільки обидві лінії в якийсь момент використовуються як годинник (генерація бітів START і STOP перемикає лінію даних). Отже, ви не можете зробити щось розумне, наприклад, утримувати лінійку годинника неактивною, і використовувати лінію даних як детектор натискання кнопки (для збереження контактів).

Є два дроти (три, якщо ви включаєте заземлення, і чотири, якщо ви також включаєте живлення) - але потужність і заземлення приймаються як вказано, тобто вони доступні на друкованій платі за необхідності, тому насправді не рахувати.

Компоненти I2C:

- Верхня вершина I2C містить регістр попереднього масштабу, регістр команд, регістр стану, регістр передачі та регістр прийому.
- Регістр попереднього масштабу використовується для зменшення високочастотного електричного сигналу до нижчої частоти шляхом цілочисельного ділення. Дані спочатку надходять у статус реєстру, і в залежності від нього регістр команд видає команди. Реєстр передачі та отримання вирішує, чи потрібно передавати або приймати дані, і ці дані передаються паралельно реєстру вводу-виводу даних
- -I²C Master Byte Controller - це байтовий контролер команд і регістр зсуву вводу-виводу даних. Байтовим контролером команд є серце трафіку зв'язку I²C на рівні байтів і є автоматом стану, який генерує різні стани байтових операцій I²C на основі бітів регістру команд. Регістр зсуву вводу-виводу даних - це компонент, який містить та обробляє пов'язані з ними дані з поточним I²C транзакцій запису та читання.
- - Контролер бітів I²C Master включає генератор тактової частоти та контролер бітових команд. Під час передачі дані зсуваються поступово в контролер командного біта, а звідти він передається на SDA. Під час прийому дані надходять на SDA, а потім в біт-контролер.[12]

2.1.5 Опис DMA-контролера

DMA-контролер (direct memory access) може отримувати доступ до системної шини незалежно від центрального процесора. Він містить регістри, доступні центральному процесору, за допомогою яких може бути сконфігурований. Може працювати з внутрішньою пам’яттю та регістрами периферії. Завдяки DMA-контролеру збільшується швидкість передачі та звільняється процесорний час, оскільки центральному процесору не потрібно відволікатись кожного разу при отриманні даних, а просто обробити фінальний масив отриманих даних.

Прямий доступ до пам'яті (DMA) - це спосіб передачі даних між пам'яттю та пристроями вводу-виводу. Це відбувається без участі процесора. У нас є два інших способи передачі даних, запрограмований ввід / вивід та введення / виведення з керуванням перериваннями.

При програмованому введенні / виведенні процесор продовжує сканувати, чи готовий будь-який пристрій для передачі даних. Якщо пристрій вводу-виводу готовий, процесор повністю віддає себе функції передачі даних між введенням-виведенням і пам'яттю. Він передає дані з високою швидкістю, але не може брати участь у будь-якій іншій діяльності під час передачі даних. Це основний недолік запрограмованого вводу-виводу.

У режимі вводу-виводу з керуванням перериваннями, коли пристрій готовий до передачі даних, тоді він піднімає переривання для процесора. Процесор завершує виконання своїх поточних інструкцій і зберігає поточний стан. Потім він переходить на передачу даних, що спричиняє затримку. Тут процесор не продовжує сканувати периферійні пристрої, готові до передачі даних. Але він повністю бере участь у процесі передачі даних. Отже, це також не ефективний спосіб передачі даних.

Зазначені два режими передачі даних не є корисними для передачі великого блоку даних. Але контролер DMA виконує це завдання швидше, а також ефективний для передачі великих блоків даних.

DMA-контролер має 8 каналів, які підтримують апаратні та програмні тригери, що зв’язують операції та передачу ланцюгів каналів, і забезпечує тривимірну передачу за допомогою наборів параметрів, щоб виконувати переміщення блоків даних, сортування даних та вилучення підкадрів різних структур даних. Механізм арбітражу каналів приймає апаратний пріоритет у поєднанні з алгоритмом обертання зваженого пріоритету. Більше того, DMA підтримує збільшення та обгортання режимів адресації та завершує передачу даних, при якій ширина даних для читання та запису відрізняється асиметричним асинхронним FIFO. Також, DMA приймає дизайн доменного режиму з двома тактами, щоб зменшити споживання енергії. Контролер DMA має функцію моста APB і забезпечує паралельну роботу шини AHB та шини APB. Даний контролер може прийняти буферний та не буферний режим передачі даних відповідно до швидкості обладнання. Завдяки 0,18 мкм бібліотечній технології SMIC досягається робоча частота 408 МГц. Експериментальні результати показують, що DMAC має кращі показники, ніж традиційні DMAC та DMA PL081.

Асинхронні послідовні схеми пропонують покращену швидкість роботи в порівнянні з їх синхронними аналогами. Було запропоновано низку нових методологій проектування, які передбачають локальне генерування годинника та використання його для самостійної синхронізації машини. Такі тактові сигнали генеруються, коли вхід змінюється, або шляхом керованого збудження, коли зміна входів вимагає зміни стану. Усі подібні конструкції, де ланцюг синхронізований локально сформованими тактовими частотами, називаються самоконтрольованими послідовними схемами. Цей документ використовує методологію проектування перемикання змінної стану через годинники, керовані даними, для реалізації контролера прямого доступу до пам'яті (DMAC) як приклад проектування. Дизайн моделюється програмно, а також реалізується з використанням дискретних апаратних компонентів. Методологію можна поширити на паралельні контролери для нейронних

мереж та автоматизувати за допомогою методів призначення стану, вже розроблених для синхронних паралельних контролерів

Контролер DMA передає дані у трьох режимах:

- Режим серійної передачі: тут, як тільки контролер DMA отримує заряд системної шини, він звільняє системну шину лише після завершення передачі даних. До цього часу процесор повинен чекати системних шин.
- Режим викрадення циклу: У цьому режимі контролер DMA змушує центральний процесор зупинити свою роботу і на короткий термін передати керування шиною контролеру DMA. Після передачі кожного байта контролер DMA звільняє шину, а потім знову запитує системну шину. Таким чином, контролер DMA викрадає тактовий цикл для передачі кожного байта.
- Прозорий режим: тут контролер DMA приймає заряд системної шини, лише якщо процесор не потребує системної шини.

Контролер DMA - це апаратний блок, який дозволяє пристроям вводу-виводу отримувати безпосередній доступ до пам'яті без участі процесора. Тут ми обговоримо роботу контролера DMA.

Кожного разу, коли пристрій вводу-виводу хоче передати дані в пам’ять або з неї, він надсилає запит DMA (DRQ) контролеру DMA. Контролер DMA приймає цей DRQ і просить центральний процесор затримати кілька тактових циклів, надіславши йому запит на утримання (HLD).[13]

ЦП отримує запит на утримання (HLD) від контролера DMA, відмовляється від шини і надсилає підтвердження про утримання (HLDA) контролеру DMA.

Отримавши підтвердження утримання (HLDA), контролер DMA підтверджує пристрій вводу-виводу (DACK), що передача даних може бути виконана, і контролер DMA приймає заряд системної шини і передає дані в пам'ять або з неї.

Коли передача даних завершена, DMA викликає переривання, щоб повідомити процесору, що завдання передачі даних закінчено, і процесор

може знову взяти під контроль шину і розпочати обробку там, де вона залишилася.

Тепер контролер DMA може бути окремим блоком, яким спільно користуються різні пристрої вводу-виводу, або він також може бути частиною інтерфейсу пристрою вводу-виводу.

Кожного разу, коли процесору пропонується прочитати або записати блок даних, тобто передати блок даних, він вказує контролеру DMA, надсилаючи наступну інформацію.

Перша інформація полягає в тому, чи потрібно дані зчитувати з пам'яті, чи дані записувати в пам’ять. Він передає цю інформацію через лінії керування зчитуванням або записом, які знаходяться між процесором та логічним блоком управління контролерів DMA.

Процесор також надає початкову адресу для блоку даних у пам'яті, звідки блок даних у пам'яті повинен читатися або де блок даних повинен бути записаний в пам'ять. Контролер DMA зберігає це у своєму адресному реєстрі. Його також називають регістром початкової адреси.

Процесор також надсилає кількість слів, тобто скільки слів потрібно прочитати або записати. Він зберігає цю інформацію в підрахунку даних або реєстрі слів.

Найважливішою є адреса пристрою вводу-виводу, який хоче читати або записувати дані. Ця інформація зберігається в реєстрі даних.

Прямий доступ до пам'яті (DMA) використовується для забезпечення швидкісної передачі даних між периферійними пристроями та пам'яттю, а також пам’яті в пам’ять. Дані можна швидко переміщати за допомогою DMA без будь-яких дій процесора. Це робить ресурси центрального процесора вільними для інших операцій.

Два контролери DMA мають 12 каналів (7 для DMA1 і 5 для DMA2), кожен призначений для управління запитами на доступ до пам'яті з однієї або декількох периферійних пристроїв. Він має арбітра для обробки пріоритету між запитами DMA.

