Зміст

					ДК61.466219.001ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розробив		Сільчук В.І.			Арифметичний пристрій	Літ.	Арк. 1 / Аркушів 58
Перевірив							
Реценз.						КПІ ім. І.Сікорського, ФЕЛ, КЕОА, гр. ДК-61	
Н. Контр.							
Затвердив		Корнєв В.П.					

Вступ

Пристрій, що поставлено за мету розробити в рамках даної розрахунково-графічної роботи, являє собою арифметичний пристрій, який виконує операції додавання, віднімання, множення та ділення над 8-розрядними числами.

Введення вхідних даних X та Y відбувається за допомогою двох тактових кнопок, діапазон значень вхідних даних — від 1 до 255, при кожному натисканні кнопки відповідне значення збільшується на одиницю. За допомогою окремої тактової кнопки відбувається вибір режиму роботи, шляхом короткочасного її натискання. Тривале натискання на цю кнопку призводить до скидання значень вхідних даних X та Y, а також вибору режиму роботи, до значень «за замовчуванням»: X = 1, Y = 1, режим роботи — додавання.

Введені значення вхідних даних, відомості про вибраний режим, а також значення результату обчислень постійно виводяться на дисплей.

В якості дисплею було вибрано рідкокристалічний (LCD — Liquid Crystal Display) дисплей типу 1602 — два рядки символів по 16 символів в кожному — на основі контролеру HD44780. Причиною вибору саме такого дисплею є його поширеність та доступність, невисока ціна, а також відносна простота в роботі з ним. Загалом, він достатньо і цілком підходить для реалізації поставленого завдання.

Виконання обчислень буде проводитись мікроконтролером STM32F401RE, розміщеним на налагоджувальній платі STM32F401 Nucleo. Він являє собою дуже потужний мікроконтролер з широкими можливостями, а також є широко розповсюдженим та популярним в наш час. Використання налагоджувальної плати значно спрощує роботу з мікроконтролером, так як на ній вже реалізована необхідна «обв'язка» МК у вигляді резисторів,

конденсаторів, зовнішніх кварцових генераторів та інших необхідних для старту та стабільної роботи мікроконтролера компонентів.Також на даній платі наявний завантажувач-налагоджувач ST-Link, який значно спрощує процес відлагодження роботи програми.

Програма керування пристроєм буде реалізована виключно з використанням бібліотеки CMSIS, з реалізацією, таким чином, прямого та послідовного доступу до конфігураційних регістрів керування периферією мікроконтролера. Такий підхід забезпечує максимально точну, після мови ассемблеру, а також повністю контрольовану та усвідомлювану розробником, роботу з периферією. Крім цього, важливим аргументом на користь використання бібліотеки CMSIS є отримання, при її використанні, максимально оптимального та компактного вихідного коду програми.

В даній роботі будуть розглянуті принципи та процес побудови пристрою, описаного вище, описані та наведені схема електрична принципова та програма керування.

Розділ 1. Опис структури пристрою та його складових

1.1. Структура пристрою.

Структурна схема пристрою наведена на наступному рисунку:

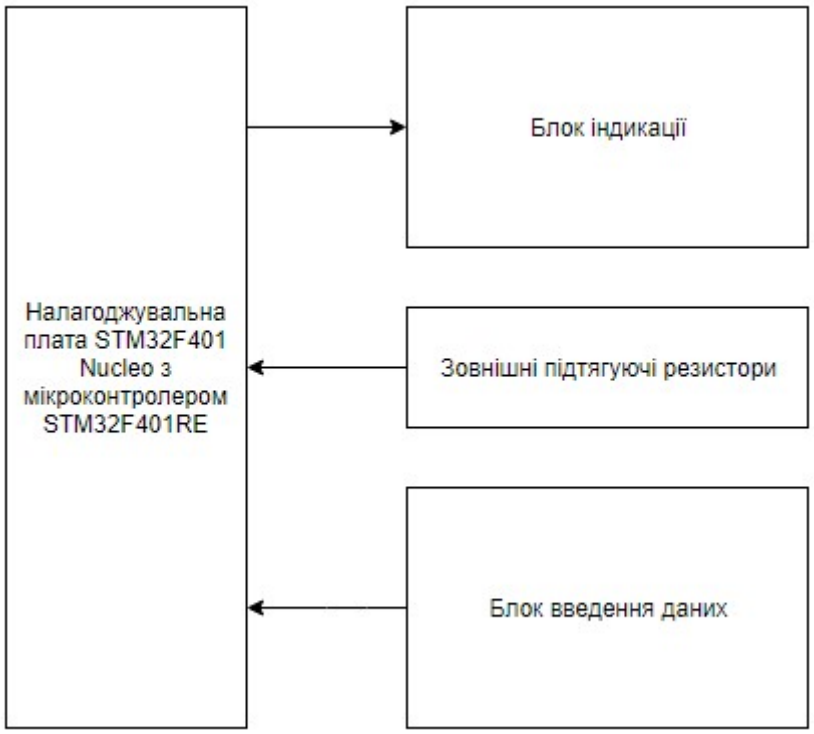

Рисунок 1.1 – Структурна схема пристрою.

Основним компонентом даного пристрою є мікроконтролер STM32F401RE, який розміщений на налагоджувальній платі STM32F401 Nucleo.

Блок введення даних являє собою 3 тактові кнопки з самоповерненням, позначені на схемі електричній принциповій пристрою, як SB1, SB2 та SB3 (див. Додаток __).

Блок зовнішніх підтягуючих резисторів містить резистори R1 та R3, необхідні для зовнішньої підтяжки до лінії живлення ліній даних та

тактового сигналу інтерфейсу I2C, що використовується для передачі даних блоку індикації.

Блок індикації складається з дисплею типу 1602 на основі контролера жидкокристалічного дисплею HD44780, а також розширювача портів вводу-виводу PCF8574 з інтерфейсом I2C.

1.2. Принцип введення даних.

Для введення значень вхідних даних X та Y використовуються тактові кнопки SB1 та SB2, для вибору режиму функціонування пристрою, а також для скидання введених значень вхідних даних — тактова кнопка SB3. Дані кнопки підключені до входів мікроконтролера PB13, PB14 та PB15, з внутрішньою підтяжкою даних пінів до живлення.

При кожному натисканні кнопки SB1 та SB2, відбувається збільшення відповідного значення вхідних даних на 1. При короткому за тривалістю натисканні на SB3 відбувається зміна режиму роботи пристрою, в порядку: додавання, віднімання, множення, ділення. При тривалому натисканні — більше 2 секунд — відбувається скидання поточних значень вхідних даних, а також поточного вибору режиму до значень «за замовчуванням» — X та Y встановлюються в «1», відбувається вибір режиму додавання. Тривалість натискання визначається за допомогою таймеру та системи змінних-«прапорців», з використанням переривань.

Реалізований також програмний захист від «дріб'язку контактів» — так званий «дебаунс» («debounce») тактових кнопок. Значення тривалості «дебаунсу» задається в програмі, і може бути зміненим за бажанням. За замовчуванням воно рівне 250 мс — на цей проміжок часу після натискання будь-якої з кнопок зчитування вхідних значень з неї та інших кнопок зупиняється, шляхом блокування зовнішніх переривань, які можуть бути

викликані натисканням даних тактових кнопок. Відлік інтервалу часу для «дебаунсу» реалізований за допомогою системного таймеру — SysTick, а також відповідних переривань системного таймеру.

1.3. Принцип відображення.

Індикація введених значень вхідних даних, відомостей про поточний режим роботи пристрою, а також результату обчислень, відбувається за допомогою рідкокристалічного дисплею типу 1602, на основі контролера HD44780.

Даний дисплей має 2 рядки з 16 знакомісцями в кожному.

Відображення потрібних даних на дисплеї відбувається постійно, при цьому оновлення даних, які виведені на дисплей, відбувається одразу після зміни користувачем значень вхідних даних шляхом натискання будь-якої із трьох використаних в схемі пристрою кнопок.

На дисплей виводяться: значення X — в верхньому лівому куті, значення Y — в нижньому лівому куті, відомості про поточний режим роботи пристрою — в верхньому правому куті, а також результат виконання арифметичних операцій над вхідними даними, згідно вибраного поточного режиму роботи — в нижньому правому куті дисплея.

Для відображення кожного із значень вхідних даних X та Y, а також результату, передбачено 5 знакомісць, тобто 5 десяткових розрядів.

Підключення дисплею до портів мікроконтролера відбувається за допомогою розширювача портів вводу-виводу PCF8574AT, зв'язок з яким відбувається за допомогою послідовного інтерфейсу передачі даних I2C, що дає змогу обмежитись всього двома, окрім ліній живлення та «землі»

розширювача портів вводу-виводу, лініями — даних (SDA) та тактового сигналу (SCL) — для реалізації даного зв’язку. Детальніше інтерфейс I2C та принцип роботи з розширювачем портів вводу-виводу PCF8574AT в контексті реалізації поставленого завдання, будуть описані далі, в відповідних розділах.

Для роботи з LCD дисплеєм, підключеним до мікроконтролера за допомогою PCF8574AT, використовується написана власноруч бібліотека `lcd1602.h`, з використанням у її реалізації виключно функцій бібліотеки CMSIS, за допомогою функцій, наданих якою, в програмі керування пристроєм відбувається як початкова конфігурація роботи дисплея, так і подальша передача йому потрібних інструкцій керування або ж даних для відображення.

Розділ 2. Проектування схеми електричної принципової

Схема електрична принципова ДК61.466219.001 Е3 пристрою наведена в Додатку __.

2.1. Вибір елементної бази для реалізації блоків пристрою.

Розглянемо детальніше вибір елементної бази пристрою.

Головним компонентом схеми, який і здійснює математичні операції, є потужний мікроконтролер STM32F401RE, який розміщений на налагоджувальній платі STM32F401 Nucleo.

Блок введення даних на схемі електричній принциповій пристрою представлений тактовими кнопками з самоповерненням SB1, SB2, SB3. Як видно зі схеми, дані кнопки підтягнуті внутрішніми резисторами R2, R4 та R5, опором 40 кОм, до лінії живлення мікроконтролера.

При складанні прототипу пристрою використовувались тактові кнопки TACTS-24N-F. Вони мають невеликі розміри а також низьку ціну. Позбавлення від «дріб'язку» контактів даних кнопок — програмне.

Блок зовнішніх підтягуючих резисторів — це резистори R1 та R3, опором 4.7 кОм. Дані резистори, як йшлося вище, підтягують лінії SDA та SCL до лінії живлення, що передбачено принципом функціонування послідовного інтерфейсу I2C.

При складанні прототипу в якості даних резисторів було використано виводні резистори МЛТ-0,5 та МЛТ-0,125, потужністю 0,5 та 0,125 Вт відповідно. В контексті даного пристрою, до цих резисторів не ставляться особливі вимоги, тому такий вибір є цілком допустимим.

Блок індикації представлений рідкокристалічним дисплеєм типу 1602 на основі контролера HD44780, а також розширювачем портів вводу-виводу

на базі мікросхеми PCF8574AT, що використовується для його підключення до мікроконтролера STM32F401RE, і дає змогу керувати дисплеєм з використанням послідовного інтерфейсу передачі даних I2C — тобто, використовуючи, як можна бачити з Рисунку 2.1, лише лінію даних SDA та лінію тактового сигналу SCL, а також лінії живлення та «землі» для живлення даних схем.

Мікросхема PCF8574AT, згідно документації, передбачає частоту тактового сигналу на лінії SCL, рівну 100 кГц. Адреса мікросхеми, в контексті комунікації по інтерфейсу I2C — 0x3F.

2.2. Налаштування системи тактування мікроконтролера.

Схему налаштування системи тактування мікроконтролера можна бачити на Рисунку 2.2.

Для генерації вихідного коду, який здійснюватиме конфігурацію роботи мікроконтролера згідно заданих параметрів, було використано спеціальний інструмент, який доступний для завантаження на веб-сайті STMicroelectronics.

