Міністерство освіти і науки України
Національний університет водного господарства та природокористування
Кафедра автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій

04-03-456М

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт з навчальної дисципліни
«Мікропроцесорна техніка та програмування мікроконтролерів»
для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» денної та заочної форм навчання
Частина 2

Рекомендовано науково-методичною радою з якості ННІЕАВГ
Протокол №11 від 01.07.2025 р.

Рівне – 2025

Методичні вказівки до лабораторних робіт з навчальної дисципліни «Мікропроцесорна техніка та програмування мікроконтролерів» для здобувачів вищої освіти першого (бакалаврського) рівня за освітньо-професійною програмою «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» спеціальності 174 «Автоматизація, комп'ютерно-інтегровані технології та робототехніка» денної та заочної форм навчання. Частина 2 [Електронне видання] / Реут Д. Т. – Рівне : НУВГП, 2025. – 56 с.

Укладач: Реут Д. Т., к.т.н., доцент кафедри автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій.

Відповідальний за випуск: Древецький В. В., д.т.н., професор, завідувач кафедри автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій.

Керівник освітньої програми «Автоматизація, комп'ютерно-інтегровані технології та робототехніка»: Христюк А. О., к.т.н., доцент, доцент кафедри автоматизації, електротехнічних та комп’ютерно-інтегрованих технологій.

Зміст

Лабораторна робота 10. Реалізація багатозадачної системи на базі FreeRTOS

Мета роботи: налаштувати операційну систему реального часу FreeRTOS для запуску на мікроконтролері ATmega328P; реалізувати систему з витісняючою багатозадачністю за допомогою FreeRTOS.

Теоретичні відомості

У більшості випадків при проектуванні систем на 8-бітних мікроконтролерах не використовують операційної системи, що дає розробнику повний контроль над всіма ресурсами мікроконтролера. За необхідності виконання декількох задач, що вимагають гарантованого часу реакції на зовнішню подію, структура програми значно ускладнюється, збільшується складність відлагодження та супроводу. Використання операційної системи реального часу дозволяє розподілити код програми в декілька відносно простих для супроводу задач, керувати доступом до ресурсів мікроконтролера, встановлювати пріоритети задач, здійснювати псеводпаралельне виконання задач з однаковим пріоритетом.

В системі FreeRTOS кожен потік виконання називається *задачею* (*task*). Задачі реалізовуються на мові С у вигляді функцій типу *void*, які приймають єдиний аргумент типу *вказівник на void* та виконують нескінченний цикл, ніколи не завершуючись.

*void AtaskFunction(void *pvParameters);*

Одну й ту ж функцію можна використати для запуску кількох задач. Створення задачі здійснюється функцією

*portBASE_TYPE **xTaskCreate**(pdTASK_CODE pvTaskCode,*
*const signed portCHAR * const pcName,*
unsigned portSHORT usStackDepth,
*void *pvParameters,*
unsigned BaseType_t uxPriority,
*xTaskHandle *pxCreatedTask);*

Функція повертає значення pdTRUE у випадку успішного створення задачі. *pvTaskCode* – ім'я функції, що реалізує задачу; *pcName* – назва задачі, що використовуватиметься при відлагодженні; *usStackDepth* – розмір стеку задачі; *pvParameters* – вказівник на ділянку пам'яті, що містить параметри, які передаються в задачу при її виклику; *uxPriority* – пріоритет задачі; *pxCreatedTask* – handle створеної задачі, який використовується для маніпуляцій з задачею (призупинка, зміна пріоритету тощо).

Задача може знаходитись у наступних станах:

- Готова до виконання (READY). Задача готова прийняти на себе управління. Як тільки дійде її черга, планувальник (диспетчер) задач переведе задачу в режим RUN.

- Виконується (RUN). Планувальник передав управління задачі, процесор виконує програмний код задачі в даний момент. У цей момент задача споживає процесорний час і робить корисну роботу, заради якої вона була створена.
- Блокована/очікує (WAIT). Задача очікує подію, зовнішню (надходження зовнішнього сигналу переривання, отримання байту модулем USART) або внутрішню (закінчився час затримки). Планувальник не перемикається на неї, процесорний час не витрачається. Як тільки очікувана подія відбудеться, то FreeRTOS призначить цій задачі стан READY.
- Зупинена (SUSPEND). Задача не вивантажена з пам'яті, дані її збережені, але вона неактивна, на жодні події не реагує і сама з цього стану вийде. Вивести її з цього стану можна тільки API командою ОС з іншої задачі.

Знищити задачу і звільнити зайняту нею пам'ять можна функцією *vTaskDelete(pxCreatedTask)*.

У випадку витісняючої багатозадачності переремикання між задачами планувальник здійснює з частотою *configTICK_RATE_HZ* (макрос у файлі *FreeRTOSConfig.h*): в кінці кожного кванту часу (величина, обернена до *configTICK_RATE_HZ*) зберігає контекст поточної задачі, на початку наступного завантажує контекст задачі з найвищим пріоритетом серед готових до виконання та передає керування цій задачі. Задача може не чекати кінця виділеного їй кванту часу, а викликати диспетчер раніше викликом *taskYIELD()*. Щоб використовувати кооперативну багатозадачність замість витісняючої, потрібно у файлі конфігурації операційної системи *FreeRTOSConfig.h* встановити макрос

#define configUSE_PREEMPTION 0

При цьому перемикання між задачами диспетчер здіснюватиме лише після виклику *taskYIELD()*, не маючи змоги перервати виконання задачі за власною ініціативою, тому для забезпечення мінімального часу реакції системи потрібно якомога раніше викликати диспетчер. Оскільки виклики диспетчера відбуваються в наперед відомі моменти, то при перемиканні між задачами потрібно зберігати менше даних, тому використання кооперативної багатозадачності потребує менших затрат оперативної пам'яті, ніж витісняючої.

Затримка у виконанні коду задачі може бути здійснена функцією

*void **vTaskDelay**(portTickType xTicksToDelay);*

Величина затримки *xTicksToDelay* задається у кількості спрацювань системного таймера (tick), для переходу до задання затримки в мілісекундах використовується константа *portTICK_RATE_MS:*

vTaskDelay(200 / portTICK_RATE_MS); //затримка 200 мс

Оскільки затримка відраховується з моменту виклику функції, то для виклику задачі періодично кожні, наприклад, 200 мс потрібно віднімати час між викликом задачі та викликом функції затрики, щоб період не збільшився до величини (200 мс + час між викликом задачі та викликом функції затримки). Для таких випадків передбачена функція

*void **vTaskDelayUntil**(portTickType * pxPreviousWakeTime,*
portTickType xTimeIncrement);

Затримка рахується від моменту часу *pxPreviousWakeTime*, який повинен бути ініціалізований у задачі до виклику згаданої функції.