Як тільки канал увімкнено, він може обслуговувати будь-який запит DMA з периферійного пристрою, підключеного до каналу. Після передачі половини байтів встановлюється прапор напівпередачі (HTIF) і генерується переривання, якщо встановлено біт увімкнення переривання половини передачі (HTIE). В кінці передачі встановлюється прапор передачі завершення (TCIF) і генерується переривання, якщо встановлений біт увімкнення переривання завершення передачі (TCIE).[14]

Висновки до розділу 2

В даному розділі було розглянуто принципи роботи модулів, які будуть використовуватися в моїй роботі, їх переваги, та причини їх вибору. З опрацьованої інформації, було розроблено список компонентів для даного приладу, та чіткий план дій по його розробці.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Основні розрахунки та створення принципової схеми

Для успішної розробки принципової схеми, потрібно відразу скласти список матеріалів, які будуть використані в її побудові.

Мій прилад складається з наступних компонентів:

– відладочна плата stm32f103c6t8-modul;
– динамічний дисплей 5641-AG зі спільним катодом;
– датчик температури та тиску BMP180;
– датчик освітлення APDS-9930;
– bluetooth-модуль JDY-10;
– 8 резисторів з опором 220 Ом;
– 4 резистора з опором 10 кОм;
– 2 резистора з опором 3,7 кОм;
– 4 біполярних n-p-n транзистора КТ315Н;
– 4 кнопки.

Після складення списку необхідно врахувати точні фізичні характеристики обраних матеріалів для пристрою.

3.1.1 Технічні характеристики відладочної плати

Відладочна плата stm32f103c6t8-modul має такі характеристики:

- центральний процесор ARM 32-bit Cortex™-M3 CPU з частотою 72 MHz;
- обробка переривань з хвостовим ланцюгуванням;
- 64 Kb флеш-пам’яті;
- 20 Kb SRAM-пам’яті;
- 8 MHz тактовий генератор;

- виділений 32 kHz тактовий генератор для RTC (годинника реального часу) з калібровкою;
- робоча напруга від 2 до 3.6 вольт;
- робоча температура від -40 до +85 °C;
- 3 таймери загального призначення.

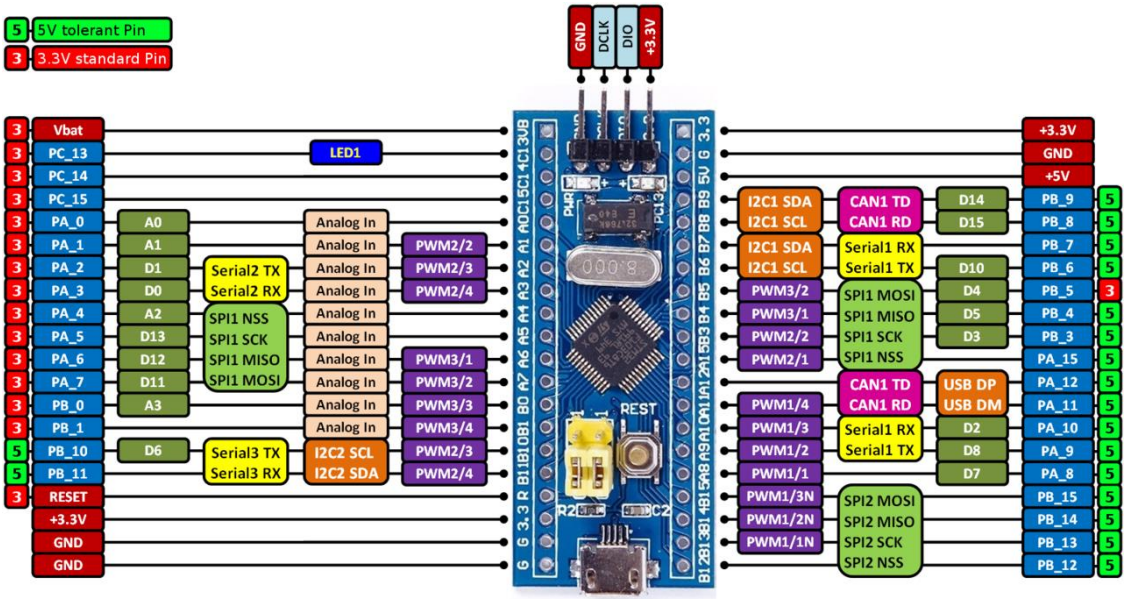


Рисунок 3.1 - Зовнішній вигляд та призначення виходів STM32F103C6T8

3.1.2 Технічні характеристики дисплею

Для відображення отриманої інформації в цій системі слугує дисплей 5641-AG.

В даній роботі я використав дисплей 5641-AG, так як він вважається легким в підключенні та простим в використанні.

Дисплей 5641-AG має такі характеристики:

- колір жовто-зелений;
- довжина хвилі світла – 570 нм;
- розміри цифри – 8.1 мм в довжину, 14.2 мм в висоту;
- максимальна сила струму – 25 мА;
- необхідна напруга – 2.1 В;

- максимальна напруга – 2.8 В.

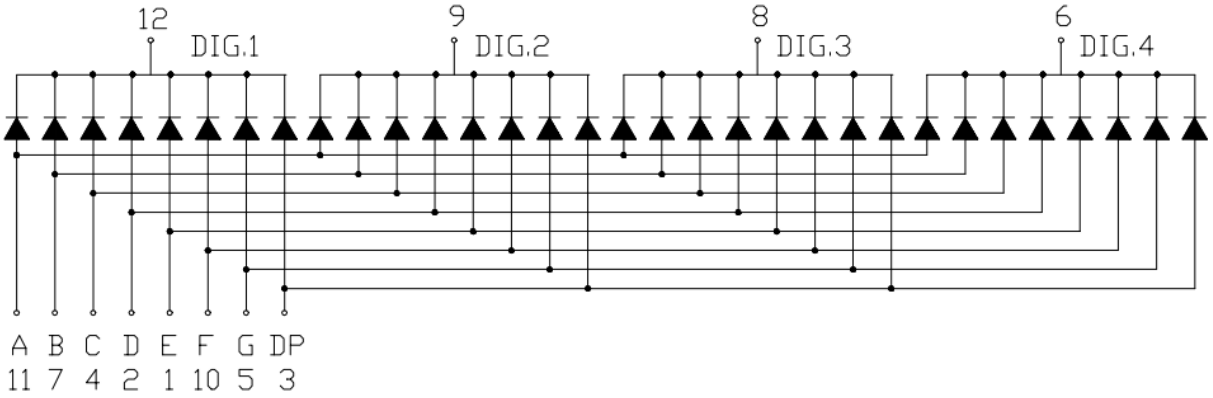


Рисунок 3.1 - Принципова схема дисплею

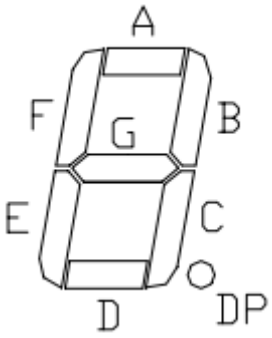


Рисунок 2.3 - Кодування сегментів

3.1.3 Датчик тиску, та його характеристики

Для вимірювання тиску я обрав датчик BMP180, тому що він є самим поширеним із доступних, та має I2C інтерфейс, що дає можливість підключення до шини.

Датчик атмосферного тиску BMP180 має такі характеристики:

- межі виміру тиску 30…110 кПа;
- похибка виміру тиску 5 Па;
- час конвертації АЦП 4.5 мс;
- можливість генерувати переривання при виявленні заданого нижнього або верхнього порогу освітлення;
- комунікація за допомогою I2C інтерфейсу;
- робоча напруга від 1.8 до 3.6 В;

- робоча температура від -40 до +85 °C.

3.1.4 Характеристики датчика освітлення

В якості датчика освітлення, я взяв модуль APDS-9930, так як датчик є поширеним та повністю підходить для створення мого проекту.

Датчик освітлення APDS-9930 має такі характеристики:

– похибка виміру 0.01 лк;
– час конвертації АЦП 2.73 мс;
– комунікація за допомогою I2C інтерфейсу;
– робоча напруга від 2.2 до 3.6 В;
– робоча температура від -40 до +85 °C.

3.1.5 Bluetooth-модуль

В якості Bluetooth модуля я використав JDY-10, тому що він підтримує використання інтерфейсу UART, та може підключитися до I2C шини.

Bluetooth-модуль JDY-10 має такі характеристики:

- максимальна потужність сигналу +8 дБ;
- чутливість прийому -92 дБ;
- швидкість обміну даними 115200 бод/с;
- дальність передачі 80 м;
- використовує технологію BLE;
- комунікація за допомогою UART інтерфейсу;
- робоча напруга від 1.9 до 3.6 В;
- робоча температура від -40 до +85 °C.

3.1.6 Визначення опору обмежувальних резисторів для дисплею

Наступним завданням стало визначення опору обмежувальних резисторів для пристрою, дана задача була поставлена для розподілу енергетичних ресурсів на платі.

Під час проведення розрахунків, з технічних причин, були задані такі обмеження:

- максимальний струм, що може проходити через вихід мікроконтролера: 25 мА;
- максимальний струм, що може проходити через один сегмент індикатора: 25 мА.