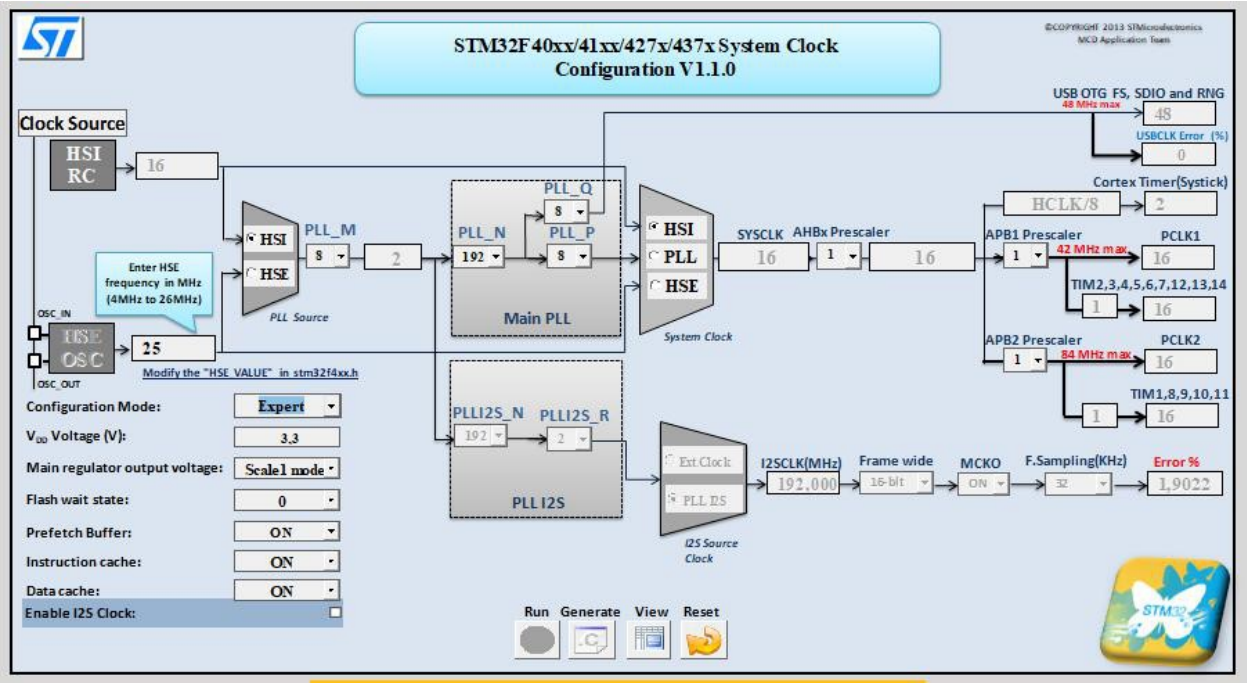

Рисунок 2.2 — Конфігурація системи тактування мікроконтролера.

Як можна бачити з рисунку, дана конфігурація передбачає роботу мікроконтролера з системною частотою (SYSCLK), рівною 16 МГц. Для цього використовується внутрішній високошвидкісний RC-генератор (HSI — High Speed Internal), який і забезпечує генерацію тактового сигналу з даною частотою. Значення коефіцієнтів переддільників для шин, до яких приєднана периферія, рівні 1, відповідно, системна частота не буде зменшуватись, і шини APB та AHB будуть тактуватись від частоти 16 МГц. Ця ж частота (або ж поділена на 8, залежно від налаштувань) буде тактувати і системний таймер SysTick.

Розділ 3. Розробка програми керування пристроєм

Блок-схема програми наведена в Додатку __, вихідний код компонентів програми наведено в Додатку __.

3.1. Призначення регістрів та змінних, а також використаних портів мікроконтролера.

Перелік використаних в програмі керування пристроєм регістрів, змінних та портів мікроконтролера, а також їх опис та призначення, наведено в Таблицях 3.1, 3.2 та 3.3.

Таблиця 3.1. Перелік та опис використаних портів мікроконтролера.

Регістр мікроконтролера	Призначення/опис
PB13	Підключення тактової кнопки для здійснення введення значення X
PB14	Підключення тактової кнопки для здійснення введення значення Y
PB15	Підключення тактової кнопки для здійснення вибору режиму функціонування пристрою, а також дл скидання значень введених даних та вибраного режиму роботи до значень «за замовчуванням»
PB8	Вихід лінії тактового сигналу (SCL) інтерфейсу I2C1, підключення зовнішнього підтягуючого резистору
PB9	Вихід лінії даних інтерфейсу (SDA) I2C1, підключення зовнішнього підтягуючого резистору

Продовження Таблиці 3.1.

5V	Підключення входу живлення LCD дисплею, підключення зовнішніх резисторів для підтяжки ліній SCL та SDA до напруги живлення.
GND	Підлючення «землі» до контактів тактових кнопок та відповідного входу LCD дисплею.

Таблиця 3.2. Перелік та опис використаних в програмі керування пристроєм регістрів.

Регістр	Призначення/опис
SysTick->CTRL	Регістр керування конфігурацією системного таймера
SysTick->LOAD	Значення, яке буде використане при перезавантаженні системного таймеру після його обнулення
SysTick->VAL	Поточне значення системного таймера (регістр лічби/рахунку)
EXTI->IMR	Регістр маскування зовнішніх переривань
RCC->APB1ENR	Регістр конфігурації тактування шини APB1, використовується для увімкнення тактування таймеру TIM2, також тактування модуля I2C1
TIM2->PSC	Налаштування значення переддільника таймера TIM2

TIM2->ARR	Регістр автоперезавантаження таймера TIM2
TIM2->DIER	Регістр налаштування дозволів переривань таймера TIM2 (DMA/Interrupt enable register)
TIM2->CR1	Регістр контролю (control register) таймера TIM2, використовується для вибору режиму рахунку «вгору»
TIM2->SR	Регістр статусу таймера TIM2, використовується для маніпуляцій з так званими «pending-бітами» в обробнику відповідних переривань
GPIOB->MODER	Конфігураційний регістр порту вводу-виводу B, використовується для вибору режиму роботи на вхід, а також для вибору режиму альтернативної функції для відповідних пінів порту B
GPIOB->PUPDR	Конфігураційний регістр порту вводу-виводу B, використовується для увімнення внутрішньої підтяжки до живлення пінів PB13, PB14 та PB15, а також для конфігурації пінів, які використовуються модулем I2C1

Продовження Таблиці 3.2.

RCC->APB2ENR	Регістр конфігурації тактування шини APB2, використовується для увімкнення тактування контролеру системної конфігурації (SYSCFG, System Configuration Controller), що здійснює керування системою зовнішніх переривань
SYSCFG->EXTICR	Регістр для налаштування вибору джерел зовнішніх переривань
EXTI->IMR	Регістр для налаштування маскування запитів на переривання від зовнішніх джерел
EXTI->FTSR	Регістр для налаштування вибору у якості «тригеру», який викличе переривання від зовнішнього джерела появу заднього фронту вхідного сигналу на вході відповідного джерела зовнішніх переривань
EXTI->PR	Регістр, що містить біти, які сигналізують про появу запиту на переривання на якійсь із ліній EXTI
RCC->AHB1ENR	Регістр для конфігурації тактування шини AHB1, використовується для увімкнення тактування порту вводу-виводу GPIOB

Продовження Таблиці 3.2.

GPIOB->AFR	Регістр налаштування вибору альтернативної функції пінів портів вводу-виводу, використовується для налаштування роботи пінів PB8 та PB9 в якості виходів ліній тактування та даних модуля I2C1
GPIOB->OTYPER	Конфігураційний регістр портів вводу виводу, використовується для налаштування режиму роботи пінів PB8 та PB9 в режимі «open drain»
I2C1->CR2	Конфігураційний регістр модуля I2C1 використовується для задання частоти тактування шини PB1, на якій розміщений даний модуль
I2C1->CCR	Конфігураційний регістр модуля I2C1 використовується для задання значення частоти вихідного сигналу тактування (Serial Clock — SCL), який буде генеруватись модулем I2C1, а також для вибору режиму роботи Fast/Standard
I2C1->TRISE	Конфігураційний регістр модуля I2C1 використовується для задання параметрів «таймінгів» сигналу SCL, який буде генеруватись відповідним модулем — значень тривалості фронті тощо

I2C1->CR1	Конфігураційний регістр модуля I2C1 використовується для увімкнення та зупинки роботи модулю I2C1, а також для ініціалізації старт та стоп-бітів
I2C1->DR	Регістр даних модулю I2C1, куди поміщаються дані, які мають бути відправлені по лінії даних
I2C1->SR1	Регістр статусу модуля I2C1, використовується для слідкування за процесом передачі/прийому даних відповідним модулем, шляхом перевірки значень відповідних бітових полів в даному регістрі, які змінюють свої значення відповідно до подій, які відбуваються при передачі/прийомі даних модулем — полів ініціалізації старт-біту, успішного надсилання адреси Slave-пристрою, сигналу про наявність вільного місця в регістрі даних, а також сигналу про закінчення передачі

Продовження Таблиці 3.2.

I2C1->SR2	Регістр статусу модуля I2C1, використовується для слідкування за процесом передачі/прийому даних відповідним модулем, шляхом перевірки значень відповідних бітових полів в даному регістрі, які змінюють свої значення відповідно до подій, які відбуваються при передачі/прийомі даних модулем

Таблиця 3.3. Перелік та опис використаних в програмі керування пристроєм глобальних змінних.

Глобальна	Призначення/опис
uint8_t LCD_show_ready	Свідчить про зміну значень вхідних даних — X, Y, режиму роботи пристрою, або ж про про скидання даних значень до значень «за замовчуванням» і потребу оновити виведені на дисплей відповідні значення
uint8_t debounce_ms_enable	«Прапорець», який свідчить про запуск роботи алгоритму ліквідації «дріб'язку контактів»
uint8_t pressure_flag	«Прапорець», який зберігає в собі значення, яке містилось в регістрі IDR для відповідного піна, до якого підключена тактова кнопка, для

	ідентифікації події тривалого натискання на дану кнопку
uint8_t in_X	Вхідне значення X, що використовується при розрахунках результату відповідних арифметичних операцій
uint8_t in_Y	Вхідне значення Y, що використовується при розрахунках результату відповідних арифметичних операцій
uint8_t in_mode	Змінна, що зберігає значення вибраного користувачем режиму роботи пристрою
uint16_t pressure_10_ms_ticks	Лічильник, який збільшується кожні 10 мс в відповідній функції, для ідентифікації тривалого натискання на відповідну кнопку (кнопку, що призначена для конфігурації режиму роботи пристрою, або ж для скидання до значень «за замовчуванням» введених користувачем вхідних значень)

uint8_t — тип даних, що являє собою беззнакове ціле восьмирозрядне число, uint16_t — тип даних, що являє собою без знакове ціле шістнадцятирозрядне число.

3.2. Перелік та короткий опис використаних в програмі керування пристроєм функцій.

— Опис функції init_systick.

Дана функція використовується для стартової ініціалізації роботи системного таймеру, задаючи початкове значення в регістрі перезавантаження, вмикаючи тактування від системної частоти, вмикаючи дозвіл переривань, а також вимикаючи системний таймер.

— Опис функції SysTick_Handler.

Обробник переривань системного таймеру. Використовується в алгоритмі ліквідації «дріб'язку контактів» тактових кнопок. В даному обробнику відбувається дозвіл замаскованих раніше, в обробнику переривань EXTI, переривань, що надходять від зовнішніх джерел, а також обнулення «прапорця», що сигналізує про активну роботу алгоритму «дебаунсу», і вимкнення після цього роботи системного таймеру.

— Опис функції systick_debounce_ms.

Дана функція приймає в якості аргументу значення часу в мілісекундах, протягом якого відбуватиметься «дебаунс» тактових кнопок, і заносить його в потрібному вигляді, після деяких арифметичних маніпуляцій згідно документації, до регістру перезавантаження системного таймеру, виставляючи при цьому в активне значення «прапорець», що сигналізує про активну роботу алгоритму «дебаунсу», і вмикаючи після цього роботу системного таймеру.

— Опис функції tim2_init.

Функція, що використовується для початкової ініціалізації роботи таймеру TIM2.

— Опис функції TIM2_IRQHandler.

Обробник переривань таймеру TIM2.

— Опис функції btn_pressure_check.

Перевірка тривалого натискання на тактову кнопку.

— Опис функції init_GPIO.

Конфігурація роботи потрібних для реалізації поставленого завдання портів вводу-виводу.

— Опис функції btn_irq_init.

Налаштування переривань від зовнішніх джерел.

— Опис функції EXTI15_10_IRQHandler.

Обробник переривань від зовнішніх джерел.

— Опис функції arithmetic_function.

Функція на мові асемблеру, в якій виконується розрахунок результатів арифметичних операцій додавання, віднімання, множення та ділення.

— Опис функції softdelay.

Реалізація часової затримки за допомогою системи циклів.

— Опис функції I2C1_init.

Ініціалізація модулю I2C1.

— Опис функції I2C_send.

Функція, що виконує надсилання даних по інтерфейсу I2C.

— Опис функції PCF8574AT_send.

Перетворення команд керування контролером LCD дисплею, переданих у функцію, у формат інструкцій, з яким працює PCF8574AT, а також відправка сформованих інструкцій до PCF8547AT за допомогою інтерфейсу I2C та функції I2C_send.