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
portTickType xLastWakeTime;

 /* Ініціалізація значення xLastWakeTime */
 xLastWakeTime = xTaskGetTickCount();

 while(1)
 {
 ...//дії, які задача повинна виконувати кожні 200 мс
 vTaskDelayUntil( &xLastWakeTime, (200/portTICK_RATE_MS));
/*змінна xLastWakeTime автоматично збільшується при виклику функції
vTaskDelayUntil*/
 }
}
```

Для реалізації передачі даних між задачами першим методом, що спадає на думку, є використання глобальних змінних, до яких має доступ кожна задача. Оскільки виконання операцій зі змінними довжиною більше 1 байта 8-бітний мікроконтролером затрачається більше одного машинного циклу, то гарантувати правильність значення змінної без додаткових засобів неможливо: нехай перша задача змінює значення змінної з 255 (0b011111111) на 256 (0b100000000) записом 1 у старший байт і 0 у молодший байт; у момент між записом двох байтів закінчується квант часу, виділений першій задачі й диспетчер передає керування другій задачі, яка зчитує значення змінної як 1023 (0b111111111, старший байт 00000001 містить нове значення, молодший байт 11111111 - попереднє значення, яке перша задача не встигла змінити). Це явище є порушенням атомарності операції (атомарною є операція, яка виконується без

переривань). Щоб виключити таку ситуацію, використовуються черги повідомлень (задача надсилає дані в чергу і далі вже операційна система відповідає за те, щоб інша задача зчитала з черги правильне значення), семафори (семафор встановлюється перед записом та скидається після запису; друга задача зчитує змінну лише якщо семафор скинутий), критичні секції (ділянки програми, в яких заборонено виклик диспетчера і відповідно перемикання на іншу задачу).

Нехай перша задача записує в послідовний порт рядок "PV=158", а друга – "SP=120". При однаковому пріоритеті задач можлива ситуація, коли планувальник перерве виконання першої задачі в момент запису і відновить виконання другої, в результаті приймаюча сторона може отримати "PV=15SP=1280". Для арбітражу доступу кількох задач до одного апартного ресурсу служать м'ютекси. Задача перед доступом до апартного ресурсу перевіряє, чи вільний м'ютекс, що відповідає цьому ресурсу. Якщо так, задача захоплює м'ютекс, здійснює потрібні дії з ресурсом та звільняє м'ютекс. Якщо м'ютекс вже захоплений іншою задачею, операційна система переведе задачу в стан очікування (WAIT) до моменту, коли м'ютекс знову стане доступним, або до закінчення таймауту. М'ютекс реалізовується на базі семафору, тому для його створення використовується функція

xSemaphoreHandle SemaphoreCreateMutex(void),

яка повертає handle, що використовується для операцій з м'ютексом. Захопити та звільнити м'ютекс можна функціями

xSemaphoreTake*(SemaphoreHandle_t xMutex,*
TickType_t xTicksToWait),

xSemaphoreGive*(SemaphoreHandle_t Mutex)*

відповідно, де *xMutex* – handle м'ютекса. Якщо впродовж *xTicksToWait* задача не змогла захопити м'ютекс, виконання задачі буде продовжено, а функція поверне pdFALSE.

Ще одним засобом виключення одночасного доступу декількох задач до ресурсу є задача-gatekeeper. Лише вона має виключний доступ до ресурсу й всі інші задачі повинні обмінюватись даними з цією задачею для роботи з ресурсом.

Черга повідомлень створюється функцією

xQueueHandle ***xQueueCreate****(unsigned BaseType_t uxQueueLength, unsigned BaseType_t uxItemSize);*

де *uxQueueLength* - максимальна кількість елементів у черзі, *uxItemSize* - розмір одного елемента. Функція повертає handle черги.

Для запису елементу в чергу служить функція

BaseType_t ***xQueueSend****(xQueueHandle xQueue, const void * pvItemToQueue,*

portTickType xTicksToWait);

де *xQueue* – handle черги, *pvItemToQueue* – вказівник на елемент, який необхідно надіслати в чергу, *xTicksToWait* – максимальний час очікування вільного місця в черзі. Для надсилання даних поза чергою, в початок черги, служить функція *xQueueSendToFront* з аналогічним попередній прототипом.

Зчитати елемент з черги можна функцією

BaseType_t ***xQueueReceive****(xQueueHandle xQueue, const void * pvBuffer, portTickType xTicksToWait)*

Функція чекає час *xTicksToWait,* доки з'явиться елемент в черзі *xQueue*, та записує його за адресою, на яку вказує *pvBuffer*.

Якщо необхідно, щоб функція чекала необмежено довго потрібної події, потрібно вказати час очікування *xTicksToWait* рівним portMAX_DELAY.

Структура FreeRTOS

FreeRTOS розповсюджується у вигляді архіву з 2 каталогів: власне FreeRTOS та FreeRTOS-Plus. Остання містить реалізацію підтримки файлових систем та функцій роботи з IP-мережами. Каталог FreeRTOS містить каталоги Source (вихідний код операційної системи), Demo (каталог з демонстраційними проектами), License (містить файл з ліцензійною угодою). Демо-проекти розміщені в каталогах *ім'я_платформи_компілятор*, наприклад PIC18_MPLAB, CORTEX_STM32F103_GCC_Rowley, та каталог Common зі спільним для всіх демо-проектів кодом.

У каталозі Source та його підкаталозі include розташовані архітектуронезалежні файли, а в каталозі Source/portable – специфічні для кожного компілятора та платформи файли port.c та portmacro.h, в яких реалізовано механізм перемикання контексту, та каталог MemMang, що містить файли з реалізацією операцій з купою (heap).

Щоб використати FreeRTOS у проекті, потрібно додати в нього файли tasks.c, list.c (містять реалізацію планувальника задач та API-функції роботи з задачами), *queue.c* (реалізація черг повідомлень), один з файлів *heap_x.c* з теки *MemMang* (реалізація функцій роботи з пам'яттю), *port.c* та файл *FreeRTOSConfig.h*, де здійснюється конфігурування операційної системи. У випадку використання семафорів або м'ютексів додатково потрібно підключити *semphr.h,* співпрограм – *croutine.h.*

Для взаємодії задач у FreeRTOS використовуються семафори, м’ютекси, черги повідомлень, можуть використовуватись глобальні змінні за умови забезпечення атомарності операцій доступу, наприклад, з використанням критичних секцій.

Функції роботи з семафорами та м’ютексами описані в **semphr.h**

Створення двійкового семафора

SemaphoreHandle_t ***SemaphoreCreateBinary****(void);*

Створення лічильного семафора

SemaphoreHandle_t ***xSemaphoreCreateCounting****(UBaseType_t uxMaxCount, UBaseType_t uxInitialCount);*

Захоплення (очікування) семафора

xSemaphoreTake*(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait);*

Таймаут xTicksToWait=*portMAX_DELAY* призведе до необмеженого очікування

Видача (звільнення) семафора

xSemaphoreGive*(SemaphoreHandle_t xSemaphore);*

Приклад використання двійкового семафора для сигналізації про настання події іншій задачі наведено нижче.