Наступним кроком було прийнято рішення для сегментів дисплею встановити обмежувальні резистори для забезпечення струму в 15 мА. Стандартна робоча напруга становить 3.3 В. З закону Ома визначаємо опір обмежувальних резисторів

$$R = \frac{U}{I} = \frac{3.3}{15 * 10^{-3}} = 220 \text{ (Ом)}$$

Опір обмежувальних резисторів для бази транзисторів обчислювався наступним чином. Мінімальний коефіцієнт посилення (h_{21}) транзисторів КТ315Н становить 50. Спад напруги на транзисторі (U_t) становить 0.7 В. Звідси випливає:

$$h_{21} = \frac{I_c}{I_b}; I_b = \frac{I_c}{h_{21}}; I_b = \frac{15 * 10^{-3}}{50} = 3 * 10^{-4}\ (A)$$

$$U_R = U - U_t; U_R = 3.3 - 0.7 = 2.6 \text{ (В)}$$

$$R = \frac{U_R}{I_b}; R = \frac{2.6}{3 * 10^{-4}} \approx 8667 \text{ (Ом)}$$

Отже, обираємо резистори з опором 10 кОм.

3.1.7 Визначення опору обмежувальних резисторів для ліній комунікації I2C

Для дротів комунікації I2C теж потрібні обмежувальні резистори, з технічних характеристик елементів схеми визначаємо головні обмеження:

- максимальний струм, що проходить по лінії, не повинен перевищувати 3 мА;
- час зміни сигналу ліній SDA та SCL не повинен перевищувати 1 мкс;

Використовуючи закон Ома:

$$R = \frac{U}{I}$$

Отримуємо, що мінімальний опір резистора має становити:

$$R_{min} = \frac{U_{CC} - U_{OL}}{I},$$

де U_{CC} – напруга, що подається для підтяжки ліній, а U_{OL} – спад напруги на транзисторі пристрою при струмі в 3 мА.

$$R_{min} = \frac{3.3 - 0.4}{3 * 10^{-3}} \approx 966.67 \text{ (Ом)}$$

Максимальний опір резистора визначається за формулою

$$R_{max} = \frac{t}{0.8473 * C},$$

де t – час зміни сигналу, C – ємність лінії.

Ємність виходів кожного з двох датчиків становить 10 пФ, ємність виходів мікроконтролера – 5 пФ. Отже:

$$R_{max} = \frac{10^{-6}}{0.8473 * 25 * 10^{-12}} \approx 47208 \text{ (Ом)}$$

Для ліній комуніікації I2C було обрано транзистори з опором 3.7 кОм.

3.1.8 Побудова принципової та функціональної схем пристрою

Використовуючи дані, отримані під час дослідження stm32f103c6t8-modul, периферійних модулів, та обчислення бажаного опору резисторів на платі, ми можемо розробити принципову схему пристрою, провести налаштування тактових частот на платі і зобразити це в електричній функціональній схемі.

Для побудови схем була використана програма DipTrace, побудова приладу проводилася з прямим використанням даних схем.

3.2 Опис засобів розробки програмного забезпечення

3.2.1 Вибір середовища розробки ПЗ

Після створення основних схем приладу, та його побудови в матеріальному варіанті, можна переходити до розробки програмного забезпечення приладу.

В якості середовища розробки було обрано SW4STM32 (System Workbench for STM32), яке являє собою середовище розробки Eclipce. модифіковане для роботи з мікроконтролерами на базі stm32. Також було використано STM32CubeMX, програму для генерації ініціалізуючого коду на мові С та налаштування виходів мікроконтролера за допомогою графічного інтерфейсу.

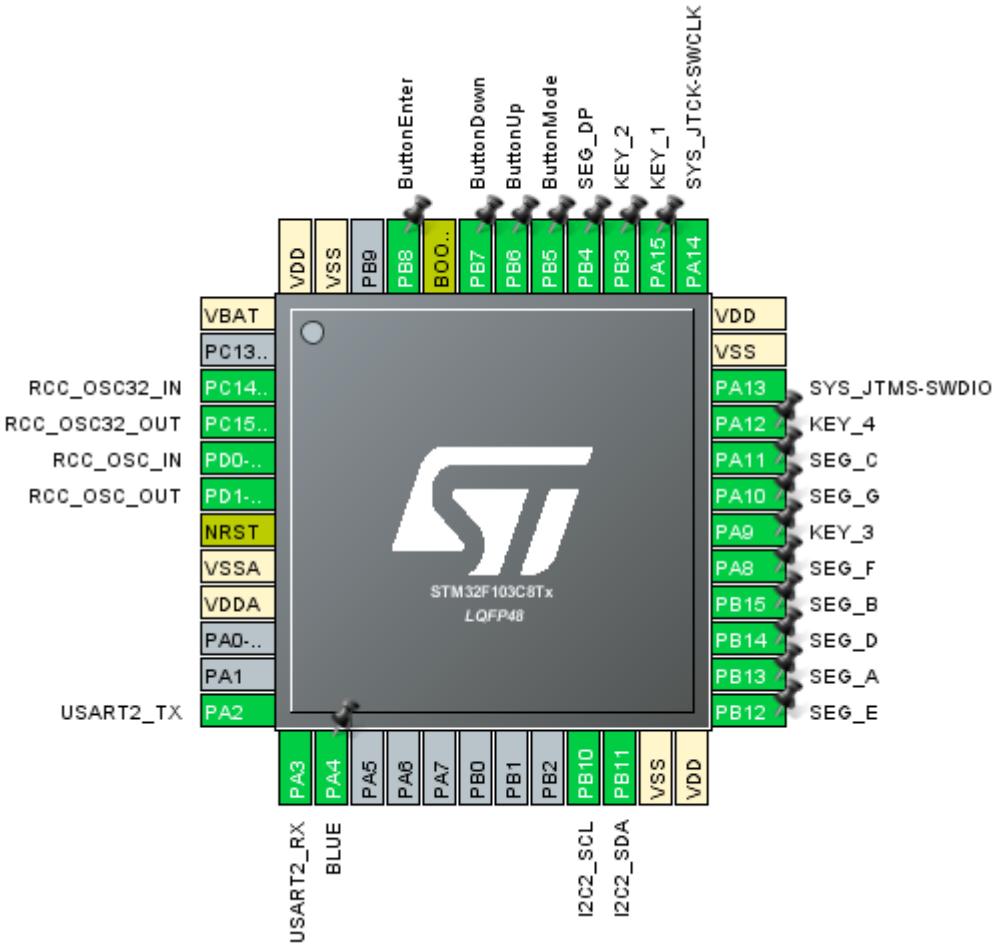


Рисунок 3.3 - Графічний інтерфейс для налаштування виходів мікроконтролера

Серед переваг STM32CubeMX:

- Інтуїтивно зрозумілий вибір мікроконтролера Cortex-M STM32 ARM для швидкого визначення деталей, які відповідають вашим вимогам
- Графічна конфігурація розв'язки з автоматичним вирішенням конфліктів
- Конфігурація графічного годинника з автоматичним підтвердженням
- Графічна конфігурація драйвера STM32Cube та програмного забезпечення проміжного програмного забезпечення з автоматичним підтвердженням обмежень
- Оцінка енергоспоживання
- Автоматичне створення середовища готових проектів IDE, включаючи код ініціалізації, для інструментів IAR, GCC та Keil
- Надається як окремий додаток та як плагін Eclipse

3.2.2 Опис функцій програми

Програма складається з наступних функцій:

– void clockInit() – ініціалізує годинник реального часу;

– void mainLoop() – містить в собі цикл, в якому виконуються незалежні від часу операції;

– void SetCharPins(char ch) – конфігурує виходи мікросхеми таким чином, щоб на дисплеї з'явився переданий символ;

– void timeToCharArray(char* symbols, uint8_t hours, uint8_t minutes) – перетворює два числа (години та хвилини) в масив з чотирьох символів;

– void setModeConfig() – змінює конфігурацію дисплею відповідно до вибраного режиму;

– void checkButtonsPressed() – перевірює, чи були натиснуті якісь кнопки, і, якщо так, викликає функції для обробки натиснень;

– void onButtonModePressed(), void onButtonModePressed(), void onButtonModePressed(), void onButtonModePressed() – функції для обробки натиснень;

– void switchDisplay() – змінює поточний символ, що відображається на дисплеї;

– void buttonPressed(uint16_t GPIO_Pin) – реєструє натиснення кнопки;

– void UART_init() – ініціалізує UART та налаштовує DMA для прийому даних;

– void bluetooth_init() – ініціалізує Bluetooth для початку комунікації;

– void UART_process_rx_data() – обробляє дані, що надійшли через UART;

– void UART2_idle_handler() – викликається, коли відбувається idle переривання;

– void UART2_tc_handler() – викликається після отримання переривання UART_IT_TC;

– void UART_send_data(uint8_t* source, uint8_t len) – пересилає дані з source довжиною len через UART за допомогою DMA;

– void bmp180_handler_init() – ініціалізує використання драйвера bmp180;

– void APDS9930_handler_init() – ініціалізує використання драйвера apds9930;

– uint32_t PaToMM(uint32_t p) – перетворює тиск в паскалях на тиск в міліметрах ртутного стовпчика.

3.2.3 Визначення списку переривань

Програмне забезпечення дозволяє обробляти певний список переривань, які відбуваються, це дозволяє збільшити стабільність роботи та уникнути певних збоїв в роботі. Також обробка переривань дуже важлива для аналізу та покращення програми під час тестування.