— Опис функції LCD1602_backlight_on.

Функція, яка вмикає підсвічування дисплею.

— Опис функції LCD1602_backlight_off.

Функція, яка вимикає підсвічування дисплею.

— Опис функції LCD1602_send_char.

Функція для виведення символу в ASCII-форматі, згідно таблиці передбачених знакогенератором та збережених в CGRAM пам'яті контролеру дисплея, символів, на дисплей, без вибору адреси знакомісця для виведення.

— Опис функції LCD1602_init.

Функція, що виконує ініціалізацію LCD дисплея.

— Опис функції LCD1602_set_cursor.

Запис бажаної адреси в вказівник DDRAM — встановлення позиції курсора.

— Опис функції LCD1602_send_char_position.

Функція для виведення символу в ASCII-форматі, згідно таблиці передбачених знакогенератором та збережених в CGRAM пам'яті контролеру дисплея, символів, на дисплей, з вибором адреси знакомісця для виведення.

— Опис функції LCD1602_position_rst.

Скидання значення вказівника адреси DDRAM в значення 0x0 — на початок діапазону адрес.

— Опис функції LCD1602_display_clear.

Очищення дисплею від виведених символів.

— Опис функції LCD1602_send_string.

Функція для для виведення рядка тексту на екран, з можливістю попереднього встановлення адреси DDRAM — позиції початку виведення.

— Опис функції LCD1602_send_integer.

Виведення числа в діапазоні від 0 до 99999 на дисплей, з можливістю попереднього встановлення адреси DDRAM — позиції початку виведення.

— Опис функції LCD1602_send_arith_results.

Виведення, згідно визначеного розробником формату виведення, значень введених користувачем вхідних даних X, Y, відомостей про режим функціонування пристрою, а також розрахованого значення результату виконання арифметичних операцій з вхідними даними, на екран.

— Опис функції arith_display.

Розрахунок за допомогою функції arithmetic_function значення результату виконання додавання/віднімання/множення/ділення вхідних даних, введених користувачем, і передача даного значення в функцію

LCD1602_send_arith_results для виведення на LCD дисплей результатів в необхідному форматі.

— Опис функції main.

Головна функція програми. В ній виконується ініціалізація всієї необхідної для роботи периферії, і перехід після цього в нескінченний цикл while(1).

3.3. Опис алгоритму та принципу роботи з дисплеєм.

3.3.1. Опис послідовного інтерфейсу передачі даних I2C.

В інтерфейсі використовуються дві лінії, це лінія тактування SCL, і лінія передачі даних SDA, які разом утворюють шину даних. Пристрої, підключені до шини діляться на ведучого і веденого. Ведучий ініціалізує процес передачі даних і видає тактові імпульси на лінію SCL, ведений приймає команди/дані, а також видає дані за запитом ведучого. Лінії SDA і SCL двонаправлені, пристрої, що підключаються до шини, повинні мати виводи, переналаштовуємі на вхід та вихід. Причому тип виходу має бути з відкритим коллектором/стоком, в зв'язку з чим, обидві лінії SDA і SCL через резистори підтягуються до позитивного полюса джерела живлення. На наступному рисунку приведена схема підключення інтерфейсу I2C.

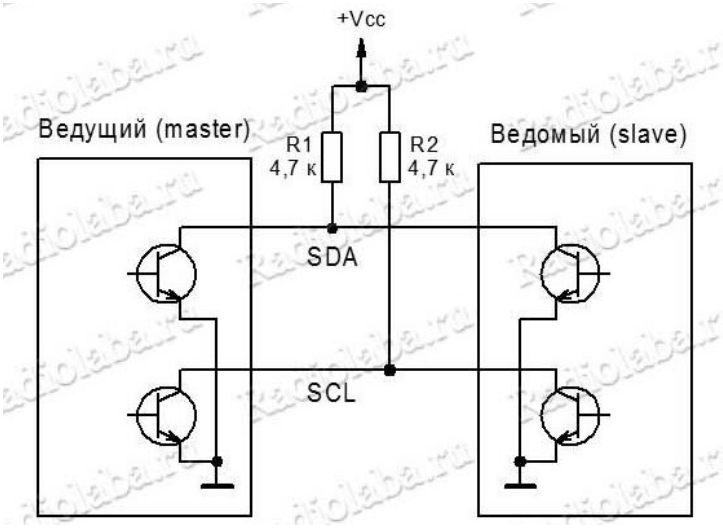

Рисунок 3.1 — Схема підключення пристроїв при використанні інтерфейсу I2C.

Інтерфейсом передбачена програмна адресація пристроїв підключених до шини, найбільш поширеною є довжина адреси в 7 біт, теоретично це дозволяє підключати на шину до 127 пристроїв, але частина адрес за специфікацією зарезервовані і не можуть використовуватися розробниками. Кожен пристрій має свою унікальну адресу, який закладений виробником, і який вказується в технічній документації. Адреса пристрою може бути фіксованою, або ж з можливістю апаратного налаштування — в цьому випадку пристрій має додаткові входи, і в залежності від рівня напруги на входах (високий або низький), можна отримати різні значення адреси. Зазвичай кількість таких входів варіюється від 1-го до 3-х, які задають значення певних бітів 7-бітної адреси. Апаратне налаштування адреси передбачене для можливості підключення декількох однотипних пристроїв на одну шину.

Кожен сеанс передачі даних починається так званим «старт-бітом». У початковому стані, коли шина вільна, обидві лінії SDA і SCL підтягнуті до високого логічного рівня. «Старт-біт» являє собою перемикання лінії SDA з високого логічного рівня на низький, в той час, коли на лінії SCL встановлений високий рівень.

Аналогічно, сеанс передачі даних завершується «стоп-бітом» — перемиканням лінії SDA з низького логічного рівня на високий, при високому рівні на лінії SCL. Дані умови генерує пристрій-ведучий (Master).

Виходячи з подій «старт» та «стоп», під час передачі даних лінія SDA може перемикатися тільки при низькому рівні на лінії SCL, тобто встановлення нових даних на лінії SDA можливе тільки після спаду логічного рівня на SCL. Протягом імпульсу тактування (високий рівень на SCL), стан лінії SDA не повинен змінюватися, так як в цей час виконується зчитування даних з лінії SDA.

Дані по інтерфейсу передаються побайтно, старшим бітом вперед, за кожним переданим байтом (8 біт) слідує біт підтвердження, пристрій (ведучий або ведений), що прийняв байт даних, встановлює низький рівень на лінії SDA

на наступному тактовому імпульсі SCL, тим самим підтверджуючи отримання байту. В цей час передаючий пристрій має опитувати лінію SDA, чекаючи відповідь про успішне отримання байта. На наступному рисунку представлена діаграма передачі даних по шині I2C.

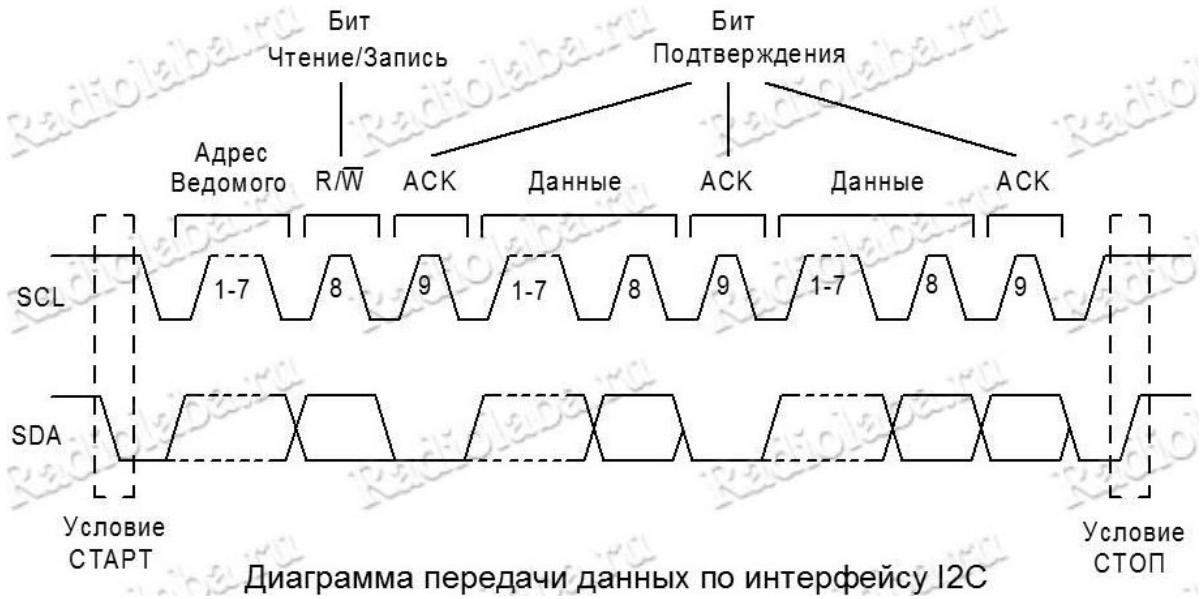

Рисунок 3.2 — Діаграма передачі даних по інтерфейсу I2C.

Спочатку передається байт з 7-бітовою адресою веденого, значення 8-го біта (R/!W) визначає напрямок передачі даних, нульове значення відповідає запису даних, тобто передача від ведучого до веденого. Якщо біт напрямку дорівнює 1, то виконується читання даних з веденого.

Ведений порівнює передану адресу зі своєю, і при збігу відгукується, встановлюючи низький рівень на лінії SDA (біт підтвердження). Ведучий, отримавши підтвердження, починає передавати байти даних, або ж приймає їх, залежно від вибраного напрямку передачі.

На наступному рисунку відображена діаграма передачі даних за допомогою інтерфейсу I2C при запису байту даних в визначений регістр від пристрою-ведучого до пристрою-веденого.

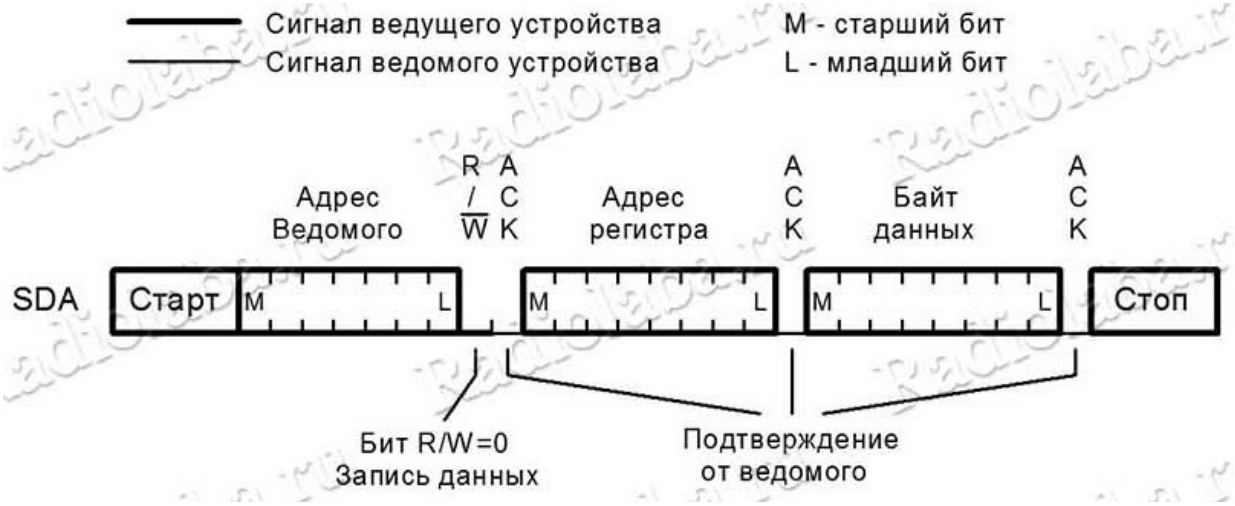

Рисунок 3.3 — Запис байту даних в заданий регістр.

3.3.2. Опис рідкокристалічного дисплею типу 1602 на основі контролера HD44780.

Даний дисплей містить 2 рядки по 16 символів. Напруга живлення може знаходитись в межах від 4.5 до 5.5В, струм споживання становить 1.2мА, без врахування підсвічування.

Дисплей має 16 виводів, їх призначення та опис наведені в наступній таблиці:

Таблиця 3.4 — Опис виводів LCD дисплея 1602 на базі контролера HD44780.