```
SemaphoreHandle_t xSemaphore = NULL;
void vATask( void * pvParameters )
{
  xSemaphore = xSemaphoreCreateBinary();
...
  xSemaphoreGive( xSemaphore ); //сигналізуємо іншій задачі
...
}

void vAnotherTask( void * pvParameters )
{
   if( xSemaphore != NULL )
   {
      if( xSemaphoreTake( xSemaphore, ( TickType_t ) 10 ) == pdTRUE )
      {   //виконуємо дії при надходженні сигналу
      }
      else
      {  //виходимо по таймауту (10) й виконуємо дії, передбачені при відсутності сигналу від іншої задачі
      }
   }
}
```

Для створення м’ютекса використовується функція *SemaphoreHandle_t* ***SemaphoreCreateMutex****(void),* а для операцій з ним – функції роботи з семафорами:

```
SemaphoreHandle_t xLcdMutex;
void vATask( void * pvParameters )
```

```
{  xLcdMutex = xSemaphoreCreateMutex();
  if( xLcdMutex != NULL )
  {
    //м’ютекс створено і може бути використано
  }
}
void vBTask( void * pvParameters )
{
   if( xLcdMutex != NULL )
   {
//намагаємось захопити м’ютекс
     if( xSemaphoreTake( xLcdMutex, ( TickType_t ) 10 ) == pdTRUE )
     {  //виконуємо дії в режимі ексклюзивного доступу до ресурсу
        LCD_print(“Hello, world”);
        xSemaphoreGive( xLcdMutex ); //звільняємо м’ютекс
     }
     else
     {  // не вдалось отримати доступ до ресурсу
     }
   }
}
```

За можливості всі налаштування модулів мікроконтролера, потрібні для роботи задачі, варто виконувати на початку цієї задачі (до початку нескінченого циклу), а не виносити за її межі. Це полегшує повторне використання коду цієї задачі в інших проектах.

План роботи

1. Налаштувати дистрибутив FreeRTOS для роботи на лабораторній платі.

2. Реалізувати псевдопаралельне виконання 2 задач блимання світлодіодами.

3. Реалізувати задачу індикації.

4. Реалізувати задачі опитування АЦП і сигналізації.

5. Реалізувати ПІД-імпульсний регулятор з індикацією та сигналізацією з використанням розроблених задач.

Порядок виконання роботи

1. Запустити STM32CubeIDE та створити новий проект для мікроконтролера STM32F072C8T6.

2. Відкрити категорію Middleware та обрати FreeRTOS. Налаштувати відповідно до рис. 1.

3. Перейти на вкладку Tasks та створити дві задачі з нормальним пріоритетом: blinkTask1 і blinkTask2. Переналаштувати HAL на використання базового таймера TIM6 замість SysTick, для чого зайти в категорію SystemCore, обрати SYS, далі Timebase Source -> TIM6. Згенерувати програмний код ініціалізації.

4. Знайти в отриманому коді реалізацію функцій, що реалізовують вказані вище задачі:

```
/* USER CODE BEGIN Header_StartTask02 */
/**
* @brief Function implementing the blinkTask1 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask02 */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END StartTask02 */
}
```

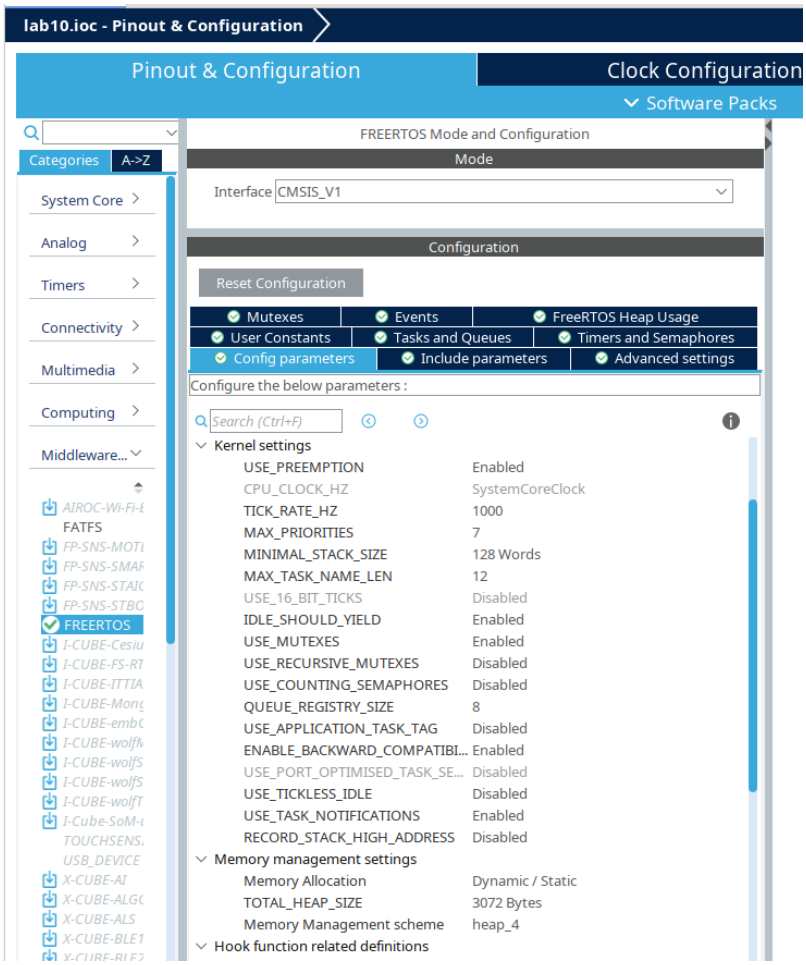


Рис. 1. Налаштування FreeRTOS в проекті

```
/* USER CODE BEGIN Header_StartTask03 */
/**
* @brief Function implementing the blinkTask2 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask03 */
void StartTask03(void const * argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  for(;;)
  {
```