Програма містить обробку наступних переривання:

- переривання таймера, відбувається кожну 1 мс, використовується для зміни символу, що відображається. Викликає функцію switchDisplay();
- 4 переривання кнопок, виникають при натисканні на відповідну кнопку. Викликає функцію buttonPressed(uint16_t GPIO_Pin);
- UART_IT_IDLE – переривання UART, виникає при відсутності вхідних даних протягом довжини одного UART-слова, служить сигналом кінця отриманих даних;
- UART_IT_TC – виникає після завершення відправки даних через UART.

3.2.4 Протокол зв’язку з мобільним телефоном

Після налаштування програмного забезпечення можемо переходити до спряження мобільного телефону з системою. Для початку було розроблено схему алгоритму, по якій було налаштовано алгоритм роботи спряження плати з мобільним телефоном та передачі команд від телефону до пристрою. В налаштуваннях Bluetooth знаходимо пристрій. А після підключення до пристрою за допомогою Bluetooth з мобільного телефону можна надсилати команди, описані в табл. 3.1.

Таблиця 3.1 – Перелік команд, які можна надсилати з мобільного телефону

Дія	Команда	Відповідь
Запит часу пристрою	TIME	“HH:MM:SS”, де HH – години, MM – хвилини, SS - секунди
Встановлення часу пристрою	SET*HHMMSS* де HH – години, MM – хвилини, SS – секунди. Символ ‘*’ не є частиною команди	Немає
Запит температури	TEMP	“TT”, де TT – температура в градусах Цельсія
Запит тиску	PRES	“PPPPPP”, де PPPPPP – тиск в паскалях
Запит освітленості	LIGHT	“LLLLL”, де “LLLLL” – освітленість в люксах

Отже при підключенні до плати ми можемо безпосередньо керувати нею на відстані.

Висновок до розділу 3

В результаті виконання досліджень, направлених на розробку принципової та функціональної схем було виконано:

Складення списку необхідних модулів для стабільного функціонування системи;

- Досліджено склад модулів відладочної плати stm32f103c6t8-modul;
- Дослідження дисплею 5641-AG, його принципової схеми, та кодування його сегментів;
- Дослідження характеристик датчика атмосферного тиску BMP180, датчика освітлення ADPS-9930 та Bluetooth-модуля JDY-10;
- Проведено розрахунки по визначенню опорів резисторів, які використовуються в схемі.

Як висновок, було розробрено принципову схему пристрою зі всіма підключеннями. Після чого, завдяки розрахункам, була розроблена електрична функціональна схема з налаштуванням тактових частот для STM32.

Також у данному розділі була створена схема алгоритму програми, по якій було налагоджено підключення до схеми мобільного телефону.

РОЗДІЛ 4. ТЕСТУВАННЯ ПРИСТРОЮ У РІЗНИХ РЕЖИМАХ

Після створення функціональних схем та підключення всіх модулів до плати, ми можемо приступати до тестування приладу.

4.1 Інструкція по експлуатації

Після подачі живлення до приладу починає світитися дисплей, а також червоний індикатор живлення на мікросхемі. Прилад має чотири кнопки: «зміна режиму», «вверх», «вниз» та «ввід», а також два режими: «режим годинника» та «режим секундоміра». Одразу після початку роботи прилад перебуває в режимі годинника.

Для підключення до приладу з інтерфейсу мобільного телефону, достатньо лише ввімкнути Bluetooth та знайти плату в списку запропонованих пристроїв.

4.1.1 Режим годинника

Індикатором цього режиму є крапка після другої цифри на дисплеї, що блимає з секундним інтервалом. Якщо натиснути кнопку вводу, можна перейти до налаштування часу. Цифру, яка блимає, можна змінити за допомогою кнопок вверх та вниз. Натискання вводу дозволить налаштувати наступну цифру, аж доки всі не будуть налаштовані. Під час налаштування зміна режиму неможлива.

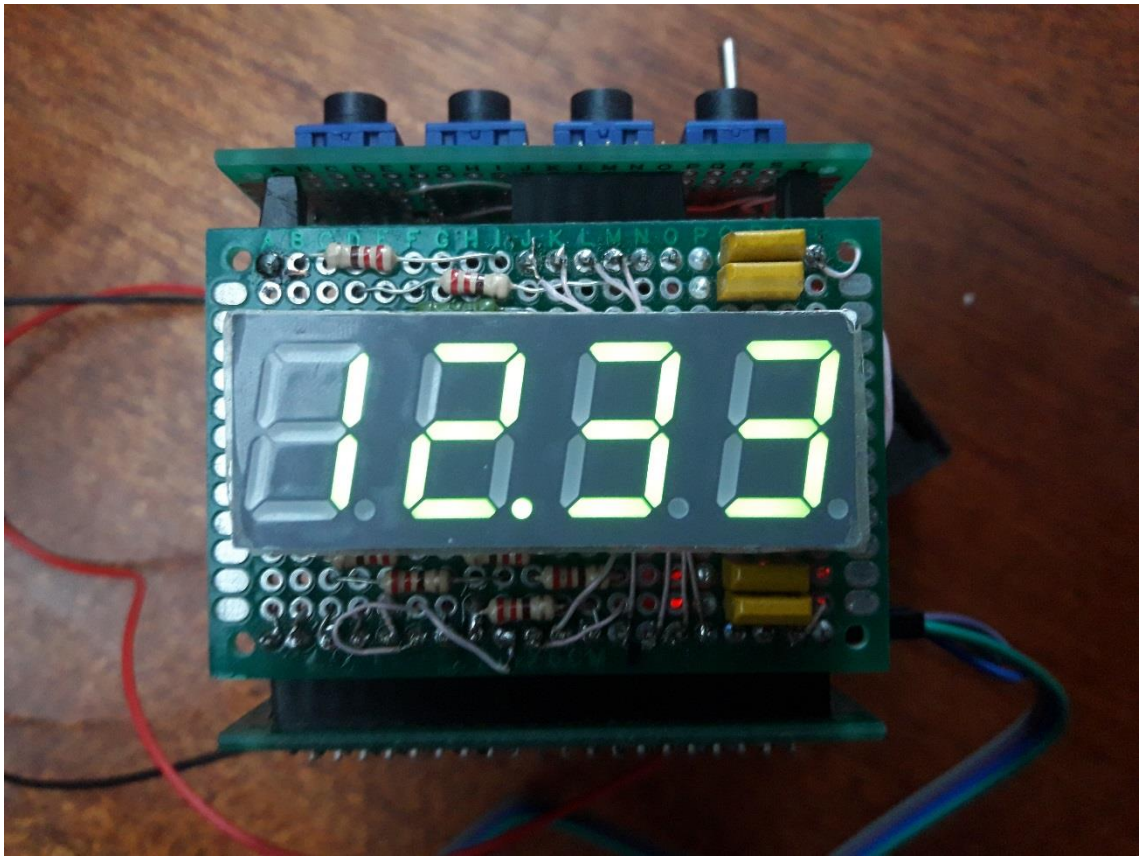


Рисунок 4.1 - Режим годинника

4.1.2 Режим секундоміра

В цьому режимі крапка після другої цифри блимає значно частіше, ніж в режимі годинника. Крайня права крапка буде світитись, якщо секундомір зупинено. Натискання кнопки вверх запускає секундомір або продовжує відлік, якщо він був зупинений. Кнопка вниз зупиняє відлік або скидає секундомір, якщо той вже стоїть.

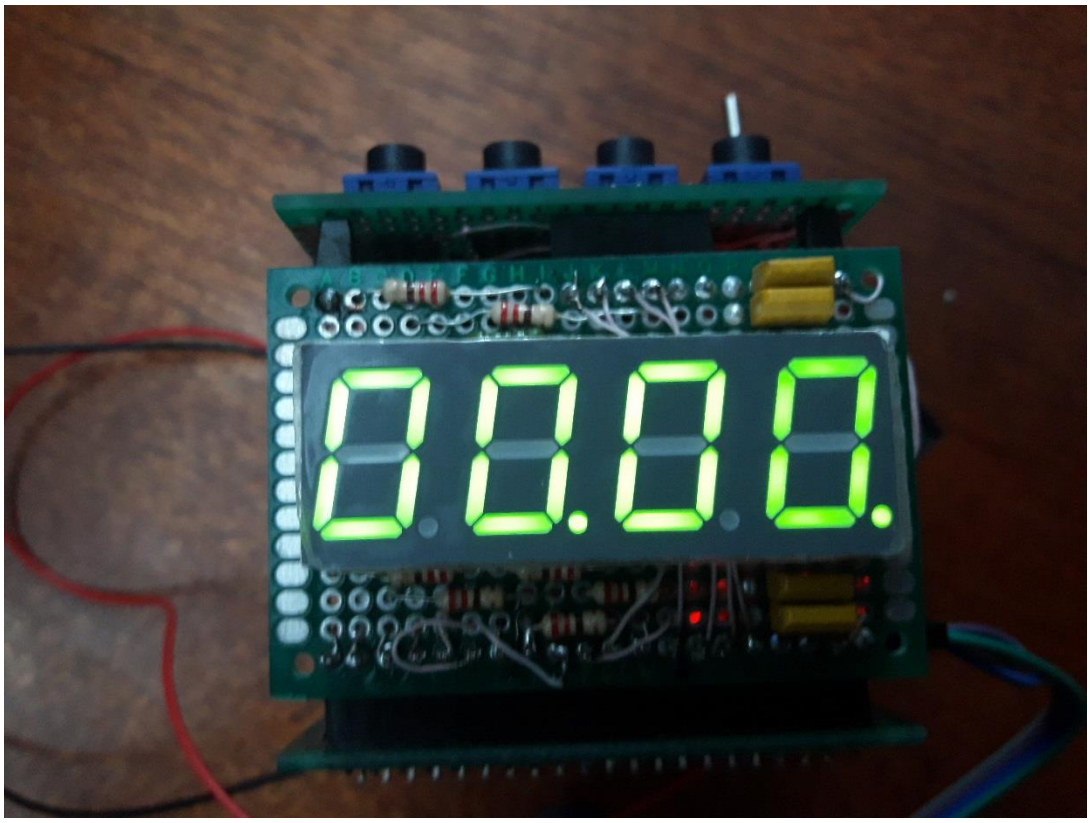


Рисунок 4.2 - Режим секундоміра

4.1.3 Режим виміру температури

Показує температуру, зчитану з датчика, в градусах Цельсія після літери t. Округлення здійснюється до цілих чисел.