Номер виводу	Назва	Опис
1	Vss	Вивід живлення дисплея «-»
2	Vdd	Вивід живлення дисплея «+»
3	Vo	Вхід регулювання контрастності дисплея
4	RS	Вхід вибору типу інструкції: 1 — дані, 0 — команда
5	R/!W	Вхід напряму передачі даних: 1— запис даних в дисплей, 0 — читання даних з дисплею
6	E	Вхід тактування

7-14	DB0-DB7	Лінії вводу/виводу даних
15	A	Вивід живлення підсвічування «+»
16	K	Вивід живлення підсвічування «-»

Вхід VO призначений для регулювання контрастності екрану, яка залежить від величини напруги на вході, зазвичай для цих цілей встановлюється змінний резистор опором 10-20 кОм, підключений до лінії живлення. За допомогою входів RS, R/!W вибирається тип інструкцій та напрямок передачі даних. Вхід тактування E призначений для "замикання" (фіксації) станів входів і ліній введення/виведення, введення інструкцій в контролер дисплея — зчитування даних відбувається по спаду сигналу (по задньому фронту). Лінії DB7-DB0 представляють собою 8-бітний інтерфейс вводу/виводу даних, при цьому за один період тактового імпульсу передається 1 байт даних. Виводи A і K є анодом та катодом світлодіодів підсвічування, на платі дисплея встановлений струмообмежуючий резистор.

Команди для керування дисплеєм представлені на наступному рисунку:

№	Описание инструкции	Код инструкции										Время выполнения
		RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	Очистка дисплея с установкой курсора в начало первой строки	0	0	0	0	0	0	0	0	0	1	1,53 мс
2	Установка курсора в начало первой строки	0	0	0	0	0	0	0	0	1	–	1,53 мс
3	Установка направления вывода символов, разрешение сдвига экрана	0	0	0	0	0	0	0	1	I/D	SH	39 мкс
4	Управление режимом питания дисплея и отображением курсора	0	0	0	0	0	0	1	D	C	B	39 мкс
5	Команда сдвига курсора и экрана	0	0	0	0	0	1	S/C	R/L	–	–	39 мкс
6	Настройка интерфейса ввода/вывода данных, количества строк для вывода символов, размера шрифта	0	0	0	0	1	DL	N	F	–	–	39 мкс
7	Запись адреса CGRAM памяти в адресный указатель	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	39 мкс
8	Запись адреса DDRAM памяти в адресный указатель	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	39 мкс
9	Команда чтения флага занятости и адресного указателя	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	0 мкс
10	Запись данных во внутреннюю память дисплея	1	0	D7	D6	D5	D4	D3	D2	D1	D0	43 мкс
11	Чтение данных из внутренней памяти дисплея	1	1	D7	D6	D5	D4	D3	D2	D1	D0	43 мкс

Рисунок 3.4 – Таблиця, в якій представлені команди керування LCD дисплеєм типу 1602.

Призначення відповідних бітів пояснюється на Рисунку 3.2.

Бит	Значение	Описание
I/D	0	Вывод символов справа-налево, декремент адресного указателя DDRAM/CGRAM памяти
	1	Вывод символов слева-направо, инкремент адресного указателя DDRAM/CGRAM памяти
SH	0	Запрет сдвига экрана при выводе символов
	1	Разрешение сдвига экрана при выводе символов
D	0	Выключить экран дисплея, сегменты погашены, содержимое внутренней памяти сохраняется
	1	Включить экран дисплея, нормальный режим работы
C	0	Отключить отображение курсора
	1	Включить отображение курсора
B	0	Отключить функцию мигания курсора
	1	Включить функцию мигания курсора
S/C	0	Выбрать курсор для сдвига
	1	Выбрать экран (вместе с курсором) для сдвига
R/L	0	Сдвиг влево (только курсор или весь экран, зависит от бита S/C)
	1	Сдвиг вправо (только курсор или весь экран, зависит от бита S/C)
DL	0	4-битный интерфейс ввода/вывода данных
	1	8-битный интерфейс ввода/вывода данных
N	0	Использовать одну строку для вывода символов
	1	Задействовать 2 строки для вывода символов
F	0	Размер шрифта 5×8 пикселей
	1	Размер шрифта 5×11 пикселей
BF	0	Контроллер дисплея готов к обработке новой команды
	1	Контроллер дисплея занят выполнением внутренних операций

Рисунок 3.5 – Призначення бітів в інструкціях керування дисплеєм типу 1602.

Дисплей можна підключити до мікроконтролеру, використовуючи всі лінії вводу/виводу, тобто через 8-бітний інтерфейс, але для цього потрібна значна кількість виводів мікроконтролера. Можна скоротити кількість висновків, якщо переключитися на 4-бітний інтерфейс (використавши інструкцію №6 з таблиці на Рисунку 3.4.), в цьому режимі задіяними є тільки лінії вводу/виводу DB7-DB4, а лінії DB3-DB0 стають неактивними. У цьому випадку 1 байт даних передається за два тактових імпульси — спочатку старші 4 біти, потім молодші.

Внутрішня пам'ять LCD 1602 поділяється на 3 види: DDRAM, CGROM і CGRAM. Область DDRAM (Display Data RAM) пам'яті використовується для зберігання 8-бітного коду ASCII символів, що відображаються на екрані.

Адреси регістрів DDRAM пам'яті пов'язані з положенням символів на екрані, дана відповідність приведена на наступному рисунку:

	Адреса ячеек															
1-я строка	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2-я строка	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Рисунок 3.6 – Відповідність адрес пам’яті DDRAM положенню символів на екрані.

В кожному рядку екрану вміщається 16 символів, але це тільки видима область, загальний же обсяг DDRAM пам'яті становить 80 байт, тобто в кожен рядок можна записати по 40 символів, і тільки 16 з них будуть відображатися на екрані, інші символи при цьому залишаться в невидимій області.

За допомогою команди зсуву екрану (інструкція №5 в таблиці на Рисунку 3.4), можна переглянути всі інші символи. Регістри з адресами 0x00-0x27 складають перший рядок, комірки 0x40-0x67 — другий рядок. Якщо задати тільки один рядок для виведення (біт N = 0 в інструкції №6 на Рисунку 3.4), то довжина рядка збільшиться, і вміщатиме 80 символів.

Пам'ять CGROM (Character Generator ROM) являє собою знакогенератор і містить дані для відображення ASCII символів. У пам'яті закладені спецзнаки, цифри, латинський алфавіт. Кожен символ займає 5 байт в пам'яті, що відповідає розміру шрифту 5 × 8 пікселів. На наступному рисунку представлена таблиця символів відповідно до ASCII кодів:

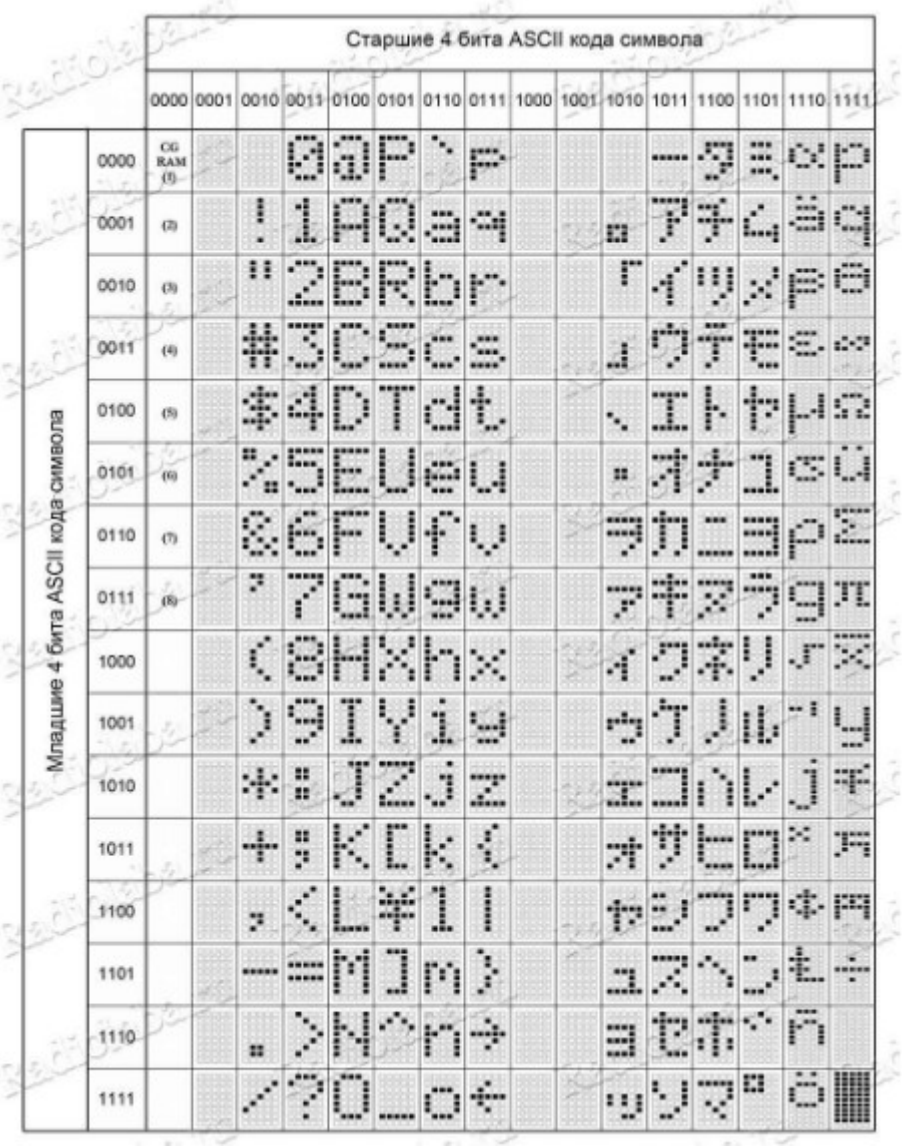

Рисунок 3.7 – Таблиця символів, доступних для виведення, і збережених в пам’яті CGROM.

Кожна інструкція виконується певний час, який вказано в наведеній вище таблиці (Рисунок 3.4), завершення внутрішньої операції можна дізнатися за допомогою читання прапора зайнятості BF (інструкція №9). Але зазвичай прапор не опитують, а просто витримують відповідну паузу після передачі інструкції. Взагалі, варто відмітити, що читання даних з дисплею застосовується рідко, і не розглядається в рамках даної роботи.

Для виведення символу необхідно записати адресу регістра DDRAM пам'яті в адресний покажчик (інструкція №8), тим самим вибравши положення

символу на екрані, потім записати в обраний регістр код ASCII символу (інструкція №10), виходячи з отриманого коду, контролер дисплея отримає дані з CGROM пам'яті для промальовування символу в заданому положенні на екрані. Після виведення символу, адресний покажчик автоматично інкрементується або декрементується, в залежності від того, яке значення було задано бітом напрямку I/D в відповідній інструкції. Таким чином, символи можна виводити послідовно, при цьому корегування адреси DDRAM пам'яті не потрібно. Наприклад, якщо перший рядок повністю заповниться символами, то відбудеться перехід на другий рядок, і навпаки.

Енергонезалежна пам'ять CGRAM (Character Generator RAM) призначена для створення унікальних символів за потреби розробника. Обсяг пам'яті невеликий, і дозволяє зберігати 8 довільних символів. Для створення одного символу розміром 5 × 8 пікселів, необхідно передати 8 байт даних в регістри пам'яті.

Перед створенням символу необхідно записати адресу регістра CGRAM пам'яті в адресний покажчик (інструкція №8), далі передати байти даних, які складуть вигляд символу (інструкція №10), адресний покажчик автоматично інкрементується або декрементується (в залежності від біта I/D), як і в разі DDRAM пам'яті. При створенні символу беруть участь тільки 5 молодших бітів байту даних. Символам присвоюються коди 0x00-0x07, відповідно до їх розташуванням в пам'яті CGRAM.

Варто відмітити, що даний режим роботи також не представляє інтересу в рамках даної роботи.

Ініціалізація дисплею LCD 1602.