```
    osDelay(1);
  }
  /* USER CODE END StartTask03 */
}
```

5. Модифікувати функції так, щоб кожна задача блимала своїм власним світлодіодом: перша - підключеним до контакту PA8, друга - підключеним до контакту PA5. Для формування затримок використовувати функції osDelay() замість HAL_Delay(). Прошити програму в мікроконтролер та налагодити її.

6. Замість блимання світлодіодом на PA5 додати іншу задачу, яка опитуватиме АЦП, зчитуватиме результат і якщо напруга вища заданого порогового значення - вмикатиме блимання світлодіода PA5. Для передачі між задачами стану сигналізація ввімкнена/вимкнена використати засоби FreeRTOS для взаємодії між задачами. Прошити програму в мікроконтролер та налагодити її.

7. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програми з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Які вимоги ставляться до функції, що реалізовує задачу?
2. У яких станах може знаходитись задача?
3. Як планувальник задач визначає, яку задачу запустити на виконання?
4. Як задається вид багатозадачності (витісняюча чи кооперативна) у FreeRTOS?
5. Де задається частота виклику планувальника задач у FreeRTOS?
6. Як реалізувати виконання функції PID() кожні 0,05 с?
7. Які засоби FreeRTOS використовуються, щоб виключити можливість порушення атомарності операції?
8. Що таке задача-gatekeeper?
9. Який файл містить специфічні для конкретної платформи механізми перемикання контексту?
10. Як створити двійковий семафор?
11. Для чого призначений м'ютекс і який заголовний файл необхідно підключити для його використання у програмі?
12. Як захопити м'ютекс?
13. Чи є атомарною операція зчитування змінної типу float в 8-бітних мікроконтролерах архітектури AVR? Чому?

Лабораторна робота 11. Функції цифрового та аналогового вводу-виводу в Arduino

Мета роботи: ознайомитися з апаратною та програмною частинами середовища Arduino. Вивчити основи роботи з аналоговими і дискретними сигналами в Arduino IDE.

Теоретичні відомості

Arduino – це відкрита електронна платформа для швидкого прототипування електроніки, що базується на простому у використанні апаратному та програмному забезпеченні.

Апаратна частина включає мікропрцесорну (як правило, мікроконтролерну) плату, а програмна – середовище Arduino IDE. Платформа Arduino є відкритою, тому існує ряд аналогів, Arduino-сумісних мікроконтролерних плат різного виконання.

Arduino Uno

Arduino Uno – це мікропроцесорна плата на основі мікроконтролера ATmega328P. До його складу входить все необхідне для зручної роботи з мікроконтролером: 14 контактів цифрових входів/виходів (з них 6 можуть використовуватися в якості ШІМ-виходів), 6 контактів аналогових входів, кварцовий резонатор на 16 МГц, роз'єм USB, роз'єм живлення, роз'єм для внутрішньосхемного програмування (AVR ISP-6) і кнопка скидання. Для початку роботи з пристроєм достатньо просто подати живлення від AC/DC-адаптера або батарейки 9 В, або підключити його до комп'ютера за допомогою USB-кабелю.

Входи і виходи

Кожен з 14 цифрових виводів (рис. 2) може працювати в якості входу або виходу. Рівень напруги на виводах обмежений напругою живлення 5В. Максимально допустимий струм, який може віддавати або споживати один вивід, становить 40 мА, проте рекомендується не перевищувати 20-25 мА. Усі виводи з'єднані з внутрішніми підтягуючими резисторами (за замовчуванням відключеними) номіналом 20-50 кОм. Крім цього, деякі виводи Arduino Uno можуть виконувати додаткові функції:

Послідовний інтерфейс: виводи 0 (RX) і 1 (TX). Використовуються для отримання (RX) і передачі (TX) даних по послідовному інтерфейсу. Ці виводи з'єднані з відповідними виводами ATmega8U2, що виконує роль перетворювача USB/UART (в платах-аналогах може використовуватись CH340).

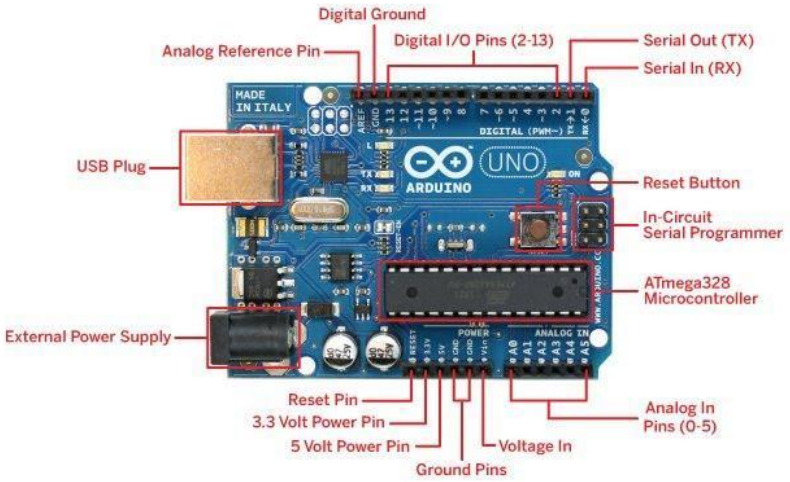


Рис. 2. Призначення виводів плати Arduino Uno

Таблиця 1

Основні характеристики Arduino Uno

Мікроконтролер	ATmega328P
Робоча напруга	5V
Вхідна напруга (рекомендована)	7-12V
Дискретні (цифрові) входи/виходи	14 (6 із них можуть використовуватися як ШІМ-виходи)
Аналогові входи	6
Максимальний струм одного вивода	40 mA
Максимальний струм вивода 3,3В	50 mA
Флеш-пам'ять	32 KB з яких 0.5 KB використовуються завантажувачем
SRAM	2 KB
EEPROM	1 KB
Тактова частота	16 MHz

Зовнішні переривання: виводи 2 і 3. Можуть служити джерелами переривань, що виникають при фронті, спаді або при низькому рівні сигналу на цих виводах.

ШІМ: виводи 3, 5, 6, 9, 10 і 11. За допомогою функції analogWrite() можуть виводити ШІМ-сигнал (можна використовувати для реалізації псевдоаналогового виходу).

Інтерфейс SPI: виводи 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Виводи можуть задіюються для зв'язку по інтерфейсу SPI та ICSP-програмування.