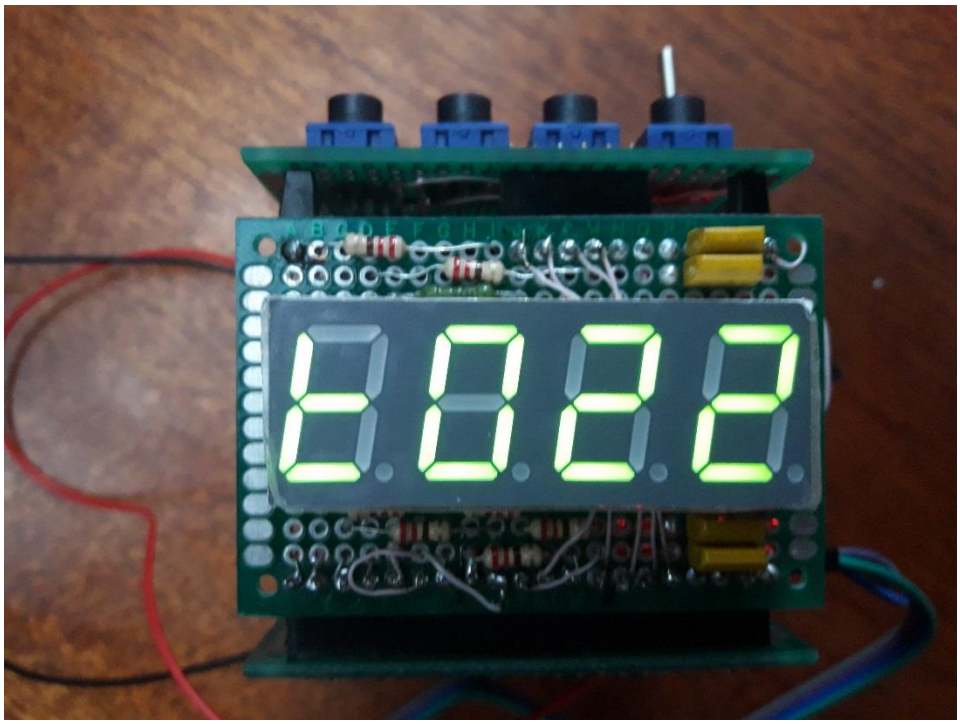


Рисунок 4.3 - Режим виміру температури

4.1.4 Режим виміру тиску

Показує тиск, зчитаний з датчика, в міліметрах ртутного стовпчика після літери Р.

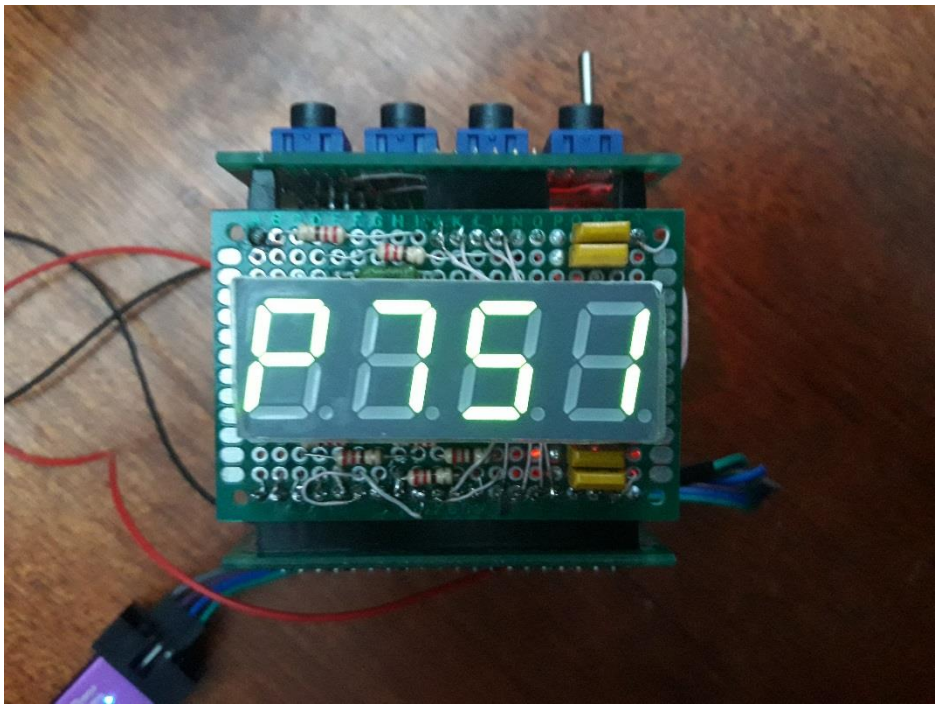


Рисунок 4.4 - Режим виміру тиску

4.1.5 Режим виміру освітлення

Показує рівень освітлення, зчитаний з датчика, в люксах після літери L. Кожна точка, що світиться, означає, що число на дисплеї має бути збільшено в 10 разів.

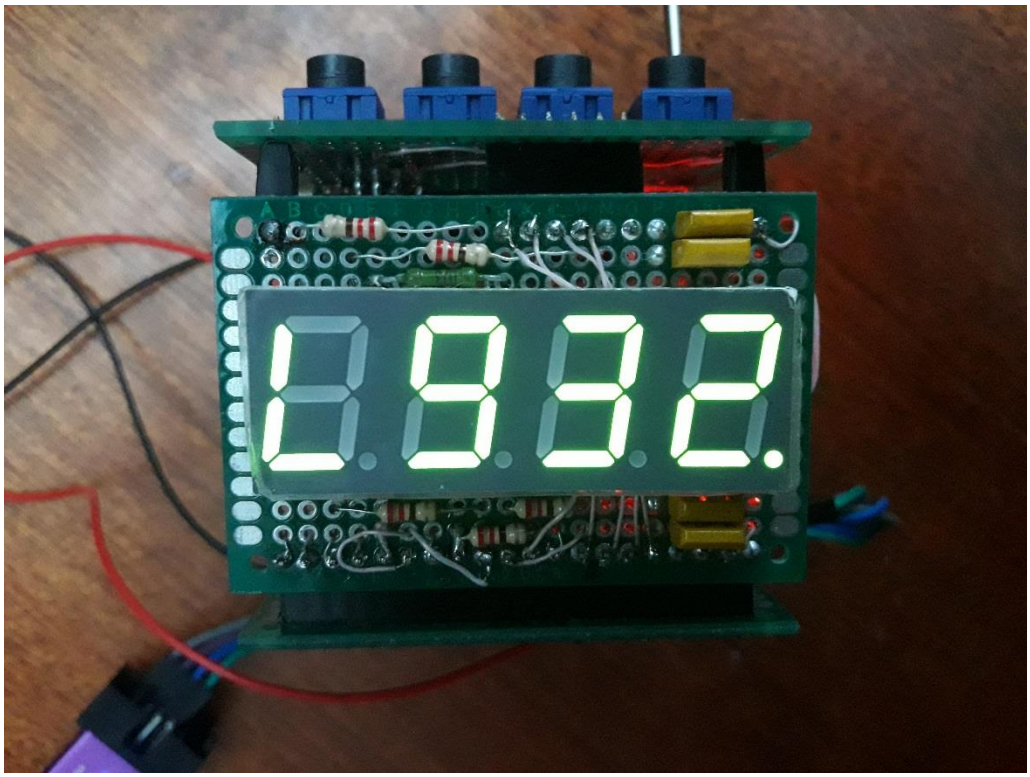


Рисунок 4.5 - Режим виміру освітлення

Висновок до розділу 4

В результаті виконання дипломної роботи було розглянуто можливості мікроконтролера stm32f103c8t6, DMA-контролера, шини даних I2C, інтерфейсу UART та датчиків тиску і освітлення.

В результаті розгляду, було визначено основні прицнипи роботи з даними модулями та створено принципову схему на базі STM32.

Після проведення потрібних розрахунків було створено функціональну електричну схему для розрахунку тактових частот на платі.

Перед створенням ПЗ було виконано схему алгоритму програми, для передачі інформації за допомогою технології bluetooth.

Також було розроблено програмне забезпечення для функціонування пристрою, що виконує функції годинника, термометра, барометра та датчика освітлення, а також створено працездатний зразок приладу.