Загалом, порядок ініціалізації наводиться в документації. В якості прикладу можна навести наступний алгоритм ініціалізації:

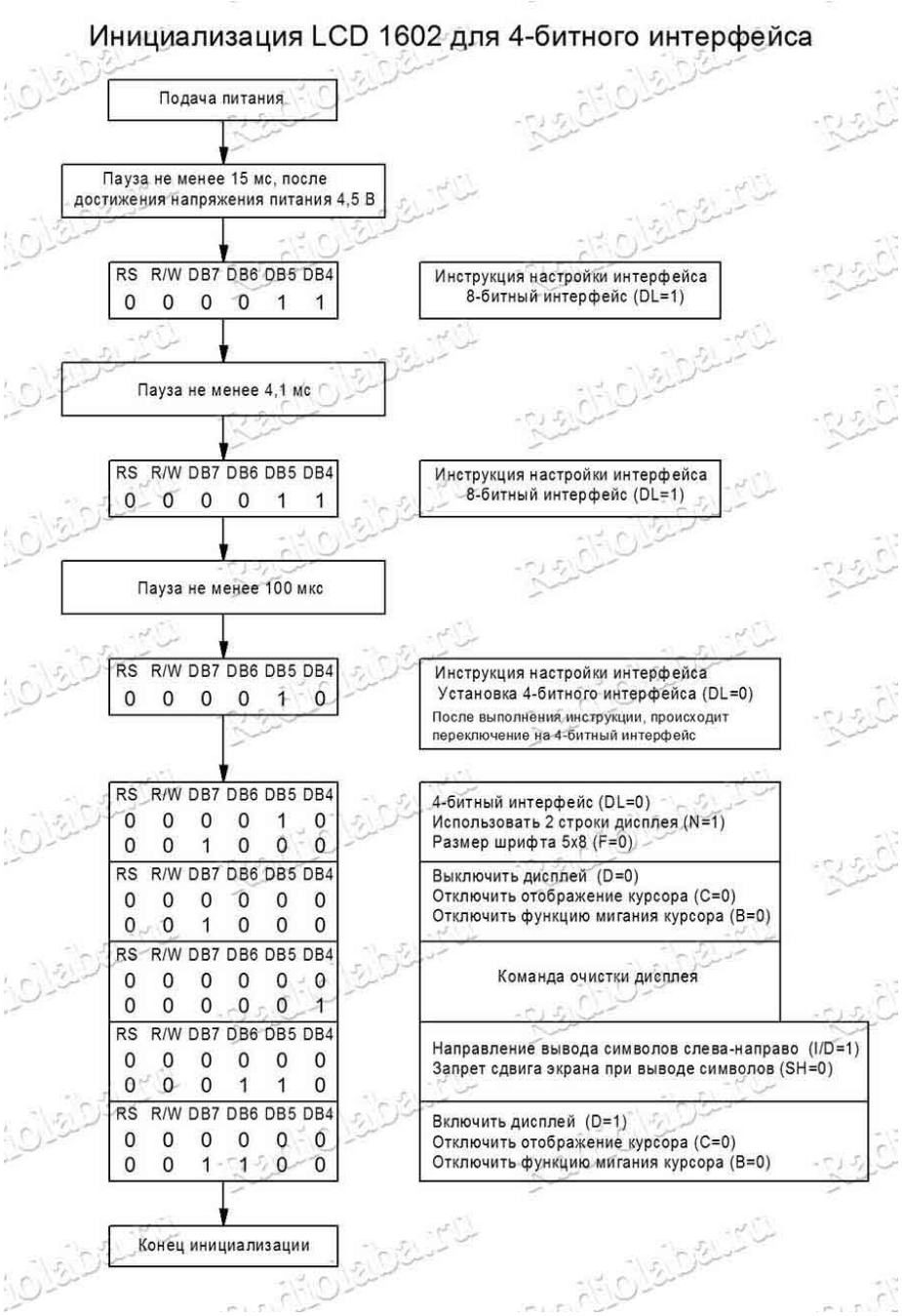

Рисунок 3.8 – Приклад порядку ініціалізації дисплею для роботи в режимі 4-бітного інтерфейсу.

Після ініціалізації згідно з алгоритмом, наведеним на Рисунку 3.8., дисплей налаштований на використання 2-х рядків, екран очищений від символів, курсор встановлений на початок першого рядка (адреса DDRAM пам'яті 0x00), відображення курсора відключено, обраний шрифт 5×8 пікселів,

напрямок виведення символів зліва-направо, зсув екрану при виведенні символів заборонений.

Якщо дозволити зсув екрану при виведенні символів (біт SH = 1), то з кожним новим символом екран буде зсуватись в обраному напрямку, тобто нові символи будуть з'являтися в заданому положенні на екрані, а інші зсуватимуться.

Додатково на екрані можна також включити відображення курсора (біт C = 1), який виглядає у вигляді лінії підкреслення, і може блимати в залежності від налаштувань. Положення курсору на екрані відповідає поточній адресі DDRAM пам'яті в адресному покажчику. Курсор можна переміщувати по екрану (інструкція №5), при цьому адресний покажчик буде інкрементуватись/ декрементуватись в залежності від напрямку зсуву.

Підключення дисплею типу 1602 через I2C інтерфейс.

Спільно з цим дисплеєм може бути використаний модуль-перехідник на основі мікросхеми PCF8574AT, яка призначена для розширення кількості ліній вводу/виводу. Дана мікросхема підключається по I2C інтерфейсу і має порт з 8 ліній вводу/виводу. Принцип функціонування полягає в тому, що при записі байта даних в мікросхему, лінії порту приймають рівні, які відповідають значенням бітів отриманого байта. Операція читання повертає байт даних, біти якого відповідають стану ліній порту. Таким чином, мікросхема дозволяє розширити кількість ліній вводу/виводу, використовуючи дві керуючі лінії — SDA та SLC.

На наступному рисунку наведена типова схема підключення дисплею та модуля розширювача портів.

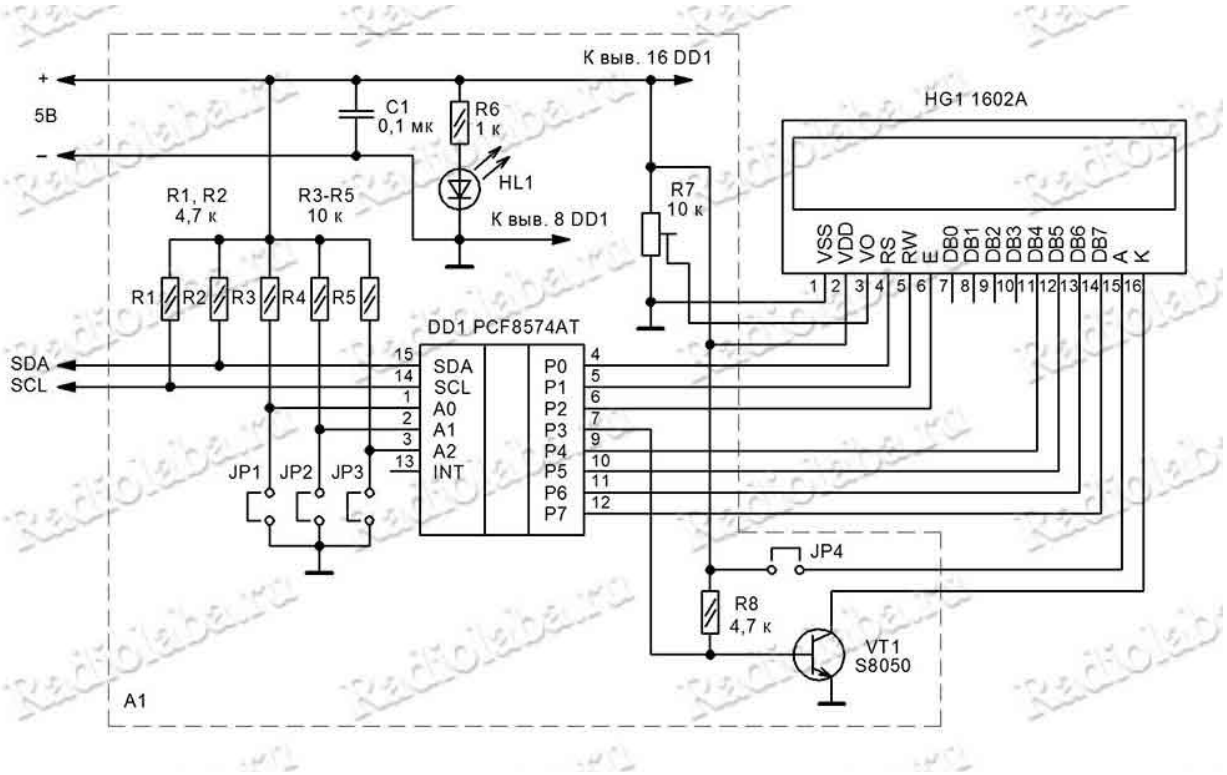

Рисунок 3.9 – Схема підключення дисплея та I2C-модуля.

Адресу мікросхеми PCF8574AT на шині I2C можна налаштовувати, при цьому старші 4 біти адреси є фіксованими, і рівні 0111, молодші 3 біти залежать від стану входів мікросхеми A2-A0. На модулі дані входи підтягнуті до високого рівня, відповідно адреса мікросхеми приймає значення 0111111.

Як видно зі схеми, наведеної вище, до мікросхеми підключена тільки частина ліній вводу/виводу дисплея DB7-DB4, це означає, що управління дисплеєм можливе тільки через 4-бітний інтерфейс. Для введення інструкції в дисплей потрібно 2 тактових імпульси на лінії E — тобто послідовність рівнів 1010 ("замикання" даних відбувається по спаду рівня), в результаті необхідно записати в мікросхему 4 байти для однієї інструкції.

На наступному рисунку наведено приклад запису інструкції в дисплей по інтерфейсу I2C.

Пример записи инструкции в LCD 1602 через I2C интерфейс (PCF8574AT)

Инструкция настройки интерфейса, количества активных строк, размера шрифта
4-битный интерфейс (DL=0), использовать 2 строки дисплея (N=1), размера шрифта 5x8 (F=0)

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	-	-

Для ввода инструкции используется 4-битный интерфейс LCD 1602

P7	P6	P5	P4	P3	P2	P1	P0	Линии ввода/вывода PCF8574AT
DB7	DB6	DB5	DB4	LED	E	R/W	RS	Линии LCD 1602 (LED-подсветка дисплея)
0	0	1	0	0	1	0	0	1-й байт: запись старшего полубайта команды, линия тактирования E=1
0	0	1	0	0	0	0	0	2-й байт: запись старшего полубайта команды, линия тактирования E=0
1	0	0	0	0	1	0	0	3-й байт: запись младшего полубайта команды, линия тактирования E=1
1	0	0	0	0	0	0	0	4-й байт: запись младшего полубайта команды, линия тактирования E=0

Рисунок 3.10 – Приклад передачі інструкції керування дисплеєм LCD 1602 через інтерфейс I2C з використанням розширювача ліній вводу-виводу PCF8574AT.

Як видно, спочатку передається старший напівбайт інструкції з бітом E = 1, потім він же, але з бітом E = 0, при цьому в дисплей передається перша половина інструкції. Далі таким же чином передається друга половина (молодший напівбайт).

Для управління підсвічуванням дисплея, на платі модуля розміщений транзистор, підключений до лінії P3 мікросхеми. Таким чином, 3-й біт (при нумерації, починаючи з 0) в байті даних, який передаватиметься за допомогою I2C, керує підсвічуванням.

3.4. Загальний опис алгоритму роботи програми.

Програма керування пристроєм починає своє виконання з функції main. Перш за все, відбувається конфігурація портів вводу-виводу — функція

init_GPIO. При цьому відбувається дозвіл тактування порту B модулю GPIO мікроконтролера. Далі піни 13, 14 та 15 даного порту налаштовуються на вхід, а також вмикається внутрішня їх підтяжка до напруги живлення.

Далі виконується функція btn_irq_init, яка виконує налаштування переривань від зовнішніх джерел — вмикає тактування контролеру конфігурації системи, вибирає піни PB13, PB14, PB15 в якості джерел зовнішніх переривань, дозволяє переривання від вищеназваних джерел, налаштовує виникнення переривань по приходу заднього фронту сигналу на відповідних пінах, які є джерелами зовнішніх переривань, а також, за допомогою відповідних функцій контролеру NVIC, дозволяє роботу обробника даних переривань (EXTI15_10_IRQn), очищує відповідні pending-біти, а також встановлює найвищий пріоритет для обробки даних переривань. Після цього відбувається глобальний дозвіл переривань.

Після цього відбувається конфігурація роботи системного таймера SysTick. Далі конфігурується робота таймера TIM2, за допомогою prescaler'а встановлюється режим роботи в 1000 тактів таймеру за секунду, а також переповнення таймера кожні 10 тактів (так званих «тіків» таймера). Також налаштовується дозвіл переривань по переповненню таймера, налаштовується режим рахування «вгору», відбувається дозвіл переривань від таймера TIM2, за допомогою відповідної функції контролера NVIC, а також глобально дозволяються переривання.

Після цього налаштовується робота модуля I2C1. Перш за все, дозволяється тактування шини APB1, яка зв'язана з даним модулем, далі виконується налаштування пінів PB8 та PB9 — виконується вибір альтернативної функції для даних пінів, а саме роботу у ролі ліній SCL та SCK модуля I2C1. Також дані піни налаштовуються на роботу в режимі альтернативної функції, а також режимі відкритого стоку (open drain).