Світлодіод: 13. Вбудований світлодіод, приєднаний до виводу 13.

В Arduino Uno є 6 аналогових входів (A0 - A5), на кожен з яких можна подати напругу в межах 0 – 5В та отримати аналогово-цифрове перетворення вхідного сигналу у вигляді 10-бітного числа (1024 різних значення). Верхню межу вхідної напруги можна змінити, використовуючи вивід AREF і функцію analogReference().

Arduino Mega 2560

Arduino Mega 2560 – це пристрій на основі мікроконтролера ATmega2560 (рис. 3).

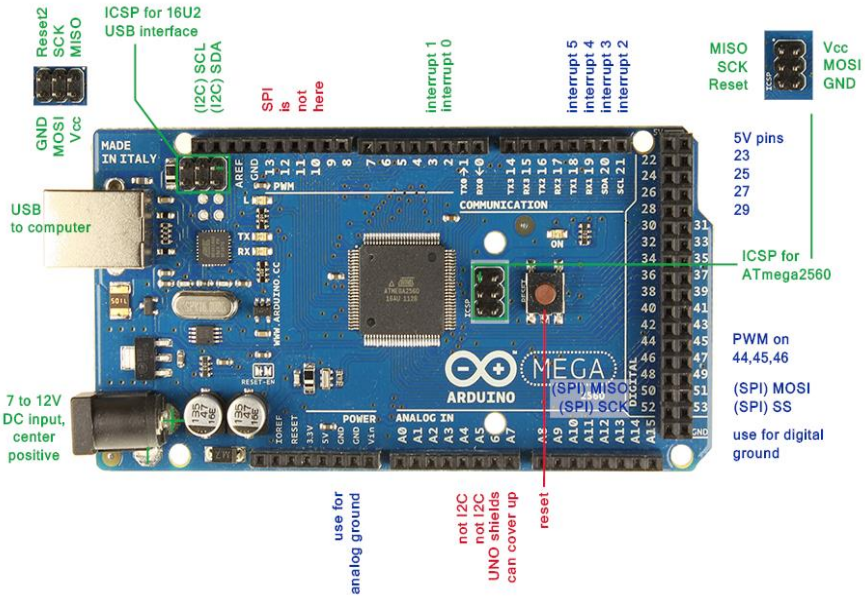


Рис. 3. Зовнішній вигляд та призначення виводів плати Arduino Mega 2560

Основні характеристики плати Arduino Mega 2560 наведені в табл. 2.

Таблиця 2

Основні характеристики Arduino Mega 2560

Мікроконтролер	ATmega2560
Робоча напруга	5V
Вхідна напруга (рекомендована)	7-12V
Дискретні (цифрові) входи/виходи	54 (15 із них можуть використовуватися як ШІМ-виходи)

Аналогові входи	16
Максимальний струм одного вивода	40 mA
Максимальний струм вивода 3,3В	50 mA
Флеш-пам'ять	256 КВ з яких 8 КВ використовуються завантажувачем
SRAM	8 КВ
EEPROM	4 КВ
Тактова частота	16 MHz

Програмування Arduino

Програмування мікроконтролерних плат Arduino здійснюється у вільно розповсюджуваному середовищі Arduino IDE.

Мікроконтролери в платах Arduino випускаються з прошитим завантажувачем (bootloader), що дозволяє завантажувати в мікроконтролер нові програми без необхідності використання зовнішнього програматора.

Також мікроконтролер можна прошити і через роз'єм для внутрішньосхемного програмування ICSP (In-Circuit Serial Programming), не звертаючи уваги на завантажувач.

Мова програмування Arduino – це трохи видозмінена мова C/C++ із дещо розширеним набором можливостей.

Кожна програма для пристроїв Arduino містить дві основні функції:

1) void setup() – функція, що виконується один раз, після кожної подачі живлення або скидання плати Arduino;

2) void loop() – виконується постійно в циклі після виконання функції void setup().

Вони неявно для користувача розміщуються в функції main() і врешті-решт компілюються компілятором мови C++.

Перед завантаженням програми в мікроконтролер слід спочатку вибрати тип плати Arduino та порт, до якого вона підключена.

Деякі функції для роботи з входам/виходами Arduino

pinMode(pin, value) – налаштування виводу з номером pin як дискретного входу (value=INPUT) або дискретного виходу (value=OUTPUT). Наприклад:

pinMode(13, OUTPUT);

digitalRead(pin) – функція зчитує із заданого входу значення HIGH або LOW.

digitalWrite(pin, value) – подає на цифровий вхід/вихід значення (value) HIGH або LOW;

analogRead(pin) – зчитує величину напруги з аналогового входу і видає результат у вигляді цілого числа від 0 до 1023.

Аналогово-цифровий перетворювач (АЦП, англ. Analog-todigital conventer, ADC) - пристрій, що перетворює вхідний аналоговий сигнал в цифровий код (цифровий сигнал).

У мікроконтролері ATmega328P є 10-розрядний АЦП. Вхідна напруга конвертується в 10-розрядне двійкове значення. Мінімальне значення відповідає 0, а максимальне - опорній напрузі V_{ref}. Отриманий результат перетворення повертається функцією analogRead(pin) як число в діапазоні 0…1023. Результат перетворення можна розрахувати за формулою

$$ADC = \frac{V_{in} \cdot 1023}{V_{ref}}$$

Внаслідок перехідних процесів швидкість АЦП є обмеженою й для отримання 10-бітних результатів перетворення максимальна частота – 15 тисяч вибірок за секунду.

План роботи

1. Ознайомитися з будовою та призначенням елементів плати Arduino Uno.

2. Ознайомитися з принципами програмування в середовищі Arduino IDE.

3. Навчитися підключати давачі з дискретними та аналоговими вихідними сигналами до Arduino та обробляти дані цих давачів.

Порядок виконання роботи

1. Ознайомитися з теоретичними відомостями.

2. Завантажити середовище Arduino IDE.

3. Ознайомитись зі схемою підключення механічної кнопки (рис. 4). Зібрати її на макетній платі. Використати підтягуючий резистор на 7,5 кОм.

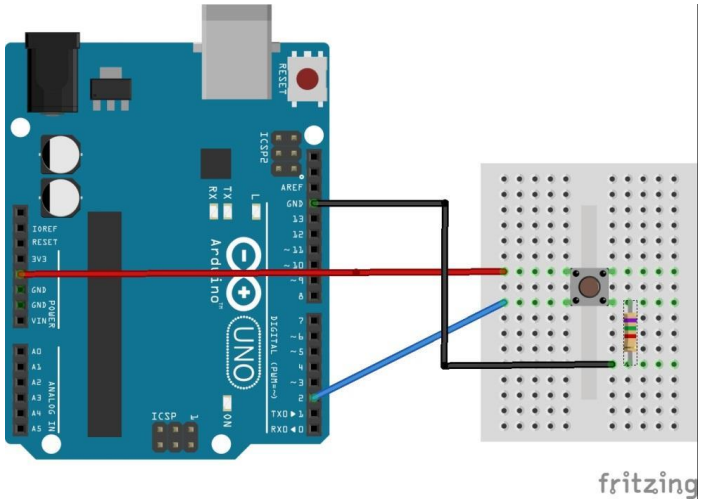


Рис. 4. Підключення кнопки до Arduino Uno

У середовищі Adruino IDE написати програму, що запалюватиме світлодіод при кожному натисканні кнопки. Скомпілювати програму та переконатися у відсутності помилок. Підключити мікроконтролерну плату до USB-порта комп'ютера.

У середовищі Arduino IDE вибрати тип плати та порт, до якого вона підключена. Завантажити створену програму в мікроконтролер. Переконатися в коректності роботи програми.

4. Підключити до плати Arduino ультразвуковий давач вимірювання відстані HC-SR04. Вивід Echo давача підключити до виводу 12 плати Arduino; вивід Trig – до виводу 11; вивід Vcc – до виводу 5 В; вивід GND – до одного з виводів GND на платі Arduino.

Завантажити в МК наступну програму:

```
#define echoPin 12 // Echo Pin
#define trigPin 11 // Trigger Pin
#define LEDPin 13 // Onboard LED
char rec;
int maximumRange = 300; // максимальна відстань
int minimumRange = 0; // мінімальна відстань
float duration, distance; // тривалість, потрібна для розрахунку відстані

void setup() {
 Serial.begin (9600);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
```

```
 pinMode(LEDPin, OUTPUT); // Use LED indicator (if required)
}

void loop() {
/* The following trigPin/echoPin cycle is used to determine the distance of the nearest object by bouncing soundwaves off of it. */
 digitalWrite(trigPin, LOW);
 delayMicroseconds(2);

 digitalWrite(trigPin, HIGH);
 delayMicroseconds(10);

 digitalWrite(trigPin, LOW); //надіслали імпульс
 duration = pulseIn(echoPin, HIGH); //чекаємо відлуння й записуємо час у змінну
 distance = duration/58.2;
 if (distance >= maximumRange || distance <= minimumRange){ //якщо значення помилкове (поза діапазоном вимірювання)
  Serial.println("error");
  Serial.print("\n\r");
  digitalWrite(LEDPin, HIGH);
  delay(50);
 }
 else {
  Serial.println(distance);
  Serial.print("\n\r");
  digitalWrite(LEDPin, LOW);
  delay(250);
 }
 delay(150);

byte l[255];
 byte count=0;
 int i;
 if (Serial.available()>0){
 while(Serial.available()>0){
  digitalWrite(LEDPin, HIGH);
 l[count] = Serial.read();
 count++;}
 for(i=0;i<count;i++){
```

```
    Serial.write(l[i]);};
    Serial.println();
    Serial.print("\n\r");
    delay(500);}
}
```

Відкрити монітор порту. Піднести руку навпроти давача на відстані 20 – 50 см. Спостерігати за виміряним значенням відстані, що виводиться на екран.

5. Підключити підстроювальний або змінний резистор за схемою подільника напруги (потенціометра) на рис. 5.

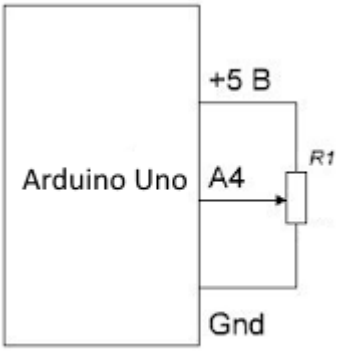


Рис. 5. Схема підключення змінного резистора до Arduino Uno

Відповідно до напруги на ковзному контакті (0...5 В) передавати у послідовний інтерфейс значення змінної norm, яка зберігає число від 0 до 100, яке прямопропорційно залежить від напруги.