Основні результати проведеної роботи полягають у наступному:

- Досліджено можливості та специфікацію мікроконтролера stm32f103c6t8.
- Розроблено принципову схему з використанням мікроконтролера, датчика температури і тиску, датчика освітлення та Bluetooth-модуля.
- Реалізовано програмне забезпечення для взаємодії мікроконтролера з периферійними пристроями.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Aghdasi F. DMA controller design using self-clocked methodology / F. Aghdasi, A. Bhasin // 7th Africon Conference in Africa / F. Aghdasi, A. Bhasin., 2004. – С. 443–450.

2. Mankar J. Review of I2C protocol / J. Mankar, C. Darode, K. Trivedi. // International Journal of Research in Advent Technology. Volume 2. – 2014. – №1. – С. 474–479.

3. Garima C. Design & development of daughter board for Raspberry Pi to support Bluetooth communication using UART / C. Garima, N. Agarwal, H. Reddy // International Conference on Computing, Communication & Automation / C. Garima, N. Agarwal, H. Reddy., 2015. – С. 949–954.

4. Dhanadravye A. D. 4. A Review on Implementation of UART using Different Techniques / A. D. Dhanadravye, S. S. Thorat. // International Journal of Computer Science and Information Technologies. – 2014. – №5. – С. 394–396.

5. Design of a micro-UART for SoC application / [A. Liakot, R. Sidek, A. Ishak та ін.] // Computers & Electrical Engineering, Volume 30, Issue 4 / [A. Liakot, R. Sidek, A. Ishak та ін.]., 2004. – С. 257–268.

6. Zhang H. Design of the Data Acquisition System Based on STM32 / H. Zhang, W. Kang // Procedia Computer Science, Vol. 17 / H. Zhang, W. Kang., 2013. – С. 222–228.

7. Lutkevich B. Microcontroller (MCU) [Електронний ресурс] / Ben Lutkevich – Режим доступу до ресурсу: https://internetofthingsagenda.techtarget.com/definition/microcontroller.

8. Zhang T. S. Data Acquisition Design Based on the STM32 / T. S. Zhang, Y. M. Li // Advanced Materials Research / T. S. Zhang, Y. M. Li., 2012. – С. 591–593, 1527–1530.

9. Research of Hand-Held Urine Analyzer Based on Photoelectric Technology and STM32F103 Embedded Systems / [Y. Yingmin, C. Chong, W. Guichu та ін.] // International Conference on Computational and Information Sciences / [Y. Yingmin, C. Chong, W. Guichu та ін.]., 2010. – С. 174–177.

10. Development of an Energy-Efficient Smart Socket Based on STM32F103. / [M. Huang, B. Wang, J. Chen та ін.] // Applied Sciences / [M. Huang, B. Wang, J. Chen та ін.]., 2018. – С. 15.

11. The Design of Musical Instrument Tuning System. Based on stm32f103 Microcomputer / L.Zhongming, L. Zhuofu, Z. Jia, S. Chunying // International Conference on Measurement, Information and Control (MIC) / L.Zhongming, L. Zhuofu, Z. Jia, S. Chunying., 2012. – С. 79–82.

12. Osborne A. An Introduction to Microcomputers / Adam Osborne – California, 1976. – (Volume 1). – С. 116–126.

13.Kistler M. Cell Multiprocessor Communication Network / Michael Kistler // Extensive benchmarks of DMA performance in Cell Broadband Engine. / Michael Kistler. – Barselona, 2006. – С. 10–23.

14. Teel J. Introduction to the STM32 Blue Pill (STM32duino) [Електронний ресурс] / John Teel // Resources for entrepreneurs developing new electronic hardware products.. – 2020. – Режим доступу до ресурсу: https://predictabledesigns.com/introduction-stm32-blue-pill-stm32duino/.

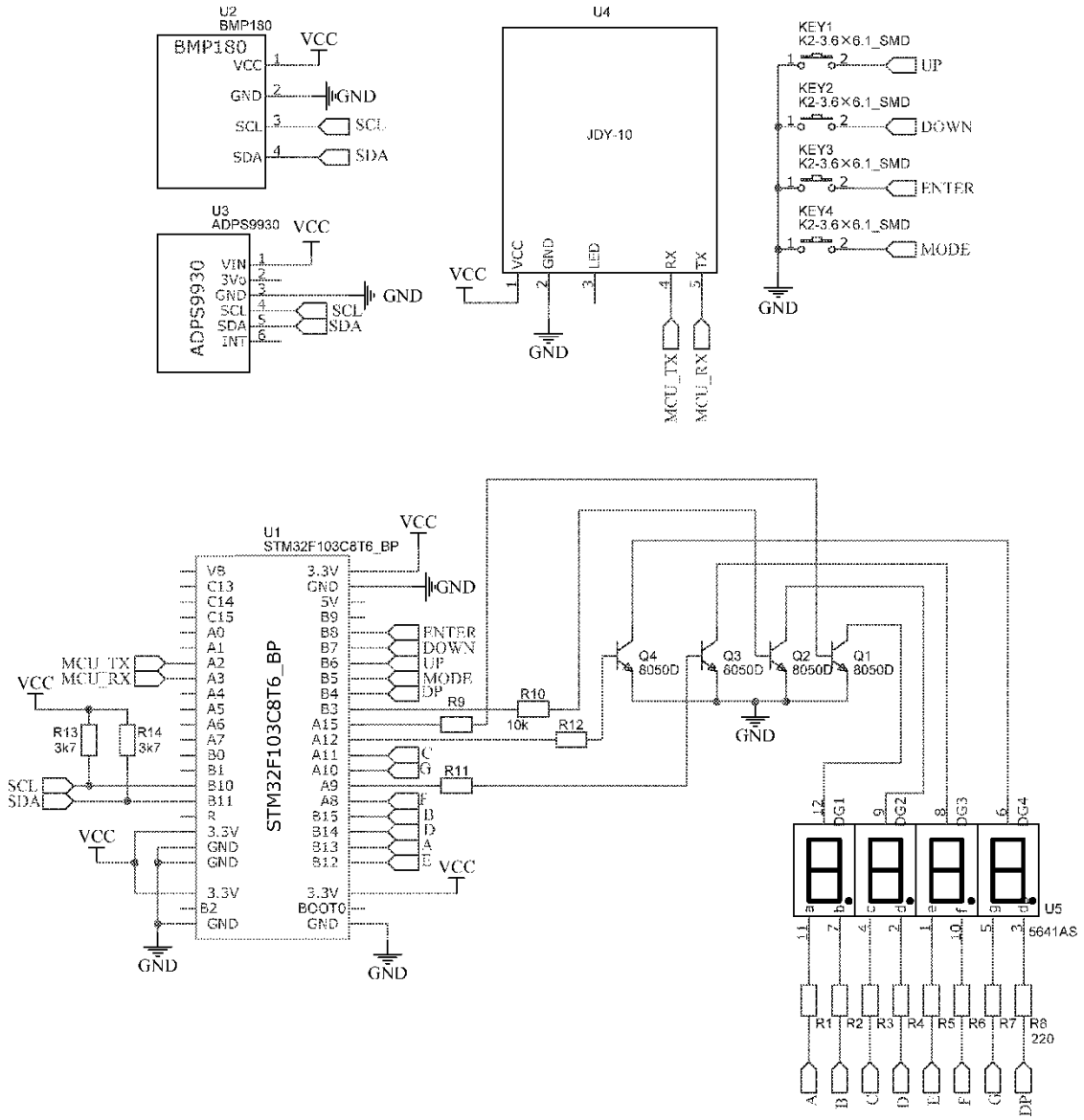
ДОДАТОК 1

Принципова схема

ІАЛЦ.467200.004 Е1

Листів 1

2021



					ІАЛЦ.467200.004 Е1	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розробив		Музика М.О.			*ІоТ пристрій на базі мікроконтролера STM32. Принципова схема.*	Літ. / Аркуш 1 / Аркушів 1
Перевір.		Ткаченко В.В.				КПІ ім. Ігоря Сікорського ФІОТ ІО-72
Н. контр.		Симоненко В.П.				
Затверд.		Стіренко С.Г.				

ДОДАТОК 2

Схема алгоритму програми

ІАЛЦ.467200.005 Е2

Листів 1

2021

					ІАЛЦ.467200.005 Е2		
Змін.	Арк.	№ докум.	Підпис	Дата			
Розробив		Музика М.О.			*Процедура отримання інформації через Bluetooth.* Схема алгоритму програми	Літ.	Аркуш 1 / Аркушів 1
Перевір.		Ткаченко В.В.					
Н. контр.		Симоненко В.П.				КПІ ім. Ігоря Сікорського ФІОТ ІО-72	
Затверд.		Стіренко С.Г.					