Що до конфігурації модуля I2C1, він налаштовується наступним чином: тактування модулю від частоти шини PB1, що рівна 16 МГц, генерація

тактового сигналу на лінії SCL з частотою 100 кГц, режим роботи — стандартний. Також необхідним чином, згідно документації та вибраного режиму роботи — стандартний режим, розраховано було значення, яке заноситься в регістр TRISE, яке відповідає за налаштування «таймінгів» сигналу SCL, який генеруватиметься модулем I2C1. Після цього відбувається дозвіл роботи модулю I2C1.

Далі відбувається конфігурація роботи LCD дисплею. Дану процедуру виконує функція LCD1602_init. Процес конфігурації полягає в послідовній відправці по інтерфейсу I2C необхідних команд керування контролером дисплея. Для цього, загалом, застосовуються функції для роботи з I2C з відповідної самописної бібліотеки lcd1602.h, написаної в рамках даної роботи для виконання даного завдання. При цьому перші кілька команд керування надсилаються за два «такти», наступні ж команди — за 4 «такти», тобто, у вигляді 4 байтів, надісланих по лінії даних SDA інтерфейсу I2C.

Послідовність команд керування дисплеєм — цілком стандартна, і наводиться виробником в документації. Загалом, після виконання процедури ініціалізації, дисплей налаштований на роботу в наступному режимі: використання 4-бітного інтерфейсу, використання 2 рядків для виведення символів, шрифт 5x8, курсор вимкнений, його «блимання» вимкнено, дисплей очищено, виведення символів — зліва направо, зсув дисплея при виведенні символів вимкнено, покажчик адреси DDRAM встановлено в значення 0x0.

Значення-команди, які надсилаються по I2C для конфігурації роботи дисплея, наведені в вихідному коді. «Декодувати» їх можна, співставивши з таблицями, наведеними на Рисунках 3.1 та 3.2.

Після цього відбувається перехід в нескінченний цикл while(1). В ньому програма весь час очікує на встановлення в значення «1» «прапорця» LCD_show_ready, який свідчить про те, що значення вхідних даних було оновлено, відповідно, з'являється потреба оновити дані, що виводяться на дисплей.

За умови, коли даний «прапорець» не рівний «1», контролер виконує «порожнюю інструкцію» — NOP.

Розглянемо алгоритм роботи програми за умови, коли «прапорець» LCD_show_ready встановлюється в «1».

Перш за все, відбувається перехід в функцію arith_display, яка приймає в якості аргументів вхідні дані X, Y, а також значення, яке відповідає за режим роботи пристрою, які користувач вводить за допомогою тактових кнопок. Дана функція, в свою чергу, спершу декларує та ініціалізує нулем значення 32-розрядної беззнакової змінної, в якій зберігатиметься результат виконання арифметичних операцій, які виконає функція arithmetic_function. Функція arithmetic_function — написана з застосуванням підходу «embedded assembly», тобто вбудованого асемблеру. Дана функція розраховує результати арифметичних операцій додавання, або віднімання, або множення, або ділення, з введеними користувачем за допомогою тактових кнопок значеннями X та Y. Вибір того, в якому режимі працюватиме пристрій, також визначається вводом користувача за допомогою тактової кнопки відповідних значень.

Після виконання функції arithmetic_function, значення, яке вона повертає, присвоюється змінній result.

Після цього отримане розраховане значення result, а також введені користувачем значення вхідних даних X, Y, а також значення, яке визначає режим роботи пристрою, передаються в функцію LCD1602_send_arith_results, яка виконує їх виведення, згідно визначеного розробником формату виведення, на екран.

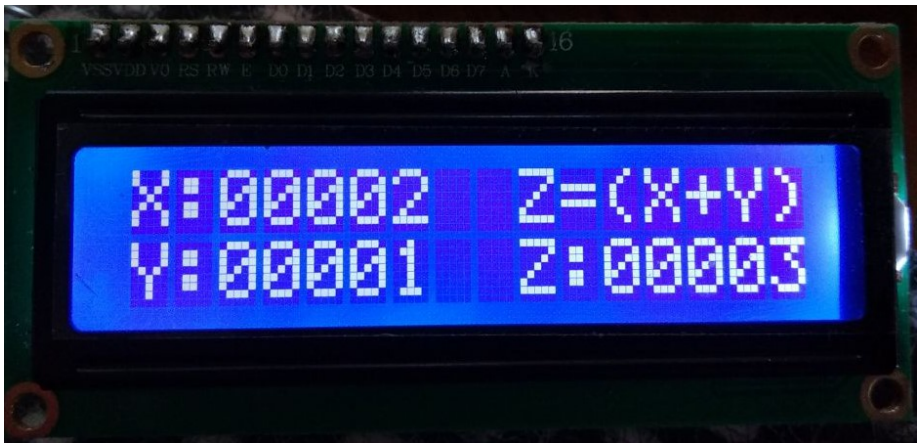

Рисунок 3.11 — Формат виведення вхідних даних X, Y, відомостей про режим роботи пристрою, а також результату виконання арифметичних розрахунків на дисплей.

В процесі роботи дана функція використовує функції LCD1602_send_string, а також LCD1602_send_integer, з бібліотеки lcd1602.h, які дозволяють вивести на дисплей рядок із текстом, або ж число, з можливістю вказати при цьому конкретну адресу DDRAM, щоб визначити, з якого з знакомісць почнеться виведення відповідних переданих в дані функції даних.

Після того, як функція LCD1602_send_arith_results виконає виведення потрібних даних на екран, відбудеться скидання «прапорця» LCD_show_ready, який свідчить, як згадувалось вище, про те, що вхідні дані, введені користувачем, були оновлені, і існує потреба в їх оновленні на дисплеї.

Також окремо варто зупинитись на роботі функції I2C_send, яка виконує відправку даних по I2C від пристрою-Master'а, яким виступає STM32F401RE, до пристрою-Slave'а, яким виступає PCF8574AT.

Формат передачі даних від пристрою-ведучого (Master) до пристрою-веденого (Slave) наведений в документації на мікроконтролер. Реалізується це, загалом, методом «поллінгу» та очікування встановлення потрібних значень бітів в регістрах статусу модуля IC21.

Перш за все, відбувається ініціалізація старт-біта. Далі відбувається очікування встановлення в «1» біту-«прапорця» SB в регістрі SR1модуля I2C1. Далі, за успішного виконання даної умови, відбувається надсилання адреси Slave'а по лінії даних, шляхом її запису в регістр DR. Після цього потрібно чекати, поки відбудеться встановлення в «1» біту ADDR в регістрі SR1, що свідчитиме про успішний прийом Slave`ом даної адреси та готовність до «спілкування» з пристроєм-ведучим. Далі, згідно документації, потрібно зчитати значення регістрів SR1 та SR2. Після цього відбувається надсилання необхідних байтів даних, шляхом їх почергового запису в регістр DR, і очікування, поки в данному регістрі не з'явиться вільне місце, про що сигналізує біт TXE в регістрі SR1. Коли наявні для відправки байти були відправлені, очікуємо встановлення в «1» значення біту BTF в регістрі SR1, що свідчить про закінчення передачі даних. Закінчується виконання даної функції формуванням стоп-біта.

Ключовою в роботі пристрою також є функція PCF8574AT_send, яка приймає на вхід 8-бітну команду керування дисплеєм, значення led_flag, яке визначає, буде увімкнена підсвітка дисплею, чи ні, значення rw_flag, що визначає, буде відбуватись запис в дисплей, чи зчитування з нього, значення rs_flag, яке визначає, інструкція буде надіслана дисплею, чи дані, а також значення cycles — кількості циклів, за які буде передана команда, а також значення delay_ms — затримки, яка буде здійснена після відправки команди.

Під кількістю «циклів» — значення cycles, мається на увазі необхідність «замикання» значення команди, яка була надіслана контролеру дисплея, заднім фронтом сигналу E. Тобто, потрібно надіслати команду спочатку з виставленим в високий рівень значенням сигналу «тактування» контролеру E, а тоді — надіслати, наприклад, її ж, але уже виставивши значення сигналу E в низький рівень. Це, в контексті розробленого алгоритму роботи — і є два «цикли».

Команди надсилання даних — ASCII-коду символів, наприклад, відбуваються за чотири «цикли» — спершу надсилаються, як уже згадувалось вище, чотири старші біти інструкції керування контролером дисплея (див. Рисунок 3.4), а також необхідні значення LED, E, R/!W та RS, згідно формату інструкцій PCF8547AT, з виставленим в високий рівень значенням біту E, далі надсилаються чотири молодші біти інструкції керування контролером дисплея, а після цього — це ж значення, але з виставленим в низький рівень значенням біту E, що призведе до «замикання» даного байту даних в пам’яті контролера дисплея.

Функція PCF8574AT_send, при цьому, перетворює інструкцію в форматі, наведеному в таблиці на Рисунку 3.4, яка передається в дану функцію у вигляді аргументу data, в формат, який підходить для передачі за допомогою PCF8574AT, переставляючи необхідним чином значення потрібних бітів, і формуючи «посилки» даних у вигляді, що підходить для надсилання по лінії даних I2C.

Наглядно відмінність в форматі команд керування контролером LCD дисплею, налаштованого на роботу в 4-бітному форматі команд керування, та форматі відповідних команд, які надсилаються з використанням PCF8574AT, відображена в наступній таблиці.

Таблиця 3.5 — Формати інструкцій керування контролером дисплея та «пакетів», що надсилаються до PCF8574AT.

Формат інструкції керування контролером LCD дисплея									
RS	R/!W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Формат «пакету», що надсилається по лінії даних SDA до PCF8547AT									
DB7/3	DB6/2	DB5/1	DB4/0	LED	E	R/!W	RS		

Як видно, «пакети», які надсилаються по лінії даних до PCF8547AT мікроконтролером, містять в старших чотирьох бітах чотири біти інструкції

керування контролером дисплея, а в чотирьох молодших бітах — біти, які визначають увімкнення/вимкнення підсвітки дисплея, біт E — сигнал тактування, заднім фронтом якого відбувається «замикання» надісланої інструкції в пам’яті контролера дисплея, а також біти R/!W та RS, які визначають, відповідно, зчитані будуть дані з контролеру дисплея, чи записані в нього, а також те, командою керування є надіслані дані, чи даними для відображення на дисплей.

Як видно також, інструкція керування контролером дисплея містить 8 біт DB7-DB0, які, зокрема, і визначають її призначення та функцію. А «пакет», що надсилається по I2C до PCF8574AT містить при цьому, в свою чергу, чотири біти з цих 8 біт, які «кодують» інструкцію. Відповідно, в 4-бітному режимі роботи дисплею, з використанням PCF8574AT, потрібно надіслати 2 «пакети» даних, щоб коректно сформувати повністю інструкцію керування контролером дисплею — спочатку в старших 4 бітах «пакету» будуть міститись старші 4 біти інструкції керування контролером дисплея (DB7-DB5), а після цього, в наступному «пакеті», його старші 4 біти міститимуть уже молодші 4 біти інструкції (DB3-DB0). Крім цього, потрібно також врахувати необхідність «замикання» надісланих даних в пам’яті контролера дисплея, надсилаючи «пакети» з послідовним чергуванням рівнів сигналу E («1» → «0»). Відповідно, в сумі на надсилання однієї 8-бітної інструкції керування контролером дисплея в 4-бітному режимі його роботи, потрібно надіслати 4 байти даних до PCF8574AT по лінії даних SDA — «пакет» з старшими 4 бітами (DB7-DB5) 8-бітної інструкції керування та сигналом E, встановленим в «1», цей же «пакет», але з сигналом E, встановленим в «0», «пакет» з молодшими 4 бітами (DB3-DB0) 8-бітної інструкції керування та сигналом E, встановленим в «1», цей же «пакет», але з сигналом E, встановленим в «0». Це — і є 4 «такти» (cycles), про які йшлося вище при поясненні принципу роботи функції PCF8574AT_send.

Після приведення інструкції до потрібного вигляду, сформовані дані передаються в функцію I2C_send, яка надсилає їх по I2C за вказану кількість

«циклів», необхідну для коректного «замикання» надісланих даних в пам’яті контролера дисплея. Після цього викликається функція softdelay, яка реалізує необхідну часову затримку після надсилання інструкції.