```
    int sensorPin = A4;   // аналоговий вхід
    int sensorValue = 0;  // значення сигналу від давача
    int norm=0; // нормоване значення сигналу з давача
void setup() {
    Serial.begin(9600);  // налаштовуємо послідовний порт для зв'язку з ПК
}
void loop() {
    sensorValue = analogRead(sensorPin);      // зчитуємо значення з давача
    norm=map(sensorValue, 0, 1023, 0, 100); // нормуємо сигнал в межах від 0 до 100
    Serial.println(norm);  // виводимо нормоване значення сигналу в
```

порт

delay(500); // затримка 0,5 с

}

6. Замість змінного резистора підключити до аналогового входу подільник, утворений постійним резистором й давачем освітленості (фоторезистором) згідно рис. 6. Змінюючи освітленість фоторезистора, спостерігати за значенням змінної norm.

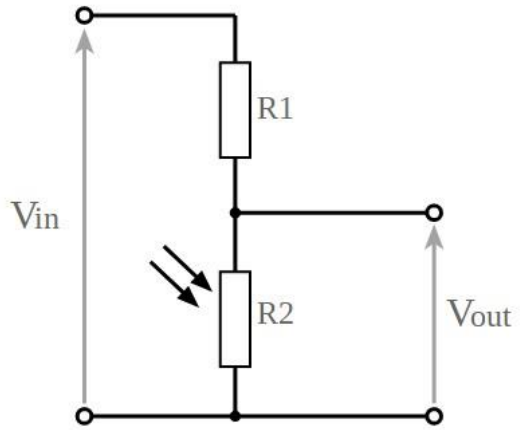


Рис. 6. Схема підключення фоторезистора до Arduino Uno

7. Оформити звіт про виконання лабораторної роботи. Звіт повинен містити: назву та мету лабораторної роботи; тексти програм з коментарями; висновок про виконання роботи.

Контрольні запитання

1. Яка функція налаштовує режим роботи піна на вхід?
2. Яка функція дозволяє зчитати стан цифрового входу?
3. Яка функція дозволяє змінити стан цифрового виходу
4. Скільки таймерів має мікроконтролер ATmega328P?
5. Яка розрядність модуля АЦП мікроконтролера ATmega328P?
6. Яка кількість каналів модуля АЦП в Arduino Uno?
7. Якою функцією можна виміряти тривалість імпульсу?
8. Якою функцією можна отримати результат АЦП?

Лабораторна робота №12. Використання рідкокристалічного дисплея та сенсорного екрану для побудови інтерфейсу користувача

Мета роботи: Навчитися підключати рідкокристалічний дисплей (РКД) та сенсорний екран до мікроконтролера. Оволодіти навичками програмування мікроконтролера для виведення інформації на РКД та обробки сенсорних подій.

Теоретичні відомості

Принцип роботи РКД

Рідкі кристали - це речовини, які мають властивості як рідини, так і твердого тіла. Вони можуть змінювати свою орієнтацію під впливом електричного поля.

РКД складається з двох скляних пластин, між якими знаходиться шар рідких кристалів. За пластинами розташовані поляризаційні фільтри. При подачі електричного поля на певні ділянки дисплея, рідкі кристали змінюють свою орієнтацію, що впливає на проходження світла через поляризаційні фільтри. Таким чином, можна керувати яскравістю окремих пікселів на дисплеї.

Типи РКД

Символьні РКД: Призначені для відображення текстової інформації. Мають обмежену кількість символів, які можуть бути відображені одночасно (наприклад, 16x2, 20x4). Зазвичай використовуються для відображення простих повідомлень та даних.

Графічні РКД: Дозволяють відображати графічні зображення та більш складний текст. Мають матрицю пікселів, кожен з яких може бути керований окремо. Використовуються для створення більш складних інтерфейсів користувача.

TFT (з тонкоплівковими транзисторами) РКД: Використовують тонкоплівкові транзистори для керування кожним пікселем. Використовуються в графічних дисплеях. Забезпечують високу якість зображення, високу контрастність та швидкий час відгуку. Використовуються в сучасних моніторах, телевізорах та мобільних пристроях.

Інтерфейси підключення РКД до мікроконтролера:

1. Паралельний інтерфейс. Використовує кілька ліній даних для передачі інформації (найчастіше 8 або 4). Забезпечує теоретично вищу швидкість передачі даних, але вимагає великої кількості виводів мікроконтролера.

2. Послідовний інтерфейс. Передає дані по одній лінії, біт за бітом. Використовує менше виводів мікроконтролера, але має меншу швидкість

передачі даних. Нерідко для підключення диплеїв використовуються такі інтерфейси як I2C (Inter-Integrated Circuit) та SPI (Serial Peripheral Interface).

Команди та функції для управління РКД

Взаємодія з дисплеєм для мікропроцесора зводиться до обміну даними з контролером дисплея, а саме запису в його пам'ять значень для кожного пікселя. Залежно від типу дисплея (монохромний, кольоровий) 1 піксель може потребувати 1, 8, 16, 32 біта для збереження даних про інтенсивність кольору(ів). Як правило, для взаємодії з контролером дисплея використовують бібліотеки. Вони найчастіше містять реалізацію таких функцій як:

- Ініціалізація дисплея.
- Очищення екрана.
- Встановлення курсора.
- Виведення символів та тексту.
- Виведення графічних елементів.

Чим більше біт потрібно для кодування значення кожного пікселя і чим більша роздільна здатність дисплея, тим більше часу на одній і тій самій швидкості інтерфейсу потрібно, щоб повністю оновити вміст дисплея. Це може ускладнити виведення анімації та відео на дисплеях з відносно повільними інтерфейсами.

Принцип роботи резистивних сенсорних екранів:

Складаються з двох шарів, розділених прошарком. При натисканні на екран шари контактують, що приводить до зміни опору. Мікроконтролер вимірює зміну опору та визначає координати натискання.

Принцип роботи ємнісних сенсорних екранів

Використовують електричну ємність, яка змінюється при дотику людини. Вимірюється зміна ємності та визначаються координати дотику. Забезпечують більш високу точність та чутливість, ніж резистивні екрани, та витримують більшу кількість натискань.

Інтерфейси підключення сенсорних екранів до мікроконтролера:

1. Аналоговий інтерфейс. Використовується для резистивних сенсорних екранів. Мікроконтролер вимірює аналоговий сигнал для визначення координат.
2. I2C та SPI: Використовуються для ємнісних сенсорних екранів. Забезпечують цифрову передачу даних.

Для коректного визначення координат натискання необхідне калібрування сенсорного екрану. Воно виконується шляхом натискання на характерні точки на екрані. Мікроконтролер запам'ятовує координати натискань та використовує їх для корекції.

При програмуванні мікропроцесора для роботи з екраном, як

правило, виникають такі задачі як визначення натискання на екран, визначення координат натискання, визначення руху пальця по екрану, визначення жестів (наприклад, збільшення/зменшення).

Приклад програми для створення інтерфейсу користувача з використанням сенсорного екрану та дисплея ILI9341 на Arduino

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <TouchScreen.h>

// Піни для дисплея ILI9341
#define TFT_CS 10
#define TFT_DC 9
#define TFT_RST 8

// Піни для сенсорного екрану XPT2046
#define XP 6
#define YP 7
#define XM A2
#define YM A3
// Калібрувальні значення для сенсорного екрану
#define TS_MINX 150
#define TS_MAXX 920
#define TS_MINY 120
#define TS_MAXY 940
// Розмір екрана
#define SCREEN_WIDTH 240
#define SCREEN_HEIGHT 320

// Створення об'єктів дисплея та сенсорного екрану
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);
TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);

// Координати кнопок
#define BUTTON_X 50
#define BUTTON_Y 50
#define BUTTON_WIDTH 140
#define BUTTON_HEIGHT 50

void setup() {
```

```
  Serial.begin(9600);
  tft.begin();
  tft.fillScreen(ILI9341_BLACK);
  drawButtons();
 }

 void loop() {
  TSPoint p = ts.getPoint();

  if (p.z > 10) { // Перевірка натискання
   int x = map(p.x, TS_MINX, TS_MAXX, 0, SCREEN_WIDTH);
   int y = map(p.y, TS_MINY, TS_MAXY, 0, SCREEN_HEIGHT);

   if (x > BUTTON_X && x < BUTTON_X + BUTTON_WIDTH && y > BUTTON_Y && y < BUTTON_Y + BUTTON_HEIGHT) {
    tft.fillScreen(ILI9341_BLUE);
    tft.setCursor(20, 20);
    tft.setTextColor(ILI9341_WHITE);
    tft.setTextSize(2);
    tft.println("Button 1 Pressed!");
    delay(1000);
    tft.fillScreen(ILI9341_BLACK);
    drawButtons();
   }
  }
 }

 void drawButtons() {
  tft.drawRect(BUTTON_X, BUTTON_Y, BUTTON_WIDTH, BUTTON_HEIGHT, ILI9341_WHITE);
  tft.setCursor(BUTTON_X + 20, BUTTON_Y + 20);
  tft.setTextColor(ILI9341_WHITE);
  tft.setTextSize(2);
  tft.println("Button 1");
 }
```

План роботи

1. Запрограмувати мікроконтролерну плату для виведення даних на дисплей з контролером ILI9341.

2. Реалізувати зчитування координат точки натиснення на сенсорний екран з контролером XPT2046.

3. Створити програму, яка забезпечує графічний інтерфейс користувача.

Порядок виконання роботи

1. Завантажити бібліотеку Adafruit_ILI9341 та встановити у середовище Arduino.

2. Відкрити приклад graphicstest й завантажити його в мікроконтролер WiFi-модуля ESP.

3. Підключити дисплей до мікроконтролерної плати відповідно до розпіновки, вказаної на початку прикладу (контакти інтерфейсу SPI, контакти D/C, CS, Reset, контакт керування підсвіткою LED й пару контактів живлення VCC, GND).

4. Перевірити роботу програми.

5. Додати в програму опитування сенсорного екрану з контролером XPT2046. Виводити в послідовний інтерфейс координату точки натиснення. Використати бібліотеку https://github.com/ThingPulse/XPT2046_Touchscreen

6. Розробити програму, яка реалізовуватиме графічний інтерфейс користувача з сенсорним вводом. На нього повинно виводитись завдання температури, кнопки +, - та ОК/Застосувати. Поточне значення завдання надсилати через послідовний інтерфейс на ПК.