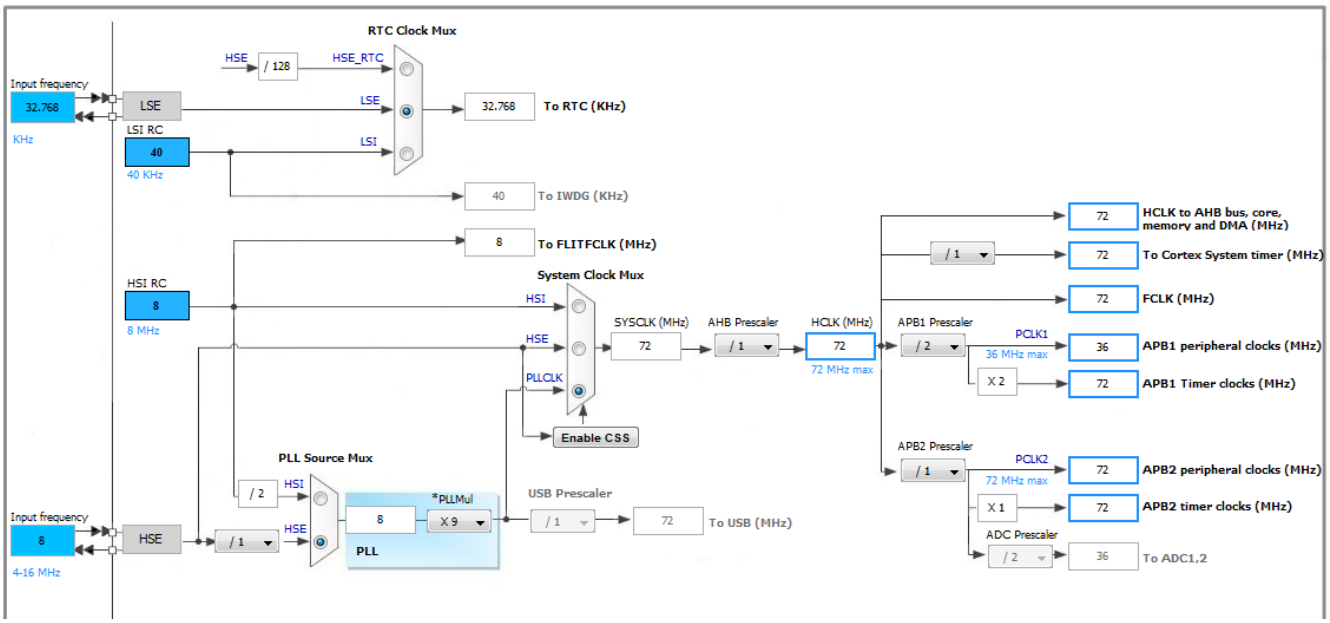
ДОДАТОК З

Функціональна схема

ІАЛЦ.467200.005 Е3

Листів 1

2021



					ІАЛЦ.467200.005 Е2	
Змін.	Арк.	№ докум.	Підпис	Дата		
Розробив		Музика М.О.			*Налаштування тактових частот для STM32.* Схема електрична функціональна.	Літ. / Аркуш 1 / Аркушів 1
Перевір.		Ткаченко В.В.				КПІ ім. Ігоря Сікорського ФІОТ ІО-72
Н. контр.		Симоненко В.П.				
Затверд.		Стіренко С.Г.				

ДОДАТОК 4

Лістинг програми

ІАЛЦ.467200.007 Е4

Листів 8

2021

clockController.h

```
#ifndef _CLOCK_CONTROLLER_
#define _CLOCK_CONTROLLER_

#include "stm32f1xx_hal.h"

typedef enum
{
	SEGMENT_OFF = 0,
	SEGMENT_ON = 1,
	SEGMENT_BLINK_1,
	SEGMENT_BLINK_2,
	SEGMENT_ALWAYS_ON,
	SEGMENT_ALWAYS_OFF
} SegmentState;

typedef enum
{
	MODE_TIME,
	MODE_TIME_CONFIG,
	MODE_STOPWATCH
} ClockMode;

void mainLoop();
void buttonPressed(uint16_t GPIO_Pin);
void switchDisplay();

#endif
```