Висновки

Отже, в результаті виконання даної розрахунково-графічної роботи, було розроблено арифметичний пристрій, який виконує операції додавання, віднімання, множення та ділення над 8-розрядними числами. Основою пристрою є мікроконтролер STM32F401RE, встановлений на налагоджувальній платі STM32F401 Nucleo. Введення даних відбувається за допомогою трьох тактових кнопок з самоповерненням. Виведення результатів обчислень відбувається на рідкокристалічний дисплей типу 1602 на основі контролера HD44780.

Список використаних джерел

1. Документація на мікроконтролер STM32F401RE/[Електронний ресурс] — Режим доступу: https://www.st.com/content/ccc/resource/technical/document/reference_manual/5d/b1/ef/b2/a1/66/40/80/DM00096844.pdf/files/DM00096844.pdf/jcr:content/translations/en.DM00096844.pdf — Дата звернення 18.06.2019.
2. Документація на налагоджувальну плату STM32F401 Nucleo/[Електронний ресурс] — Режим доступу: https://www.st.com/content/ccc/resource/technical/document/user_manual/98/2e/fa/4b/e0/82/43/b7/DM00105823.pdf/files/DM00105823.pdf/jcr:content/translations/en.DM00105823.pdf — Дата звернення 18.06.2019.
3. Документація на LCD дисплей типу 1602 на основі контролера HD44780/[Електронний ресурс] — Режим доступу: https://www.openhacks.com/uploadsproductos/eone-1602a1.pdf — Дата звернення 18.06.2019.
4. Документація на розширювач портів вводу-виводу на основі мікросхеми PCF8574AT/[Електронний ресурс] — Режим доступу: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf — Дата звернення 18.06.2019.
5. Інструмент для налаштування тактування мікроконтролеру STM32F401RE — Clock configuration tool for STM32F40x/41x microcontrollers (AN3988)/ [Електронний ресурс] — Режим доступу: https://my.st.com/content/my_st_com/en/products/development-tools/software-development-tools/stm32-software-development-tools/stm32-configurators-and-code-generators/stsw-stm32091.html — Дата звернення 18.06.2019.