clockController.c

```
#include "clockController.h"
#include "stm32f1xx_hal.h"

//prototypes
void setModeConfig();
void checkButtonsPressed();

void onButtonModePressed();
void onButtonUpPressed();
void onButtonDownPressed();
void onButtonEnterPressed();

//prototypes end

//constants
#define CHAR_ARRAY_LENGTH 16

uint16_t keysPins[4] = {KEY_1_Pin, KEY_2_Pin, KEY_3_Pin, KEY_4_Pin};
GPIO_TypeDef* keysPorts[4] = {KEY_1_GPIO_Port, KEY_2_GPIO_Port, KEY_3_GPIO_Port,
KEY_4_GPIO_Port};

uint16_t segPins[8] = {SEG_A_Pin, SEG_B_Pin, SEG_C_Pin, SEG_D_Pin, SEG_E_Pin,
SEG_F_Pin, SEG_G_Pin, SEG_DP_Pin};
GPIO_TypeDef* segPorts[8] = {SEG_A_GPIO_Port, SEG_B_GPIO_Port, SEG_C_GPIO_Port,
SEG_D_GPIO_Port, SEG_E_GPIO_Port, SEG_F_GPIO_Port, SEG_G_GPIO_Port,
SEG_DP_GPIO_Port};

static __IO char charsArray[CHAR_ARRAY_LENGTH] = {'0', '1', '2', '3', '4', '5', '6',
'7', '8', '9', 'H', 'E', 'L', 'S', 'I', 't'};
```

```
static __IO uint8_t charsCodesArray[CHAR_ARRAY_LENGTH] = {252, 96, 218, 242, 102,
182, 190, 224, 254, 246, 110, 158, 28, 182, 12, 30};

//constants end

extern TIM_HandleTypeDef htim1;
extern RTC_HandleTypeDef getHRTC();


uint8_t digitCounter = 0;
uint8_t digitConfigable;
char symbols[4];

uint8_t dotsMode[4] = {SEGMENT_ALWAYS_OFF, SEGMENT_ALWAYS_OFF, SEGMENT_ALWAYS_OFF,
SEGMENT_ALWAYS_OFF};
uint8_t digitsMode[4] = {SEGMENT_ALWAYS_ON, SEGMENT_ALWAYS_ON, SEGMENT_ALWAYS_ON,
SEGMENT_ALWAYS_ON};

RTC_TimeTypeDef rtcTime;
RTC_HandleTypeDef hrtc;
ClockMode clockMode;
uint8_t stateChanged = 0;

volatile uint8_t buttonsPressedCounter[4] = {0,0,0,0};

uint32_t stopwatchStart = 0;
uint32_t stopwatchStop = 0;

void SetCharPins(char ch) {
	//reset all pins
	for (int i = 0; i < 8; i++) {
		HAL_GPIO_WritePin(segPorts[i], segPins[i], GPIO_PIN_RESET);
	}

	uint8_t currentDigitState = 0;
	if (digitsMode[digitCounter] == SEGMENT_ALWAYS_ON)
		currentDigitState = 1;
	else if (digitsMode[digitCounter] == SEGMENT_BLINK_1)
		currentDigitState = (HAL_GetTick() / 1000) % 2;
	else if (digitsMode[digitCounter] == SEGMENT_BLINK_2)
		currentDigitState = (HAL_GetTick() / 250) % 2;

	if (currentDigitState) {
		uint8_t code = 0;
		for (int i = 0; i < CHAR_ARRAY_LENGTH; i++)
			if (charsArray[i] == ch)
				code = charsCodesArray[i];

		if (code & (1 << 7))
			HAL_GPIO_WritePin(SEG_A_GPIO_Port, SEG_A_Pin, GPIO_PIN_SET);
		if (code & (1 << 6))
			HAL_GPIO_WritePin(SEG_B_GPIO_Port, SEG_B_Pin, GPIO_PIN_SET);
		if (code & (1 << 5))
			HAL_GPIO_WritePin(SEG_C_GPIO_Port, SEG_C_Pin, GPIO_PIN_SET);
		if (code & (1 << 4))
			HAL_GPIO_WritePin(SEG_D_GPIO_Port, SEG_D_Pin, GPIO_PIN_SET);
		if (code & (1 << 3))
			HAL_GPIO_WritePin(SEG_E_GPIO_Port, SEG_E_Pin, GPIO_PIN_SET);
		if (code & (1 << 2))
			HAL_GPIO_WritePin(SEG_F_GPIO_Port, SEG_F_Pin, GPIO_PIN_SET);
```

```
		if (code & (1 << 1))
			HAL_GPIO_WritePin(SEG_G_GPIO_Port, SEG_G_Pin, GPIO_PIN_SET);
	}


	if (dotsMode[digitCounter] == SEGMENT_ALWAYS_ON)
		HAL_GPIO_WritePin(SEG_DP_GPIO_Port, SEG_DP_Pin, GPIO_PIN_SET);
	else if (dotsMode[digitCounter] == SEGMENT_BLINK_1 && ((HAL_GetTick() / 1000)
% 2))
		HAL_GPIO_WritePin(SEG_DP_GPIO_Port, SEG_DP_Pin, GPIO_PIN_SET);
	else if (dotsMode[digitCounter] == SEGMENT_BLINK_2 && ((HAL_GetTick() / 250) %
2)) {
		HAL_GPIO_WritePin(SEG_DP_GPIO_Port, SEG_DP_Pin, GPIO_PIN_SET);
	}
}


void timeToCharArray(char* symbols, uint8_t hours, uint8_t minutes) {
	symbols[0] = (hours / 10) + '0';
	symbols[1] = (hours % 10) + '0';
	symbols[2] = (minutes / 10) + '0';
	symbols[3] = (minutes % 10) + '0';
}

void clockInit() {
	hrtc = getHRTC();

	rtcTime.Hours = 0;
	rtcTime.Minutes = 0;
	rtcTime.Seconds = 0;

	HAL_RTC_SetTime(&hrtc, &rtcTime, RTC_FORMAT_BIN);

	clockMode = MODE_TIME;
	setModeConfig();
}

void mainLoop() {

	while (1) {
		if (clockMode == MODE_TIME) {
			HAL_RTC_GetTime(&hrtc, &rtcTime, RTC_FORMAT_BIN);
			timeToCharArray(symbols, rtcTime.Hours, rtcTime.Minutes);
		}

		else if (clockMode == MODE_TIME_CONFIG) {
			HAL_RTC_SetTime(&hrtc, &rtcTime, RTC_FORMAT_BIN);
			timeToCharArray(symbols, rtcTime.Hours, rtcTime.Minutes);
		}

		else if (clockMode == MODE_STOPWATCH) {
			uint8_t minutes = 0, seconds = 0;
			if (stopwatchStart) {
				uint32_t time;
				if (stopwatchStop)
					time = stopwatchStop - stopwatchStart;
				else
					time = HAL_GetTick() - stopwatchStart;

				time /= 1000;
```

```
                        seconds = time % 60;
                        minutes = (time / 60) % 60;
                    }
                    timeToCharArray(symbols, minutes, seconds);
                }

                checkButtonsPressed();

                if (stateChanged) {
                    setModeConfig();
                    stateChanged = 0;
                }

        }
}

void setModeConfig() {
        if (clockMode == MODE_TIME) {
                for (int i = 0; i < 4; i++) {
                    digitsMode[i] = SEGMENT_ALWAYS_ON;
                    dotsMode[i] = SEGMENT_ALWAYS_OFF;
                }
                dotsMode[1] = SEGMENT_BLINK_1;
        }

        else if (clockMode == MODE_TIME_CONFIG) {
                for (int i = 0; i < 4; i++) {
                    digitsMode[i] = SEGMENT_ALWAYS_ON;
                    dotsMode[i] = SEGMENT_ALWAYS_OFF;
                }
                dotsMode[1] = SEGMENT_BLINK_1;
                digitsMode[digitConfigable] = SEGMENT_BLINK_2;
        }

        else if (clockMode == MODE_STOPWATCH) {
                for (int i = 0; i < 4; i++) {
                    digitsMode[i] = SEGMENT_ALWAYS_ON;
                    dotsMode[i] = SEGMENT_ALWAYS_OFF;
                }
                dotsMode[1] = SEGMENT_BLINK_2;
                if (stopwatchStart && !stopwatchStop)
                    dotsMode[3] = SEGMENT_ALWAYS_OFF;
                else
                    dotsMode[3] = SEGMENT_ALWAYS_ON;
        }
}

void checkButtonsPressed() {
        if (buttonsPressedCounter[0]) {
                buttonsPressedCounter[0]--;
                onButtonModePressed();
        }

        if (buttonsPressedCounter[1]) {
                buttonsPressedCounter[1]--;
                onButtonUpPressed();
        }

        if (buttonsPressedCounter[2]) {
                    buttonsPressedCounter[2]--;
```

```
                onButtonDownPressed();
        }

    if (buttonsPressedCounter[3]) {
        buttonsPressedCounter[3]--;
        onButtonEnterPressed();
    }
}


void onButtonModePressed() {
    if (clockMode == MODE_TIME)
        clockMode = MODE_STOPWATCH;

    else if (clockMode == MODE_STOPWATCH)
        clockMode = MODE_TIME;

    stateChanged = 1;
}


void onButtonUpPressed() {
    if (clockMode == MODE_TIME_CONFIG) {
        if (digitConfigable == 0) {
            uint8_t hours = rtcTime.Hours;
            hours += 10;
            if (hours / 10 > 2)
                hours -= 30;
            rtcTime.Hours = hours;
        }

        else if (digitConfigable == 1) {
            uint8_t hours = rtcTime.Hours;
            if ((hours % 10 == 9) || ((hours / 10 == 2) && (hours % 10 >=
3)))
                hours = hours - hours % 10;
            else
                hours += 1;
            rtcTime.Hours = hours;
        }

        else if (digitConfigable == 2) {
            uint8_t minutes = rtcTime.Minutes;
            minutes += 10;
            if (minutes >= 60)
                minutes -= 60;
            rtcTime.Minutes = minutes;
        }

        else if (digitConfigable == 3) {
            uint8_t minutes = rtcTime.Minutes;
            if (minutes % 10 == 9)
                minutes -= 9;
            else
                minutes += 1;
            rtcTime.Minutes = minutes;
        }
    }

    else if (clockMode == MODE_STOPWATCH) {
```

```
			if (stopwatchStart == 0) {
				stopwatchStart = HAL_GetTick();
			}
			else if (stopwatchStart && stopwatchStop) {
				stopwatchStart = HAL_GetTick() + stopwatchStart - stopwatchStop;
				stopwatchStop = 0;
			}
			dotsMode[3] = SEGMENT_ALWAYS_OFF;
		}
	}


	void onButtonDownPressed() {
		if (clockMode == MODE_TIME_CONFIG) {
			if (digitConfigable == 0) {
				uint8_t hours = rtcTime.Hours;

				if (hours < 10)
					hours += 20;
				else
					hours -= 10;

				rtcTime.Hours = hours;
			}

			else if (digitConfigable == 1) {
				uint8_t hours = rtcTime.Hours;
				if (hours % 10 == 0) {
					if (hours / 10 == 2)
						hours += 3;
					else
						hours += 9;
				}
				else
					hours -= 1;
				rtcTime.Hours = hours;
			}

			else if (digitConfigable == 2) {
				uint8_t minutes = rtcTime.Minutes;

				if (minutes < 10)
					minutes += 50;
				else
					minutes -= 10;
				rtcTime.Minutes = minutes;
			}

			else if (digitConfigable == 3) {
				uint8_t minutes = rtcTime.Minutes;
				if (minutes % 10 == 0)
					minutes += 9;
				else
					minutes -= 1;
				rtcTime.Minutes = minutes;
			}
		}

		else if (clockMode == MODE_STOPWATCH) {
			if (stopwatchStop == 0) {
```

```
                stopwatchStop = HAL_GetTick();
            }
            else {
                stopwatchStart = 0;
                stopwatchStop = 0;
            }
            dotsMode[3] = SEGMENT_ALWAYS_ON;
        }
}


void onButtonEnterPressed() {
        if (clockMode == MODE_TIME) {
            clockMode = MODE_TIME_CONFIG;
            digitConfigable = 0;
            stateChanged = 1;
        }

        else if (clockMode == MODE_TIME_CONFIG) {
            digitConfigable++;
            stateChanged = 1;
            if (digitConfigable >= 4) {
                clockMode = MODE_TIME;
                rtcTime.Seconds = 0;
            }
        }
}

void buttonPressed(uint16_t GPIO_Pin) {
        if (GPIO_Pin == ButtonMode_Pin)
            buttonsPressedCounter[0]++;
        else if (GPIO_Pin == ButtonUp_Pin)
            buttonsPressedCounter[1]++;
        else if (GPIO_Pin == ButtonDown_Pin)
            buttonsPressedCounter[2]++;
        else if (GPIO_Pin == ButtonEnter_Pin)
            buttonsPressedCounter[3]++;
}

void switchDisplay() {
        //close key
        HAL_GPIO_WritePin(keysPorts[digitCounter], keysPins[digitCounter],
GPIO_PIN_RESET);

        digitCounter++;

        if (digitCounter > 3)
            digitCounter = 0;

        //set segments
        SetCharPins(symbols[digitCounter]);

        //open key
            HAL_GPIO_WritePin(keysPorts[digitCounter], keysPins[digitCounter],
GPIO_PIN_SET);
}
```

buttonHandlers.h

```
#ifndef BUTTONHANDLERS_H_
#define BUTTONHANDLERS_H_
```

```c
#include "stm32f1xx_hal.h"

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin);

#endif /* BUTTONHANDLERS_H_ */
```

buttonHandlers.c

```c
#include "buttonHandlers.h"
#include "clockController.h"
#include "stm32f1xx_hal.h"

#define BUTTONS_NUMBER 4

uint16_t buttonsPinsArray[4] = {ButtonMode_Pin, ButtonUp_Pin, ButtonDown_Pin,
ButtonEnter_Pin};
uint32_t buttonsLastTimePressed[4] = {0, 0, 0, 0};

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
	for (int i = 0; i < BUTTONS_NUMBER; i++) {
		if (GPIO_Pin == buttonsPinsArray[i]) {
			if (HAL_GetTick() - buttonsLastTimePressed[i] < 100) {
				buttonsLastTimePressed[i] = HAL_GetTick();
				return;
			}

			buttonsLastTimePressed[i] = HAL_GetTick();

			buttonPressed(GPIO_Pin);
		}
	}

}
```

timerHandlers.h

```c
#ifndef TIMERHANDLERS_H_
#define TIMERHANDLERS_H_

#include "stm32f1xx_hal.h"

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);

#endif /* TIMERHANDLERS_H_ */
```

timerHandlers.c

```c
#include "timerHandlers.h"
#include "clockController.h"

extern TIM_HandleTypeDef htim1;

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
	if (htim == &htim1) {
		switchDisplay();
	}
}
```