6. Підключення LCD1602 по I2C інтерфейсу/[Електронний ресурс] — Режим доступу: https://radiolaba.ru/microcotrollers/podklyuchenie-lcd-1602-po-i2c-interfeysu.html/ — Дата звернення 18.06.2019.
7. I2C інтерфейс/[Електронний ресурс] — Режим доступу: https://radiolaba.ru/microcotrollers/i2c-interfeys.html/ — Дата звернення 18.06.2019.

```
#include <stdint.h>
#include "stm32f4xx.h"
#include "lcd1602.h"

#define frequency  16000000UL   // 16 MHz high-speed internal (RC)

/*    GLOBAL VARIABLES START                        */

uint8_t LCD_show_ready = 0, debounce_ms_enable = 0, pressure_flag;
uint8_t in_X = 1, in_Y = 1, in_mode = 0;     // ADD by default
uint16_t pressure_10_ms_ticks = 0;

/*    GLOBAL VARIABLES END                          */

/********************************************** SYSTICK
**********************************************/

void init_systick(void)
{
      SysTick->CTRL = 0;        // disable SysTick
      SysTick->LOAD = frequency - 1;  // set the initial reload value
      SysTick->VAL = 0;         // reset the curent SysTick counter value

      SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk;
      SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;   // switch off

      // use proc.clock
      // (1 = processor clock, 0 = external clock (HCLK/8 = 2 MHz)),
      // enable interrupts (1 = enable)
}

void SysTick_Handler(void)
{
      if(debounce_ms_enable)
      {
            EXTI->IMR |= (EXTI_IMR_IM13 | EXTI_IMR_IM14 | EXTI_IMR_IM15);
            // enable disabled interrupts

            debounce_ms_enable = 0;

            SysTick->CTRL &= ~SysTick_CTRL_ENABLE_Msk;
      }
}

void systick_debounce_ms(uint32_t debounce_ms)
{
      // default value is 250 ms debounce
      uint32_t time_ms = (debounce_ms >= 1 && debounce_ms <= 1000) ?
debounce_ms : 250;

      SysTick->VAL = 0;         // clear current CNT value
      SysTick->LOAD = ((frequency / 1000) * time_ms) - 1;
      // (time * 10^-3 * frequency) - 1 = (time * 10^-3 * 16 000 000) - 1 =
      // (time * 16 000) - 1

      debounce_ms_enable = 1;

      SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;    // enable SysTick
}
```

```
/*****************************************************************************
*****************************/

/************************************************ TIM2
*************************************************/
void tim2_init(void)
{
	RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;    // TIM2 clock
	TIM2->PSC = 16000 - 1;    // 1000 ticks per second - 1ms - prescaler
	TIM2->ARR = 10-1;
	TIM2->DIER |= TIM_DIER_UIE;     // TIM2 DMA/interrupt enable register -
	TIM2->CR1 &= ~TIM_CR1_DIR;      // upcounting, DIR = 0

	NVIC_EnableIRQ(TIM2_IRQn);      // enable interrupts by TIM2
	__enable_irq();                 // global interrupts enable
}

void TIM2_IRQHandler(void)
{
	TIM2->SR &= ~TIM_SR_UIF;  // clear pending bit in the interrupt handler

	pressure_10_ms_ticks = ((GPIOB->IDR & GPIO_IDR_ID15) == pressure_flag) ?
(pressure_10_ms_ticks + 1) : 0;

	if(pressure_10_ms_ticks >= 200) // 200 x 10ms =
	                                //2000ms = 2s --> set default values
	{
		in_X = 1;
		in_Y = 1;
		in_mode = 0;

		LCD_show_ready = 1;

		pressure_10_ms_ticks = 0; // reset cnt

		TIM2->CR1 &= ~TIM_CR1_CEN;      // switch off TIM2
	}
}

void btn_pressure_check(void)
{
	// set pressure flag:
	pressure_flag = (GPIOB->IDR & GPIO_IDR_ID15);// PB15 == 1 -->
pressure_flag = 1

	TIM2->CR1 |= TIM_CR1_CEN;              // switch on TIM2
}

/*****************************************************************************
*****************************/

/************************************************ GPIO
*************************************************/

void init_GPIO(void)
{
	// OSPEEDR - 2 MHz by default after reset
```

```
	// RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;      // enable GPIOA
	// GPIOA->MODER |= GPIO_MODER_MODE5_0;       // PA5 for output  //
	GPIOA->MODER &= ~GPIO_MODER_MODE5_1;   // push-pull mode is by default


	RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;   // enable GPIOB

	GPIOB->MODER &= ~GPIO_MODER_MODE13;                // PB13 for input
	GPIOB->MODER &= ~GPIO_MODER_MODE14;                // PB14 for input
	GPIOB->MODER &= ~GPIO_MODER_MODE15;                // PB15 for input

	GPIOB->PUPDR |= GPIO_PUPDR_PUPD13_0;               // pull-up for PB13
- [01]
	GPIOB->PUPDR &= ~GPIO_PUPDR_PUPD13_1;
	GPIOB->PUPDR |= GPIO_PUPDR_PUPD14_0;               // pull-up for PB14
- [01]
	GPIOB->PUPDR &= ~GPIO_PUPDR_PUPD14_1;
	GPIOB->PUPDR |= GPIO_PUPDR_PUPD15_0;               // pull-up for PB15
- [01]
	GPIOB->PUPDR &= ~GPIO_PUPDR_PUPD15_1;
}

/*****************************************************************************
***************************/

/********************************************** EXTI
*********************************************/

void btn_irq_init(void)
{
	RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;  // SYSCFG (System configuration
controller) clock through APB2 bus enable

	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI13_PB;      // PB13 -> EXTI13
	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI14_PB;      // PB14 -> EXTI14
	SYSCFG->EXTICR[3] |= SYSCFG_EXTICR4_EXTI15_PB;      // PB15 -> EXTI15


	EXTI->IMR |= (EXTI_IMR_IM13 | EXTI_IMR_IM14 | EXTI_IMR_IM15);
	// interrupt request mask - IM11 and IM12 are not masked now

	EXTI->FTSR |= (EXTI_FTSR_TR13 | EXTI_FTSR_TR14 | EXTI_FTSR_TR15);
	// falling trigger

	NVIC_EnableIRQ(EXTI15_10_IRQn);
	NVIC_ClearPendingIRQ(EXTI15_10_IRQn);
	NVIC_SetPriority(EXTI15_10_IRQn, 0);   // highest priority

	__enable_irq();    // enable interrupts, PRIMASK reset
}

void EXTI15_10_IRQHandler(void)
{
	EXTI->IMR &= ~EXTI_IMR_IM13;    // disable EXTI13 interrupts
	EXTI->IMR &= ~EXTI_IMR_IM14;    // disable EXTI14 interrupts
	EXTI->IMR &= ~EXTI_IMR_IM15;    // disable EXTI15 interrupts

	systick_debounce_ms(500); // start debouncing
```

```
	if(EXTI->PR & EXTI_PR_PR13)     // PB13 - in_X
	{
		in_X++;

		EXTI->PR |= EXTI_PR_PR13;
		// clear pending flag by writing 1
		//(clear event flag after work)
	}
	else if(EXTI->PR & EXTI_PR_PR14)// PB14 - in_Y
	{
		in_Y++;

		EXTI->PR |= EXTI_PR_PR14; // clear pending flag by writing 1
	}
	else if(EXTI->PR & EXTI_PR_PR15)// PB15 - MODE/RST
	{
		btn_pressure_check();
		// save PB15 IDR value -->
		//start TIM2 10ms counting for 3 seconds

		in_mode = (in_mode <= 2) ? (in_mode + 1) : 0;

		EXTI->PR |= EXTI_PR_PR15; // clear pending flag by writing 1
	}

	LCD_show_ready = 1;
}

/*****************************************************************************
**************************/

/************************************************ ASM
************************************************/

__asm uint32_t arithmetic_function(uint8_t X, uint8_t Y, uint8_t mode)
{
	// embedded assembly function

	// X - RO, Y - R1, mode - R2

	// mode == 0 --> Z = (A + B)
	// mode == 1 --> Z = (A - B)
	// mode == 2 --> Z = (A * B)
	// mode == 3 --> Z = (A / B)

	CMP	R2, #0		// mode == 1?
	BEQ	add_label

	CMP	R2, #1		// mode == 2?
	BEQ	sub_label

	CMP	R2, #2		// mode == 3?
	BEQ	mul_label

	CMP	R2, #3		// mode == 4?
	BEQ	div_label

add_label
	ADD	R0, R0, R1	// R0 = R0 + R1 = X + Y
```

```
        BAL    end_label

sub_label
        SUB    R0, R0, R1   // R0 = R0 - R1 = X - Y
        BAL    end_label

mul_label
        MUL    R0, R0, R1   // R0 = R0 * R1 = X * Y
        BAL    end_label

div_label
        UDIV   R0, R0, R1   // R0 = R0 / R1 = X / Y
        BAL    end_label

end_label
        BX     LR
}

/*********************************************************************************
*********************************/

/************************************************ main loop
************************************************/

int main(void)
{
        init_GPIO();
        btn_irq_init();
        init_systick();
        tim2_init();

        I2C1_init();
        LCD1602_init();

        while(1)
        {
                if(LCD_show_ready)
                {
                        arith_display(in_X, in_Y, in_mode);

                        LCD_show_ready = 0;
                }
                else
                {
                        __NOP;
                }
        }
}
```

```
#include "lcd1602.h"
#include <stdint.h>
#include "stm32f4xx.h"

#define frequency  16000000UL          // 16 MHz high-speed internal (RC)

/*    I2C CODE DEFINES && VARIABLES && MACRO START */

#define I2C_MODE_TRANSMIT 0
#define I2C_ADDR(address, mode) (((address) << 1) | (mode))     //
[[addr[7:0]],[R/!W bit]]

#define PCF8574AT_ADDRESS 0x3F  // tested with Arduino

#define LCD_READ 1 // R / !W
#define LCD_WRITE 0// R / !W
#define LCD_INSTR 0// RS
#define LCD_DATA 1 // RS
#define ENA 1      // latching
#define N_ENA 0    // latching

/*    DISPLAY INSTRUCTIONS                          */

#define INTERFACE_8_BIT 0x30    // 0011 XXXX -> 001 [DATA 8/4] XXXX
#define INTERFACE_4_BIT_2_ROWS_FONT_5X8 0x28
// 0010 10XX -> 001 [DATA 8/4] [Number of lines 1/2] [Font size] XX
#define INTERFACE_4_BIT   0x20
#define DISPLAY_ON_CURSOR_ON_BLINK_DISABLE 0xE
// 0000 1110 -> 0000 1 [Display on/off] [Cursor] [Blink]
#define DISPLAY_ON_CURSOR_ON_BLINK_ENABLE 0xF
// 0000 1111 -> 0000 1 [Display on/off] [Cursor] [Blink]
#define DISPLAY_CLEAR 0x1              // 0000 0001
#define DIRECTION_L_TO_R_SHIFT_DISABLE 0x6
// 0000 0110 -> 0000 01 [DDRAM address Increment/Decrement] [Shift]
#define DDRAM_ADDRESS_0X0 0x80
#define DISPLAY_ON_CURSOR_OFF_BLINK_DISABLE 0xC
#define DISPLAY_OFF_CURSOR_OFF_BLINK_DISABLE 0x8

/*    I2C CODE DEFINES && VARIABLES && MACRO END   */

void softdelay(volatile unsigned long N)
{
      volatile unsigned long inner;

      while (N--)
      {
            for (inner = 100; inner > 0; inner--);
      }
}

// I2C initialization

void I2C1_init(void)
{
      RCC->APB1ENR |= RCC_APB1ENR_I2C1EN;

      RCC->AHB1ENR |= RCC_AHB1ENR_GPIOBEN;   // enable GPIOB

      GPIOB->AFR[1] |= (4 << (4 * 0));// PB8 - SCL - for AF4
```

```
	GPIOB->AFR[1] |= (4 << (4 * 1));// PB9 - SCL - for AF4

	GPIOB->MODER &= ~(GPIO_MODER_MODER9 |GPIO_MODER_MODER8);
	// AF open drain mode, external pull-ups
	GPIOB->MODER |= (GPIO_MODER_MODER9_1 |  GPIO_MODER_MODER8_1);
	GPIOB->OTYPER |=(GPIO_OTYPER_OT_9 |GPIO_OTYPER_OT_8);

	I2C1->CR2 &= ~I2C_CR2_FREQ;
	I2C1->CR2 |= 16;          // 16 MHz - PB1

	I2C1->CCR &= ~I2C_CCR_CCR;
	I2C1->CCR |= 80;          // 100 kHz SCL frequency

	I2C1->CCR &= ~I2C_CCR_FS;

	I2C1->TRISE = 17;         // (1000ns / Tpclk1) + 1

	I2C1->CR1 |= I2C_CR1_PE;  // peripheral enable
}

void I2C_send(uint8_t *pack, uint8_t pack_num)
{
	// Sends N = pack_num bytes of the data to the Slave.
	// Check "Figure 164. Transfer sequence diagram for master transmitter"
	//to see the events flow.

	I2C1->CR1 |= I2C_CR1_START;            // start bit
	while (!(I2C1->SR1 & I2C_SR1_SB));     // wait for SB (event 5)
	(void) I2C1->SR1;
	// clear SB flag - EV5: SB = 1,
	// cleared by reading SR1 register followed by writing DR
	                           // register with Address
	I2C1->DR = 0x7E;
	//I2C1->DR = 0x4E;
	while (!(I2C1->SR1 & I2C_SR1_ADDR));   // wait for ADDR
	(void) I2C1->SR1;
	// clear ADDR flag - EV6: ADDR = 1, cleared by reading
	// SR1 register followed by
	(void) I2C1->SR2;         // reading SR2

	I2C1->DR = *pack;
	while (!(I2C1->SR1 & I2C_SR1_TXE));    // wait for data register empty

	pack++;

	for(uint8_t num = 0; num < pack_num-1; num++)
	{
		I2C1->DR = *pack++;
		// send and go to the next cell of the input array
		while (!(I2C1->SR1 & I2C_SR1_TXE));
	}

	while (!(I2C1->SR1 & I2C_SR1_BTF));          // byte transfer finished

	I2C1->CR1 |= I2C_CR1_STOP;                   // stop bit
}

/*
void I2C_send_byte(uint8_t byte)
```

```
{
	I2C1->CR1 |= I2C_CR1_START;

	while (!(I2C1->SR1 & I2C_SR1_SB));
	(void) I2C1->SR1;

	I2C1->DR = 0x7E;

	while (!(I2C1->SR1 & I2C_SR1_ADDR));

	(void) I2C1->SR1;
	(void) I2C1->SR2;

	I2C1->DR = byte;
	while (!(I2C1->SR1 & I2C_SR1_TXE));

	while (!(I2C1->SR1 & I2C_SR1_BTF));

	I2C1->CR1 |= I2C_CR1_STOP;
}
*/

// implementation of latching inputting way
void PCF8574AT_send(uint8_t data, uint8_t led_flag, uint8_t rw_flag, uint8_t
rs_flag, uint8_t cycles, uint32_t delay_ms)
{
	// latching timings (delays) were not taken in account

	// cycles - 2 (instr -> latched_instr) or 4
	//(MSB_instr -> latched_MSB_instr -> LSB_instr -> latched_LSB_instr)
	// RS: 0 = instr, 1 = data
	// rw - R / !W

	uint8_t led, ena, rw, rs, data_msbits, data_lsbits;
	// data's 4 most significant bits, and 4 least significant bits

	led = led_flag << 3;	// LED: 0 - backlight OFF, 1 - backlight ON
	ena = ENA << 2;
	rw = rw_flag << 1;		// R/!W: 0 - write, 1 - read
	rs = rs_flag;			// RS: 0 - instruction, 1 - data

	data_msbits = data & 0xF0;	// 1111 1111 & F0 = 1111 0000
	data_lsbits = (data << 4) & 0xF0;	// 1111 0001 << 4 = 0001 0000

	uint8_t package[4];

	for(uint8_t cells = 0; cells < 4; cells++)
	{
		package[cells] = 0;
	}

	package[0] = data_msbits | led | ena | rw | rs;		// ENA = 1
	package[1] = package[0] ^ 4;	// ENA = 0 --> LATCHED!
	package[2] = data_lsbits | led | ena | rw | rs;		// ENA = 1
	package[3] = package[2] ^ 4;	// ENA = 0 --> LATCHED!

	I2C_send(package, cycles);	// send ready instruction

	softdelay(delay_ms);	// wait some time
```

```
	//st_delay_ms(delay_ms);  // required delay after instruction was sent
	//while(~(st_delay_ms_rdy));    // yep, just wait here
}

void LCD1602_backlight_on(void)
{
	PCF8574AT_send(0x0, 1, LCD_WRITE, LCD_INSTR, 2, 1); // 1 ms delay, 2
cycles
}

void LCD1602_backlight_off(void)
{
	PCF8574AT_send(0x0, 0, LCD_WRITE, LCD_INSTR, 2, 1); // 1 ms delay, 2
cycles
}

void LCD1602_send_char(uint8_t character)
{
	PCF8574AT_send(character, 1, LCD_WRITE, LCD_DATA, 4, 3);  // 4 cycles, 3
ms delay
}

void LCD1602_init()
{
	// RS | R/!W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0
	// - instruction format from datasheet
	// DB7 | DB6 | DB5 | DB4 | LED | E | R/!W | RS
	// - I2C (PCF8547AT instruction format)

	// softdelay(5);
	// some (40 ms, for the best way) delay before initialization start

	/*
	uint8_t instruction[] =
	{
		INTERFACE_8_BIT,
		INTERFACE_8_BIT,
		INTERFACE_4_BIT,
		INTERFACE_4_BIT_2_ROWS_FONT_5X8,
		DISPLAY_OFF_CURSOR_OFF_BLINK_DISABLE,
		DISPLAY_CLEAR,
		DIRECTION_L_TO_R_SHIFT_DISABLE,
		DISPLAY_ON_CURSOR_OFF_BLINK_DISABLE
	};
	*/

	uint8_t instruction[] = {0x33,0x32,0x28,0x0c,0x01,0x06,0x80};

	//  - or this:
	//
0x33,0x32,INTERFACE_4_BIT_2_ROWS_FONT_5X8,DISPLAY_ON_CURSOR_OFF_BLINK_DISABLE,
DISPLAY_CLEAR, DIRECTION_L_TO_R_SHIFT_DISABLE, DDRAM_ADDRESS_0X0
	// 0x30, 0x28, 0x28, 0xE, 0x1, 0x6 - my variant

	// uint8_t instr_count = sizeof(instruction) / sizeof(instruction[0]);
	// number of initialization instructions
	uint8_t instr_count = 7;

	uint8_t instr;
```

```
	for(instr = 0; instr < 3; instr++)
	{
		PCF8574AT_send(instruction[instr], 1, LCD_WRITE, LCD_INSTR, 2,
3);	// 3 ms delay, 2 cycles
	}
	for(instr = 3; instr < instr_count; instr++)
	{
		PCF8574AT_send(instruction[instr], 1, LCD_WRITE, LCD_INSTR, 4,
3);	// 3 ms delay, 4 cycles
	}
}

void LCD1602_set_cursor(uint8_t position)
{

	uint8_t data;

	data = (1 << 7) | position;

	PCF8574AT_send(data, 1, LCD_WRITE, LCD_INSTR, 4, 3);	// 4 cycles, 3
ms delay
}

void LCD1602_send_char_position(uint8_t character, uint8_t position)
{
	LCD1602_set_cursor(position);
	LCD1602_send_char(character);
}

void LCD1602_position_rst(void)
{
	PCF8574AT_send(0x2, 1, LCD_WRITE, LCD_INSTR, 4, 1); // 1 ms delay, 2
cycles
}

void LCD1602_display_clear(void)
{
	LCD1602_set_cursor(0x00);

	for(uint8_t position = 0x0; position <= 0x67; position++)
	{
		LCD1602_send_char(' ');
	}

	LCD1602_set_cursor(0x00);		// go to the start
}

void LCD1602_send_string(uint8_t *string, uint8_t position)
{
	LCD1602_set_cursor(position);

	while(*string)
	{
		LCD1602_send_char((uint8_t) *string);

		string++;
	}
}
```

```
void LCD1602_send_integer(uint16_t number, uint8_t position)
{
	// uint16_t tens_of_thousands, thousands, hundreds, tens, units;
	uint8_t output[5];

	output[0] = number / 10000;
	output[1] = (number - (output[0] * 10000)) / 1000;
	output[2] = (number - (output[0] * 10000) - (output[1] * 1000)) / 100;
	output[3] = (number - (output[0] * 10000) - (output[1] * 1000) -
(output[2] * 100)) / 10;
	output[4] = (number - (output[0] * 10000) - (output[1] * 1000) -
(output[2] * 100) - (output[3] * 10));

	LCD1602_set_cursor(position);

	for(uint8_t i = 0; i < 5; i++)
	{
		LCD1602_send_char(output[i] + 48);
	}
}


void LCD1602_send_arith_results(uint8_t X, uint8_t Y, uint8_t mode, uint32_t
result)
{

	// LCD1602_display_clear();

	LCD1602_send_string("X:", 0x00);
	LCD1602_send_integer(X, 0x02);

	LCD1602_send_string("Y:", 0x40);
	LCD1602_send_integer(Y, 0x42);

	LCD1602_send_string("Z:", 0x49);
	LCD1602_send_integer(result, 0x4B);

	switch(mode)
	{
		case 0:
			LCD1602_send_string("Z=(X+Y)", 0x09);
			break;
		case 1:
			LCD1602_send_string("Z=(X-Y)", 0x09);
			break;
		case 2:
			LCD1602_send_string("Z=(X*Y)", 0x09);
			break;
		case 3:
			LCD1602_send_string("Z=(X/Y)", 0x09);
			break;
		default:
			LCD1602_send_string("ERRMODE", 0x09);
	}
}
```

```
void arith_display(uint8_t X, uint8_t Y, uint8_t mode)
{
	uint32_t result = 0;

	result = arithmetic_function(X, Y, mode);

	LCD1602_send_arith_results(X, Y, mode, result);
}
```

```
#ifndef lcd1602_h
#define lcd1602_h

#include <stdint.h>

void softdelay(volatile unsigned long N);

void I2C1_init(void);
void I2C_send(uint8_t *pack, uint8_t pack_num);
// void I2C_send_byte(uint8_t byte);
void PCF8574AT_send(uint8_t data, uint8_t led_flag, uint8_t rw_flag, uint8_t
rs_flag, uint8_t cycles, uint32_t delay_ms);

void LCD1602_backlight_on(void);
void LCD1602_backlight_off(void);
void LCD1602_send_char(uint8_t character);
void LCD1602_init(void);
void LCD1602_set_cursor(uint8_t position);
void LCD1602_send_char_position(uint8_t character, uint8_t position);
void LCD1602_position_rst(void);
void LCD1602_display_clear(void);
void LCD1602_send_string(uint8_t *string, uint8_t position);
void LCD1602_send_integer(uint16_t number, uint8_t position);

void LCD1602_send_arith_results(uint8_t X, uint8_t Y, uint8_t mode, uint32_t
result);
void arith_display(uint8_t X, uint8_t Y, uint8_t mode);

#endif
```