7. Зробити висновки. Звіт повинен містити: титульний лист; тему, мету роботи; порядок виконання; створені програми; висновки.

Контрольні запитання

1. Які типи РКД використовуються в системах на базі мікроконтролерів?

2. Які інтерфейси підключення РКД до мікроконтролера ви знаєте?

3. Як працюють резистивні та ємнісні сенсорні екрани?

4. Які групи контактів можна виділити в модулі сенсорного дисплею за призначенням?

5. Які основні принципи проектування інтерфейсу користувача?

Лабораторна робота №13. Інфрачервоне дистанційне керування

Мета роботи: навчитись використовувати інфрачервоний пульт й інфрачервоний приймач для дистанційної передачі команд.

Теоретичні відомості

Інфрачервоний пульт дистанційного керування всюдисущий у повсякденному житті. Використовується для управління різними побутовими приладами, такими як телевізори, музичні центри, відеомагнітофони та приймачі супутникового сигналу. Інфрачервона система дистанційного керування складається з передавача (випромінювача інфрачервоного випромінювання) та інфрачервоного приймального модуля, а також мікроконтролера, здатного декодувати прийнятий сигнал.

Інфрачервоний несучий сигнал частотою 38 кГц, який випромінює пульт дистанційного керування, формується мікросхемою кодування на пульті дистанційного керування. Він складається з преамбули, адреси, інвертованої адреси, даних й інвертованих даних. Логічна 1 та логічний 0 відрізняються часовим інтервалом між імпульсами. Адреса одного й того ж пульта дистанційного керування не змінюється. Поле даних може містити код натиснутої кнопки. Коли натискається кнопка дистанційного керування, пульт дистанційного керування надсилає інфрачервоний сигнал. Коли IR-приймач отримує сигнал, програма розшифровує несучий сигнал і визначає, яка клавіша натиснута. MCU декодує отриманий сигнал 0/1, оцінюючи, яка клавіша натиснута на пульті дистанційного керування.

Модуль інфрачервоного приймача (рис. 7) містить головку інфрачервоного приймача, яка об’єднує прийом, підсилення та демодуляцію. Його вбудована мікросхема після демодуляції виводить прийнятий код у вигляді TTL-сумісного сигналу. Інфрачервоний приймальний модуль у лабораторній роботі має три контакти: сигнальну лінію, VCC і GND.

Рис. 7. Модуль інфрачервоного приймача

Коди кнопок пульта дистанційного керування, що використовується

в лабораторній роботі, наведені на рис. 8.

План роботи

1. Ознайомитись із складовими інфрачервоної системи дистанційного керування й кодами кнопок.

2. Запрограмувати керування дискретним навантаженням за командами з пульта.

3. Запрограмувати передачу інфрачервоним передавачем команди однією системою і прийом іншою.

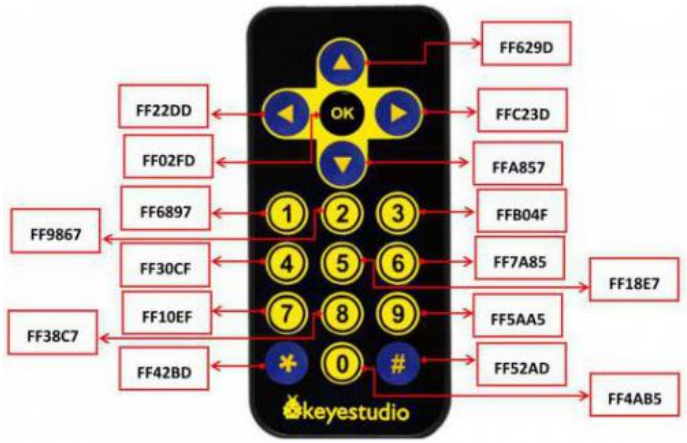


Рис. 8. Коди кнопок пульта дистанційного керування

Порядок виконання роботи

1. Завантажити бібліотеку IRremoteTank з https://www.dropbox.com/sh/yfip8hquyfmi54m/AABiWhMX0TiZnUiSaz4hs9-Ka/3.%20Tutorial%20-Arduino/2.%20Libraries?dl=0&subfolder_nav_tracking=1 та встановити у середовище Arduino.

2. Підключити інфрачервоного приймач до Arduino Uno, вихідний сигнал подати на аналоговий вхід A0.

3. Завантажити програму, яка надсилатиме у послідовний інтерфейс дані, прийняті від ІЧ-пульта:

```
#include <IRremoteTank.h>
int RECV_PIN = A0;
IRrecv irrecv(RECV_PIN);
decode_results results;
void setup()
{
        Serial.begin(9600);
```

```
        irrecv.enableIRIn(); //ввімкнення прийому
    }
     void loop() {
      if (irrecv.decode(&results))//якщо декодування завершилось
успішно, зчитати отримані дані
      {
        Serial.println(results.value, HEX);//представити у
шістнадцятковому форматі отримані дані й надіслати відповідний
рядок символів у послідовний інтерфейс
        irrecv.resume(); // відновити прийом для отримання наступних
даних
      }
      delay(100);
    }
```

4. Модифікувати програму з попереднього завдання так, щоб при натисканні обраної кнопки вмикалось навантаження (запалювався світлодіод), а при натисканні іншої – вимикався.

5. Розробити систему керування воротами: при натисканні однієї кнопки пульта двигун вмикається для руху в одному напрямку, при натисканні другої — в іншому, третьої — зупиняється. Для зупинки воріт у крайніх положеннях використовуються кінцеві вимикачі, підключені до цифрових входів Arduino Uno.

6. Використовуючи модуль з інфрачервоним світлодіодом, запрограмувати іншу плату Arduino Uno так, щоб забезпечити передачу команд з кодами кнопок при отриманні відповідних даних через послідовний інтерфейс від комп'ютера (при введенні однієї літери в моніторі послідовного порту ворота повинні відкриватись, іншої — закриватись).

7. Зробити висновки. Звіт повинен містити: титульний лист; тему, мету роботи; порядок виконання; створені програми; висновки.

Контрольні запитання

1. Які компоненти є в інфрачервоній системі дистанційного керування?

2. Яка несуча частота використовується в лабораторній роботі?

3. Як приймач розрізняє логічну 1 та 0 в даній роботі?

4. Як в програмі організувати виконання дій при натисканні різних кнопок на пульті дистанційного керування?

Лабораторна робота №14. Підключення RFID-зчитувача до мікроконтролера через інтерфейс SPI

Мета роботи: навчитись використовувати RFID-зчитувачі на базі MFRC522 для реалізації системи контролю доступу; навчитись підключати електронні компоненти через інтерфейс SPI.

Теоретичні відомості

RFID (Radio-Frequency Identification) - технологія радіочастотної ідентифікації, яка дозволяє зчитувати дані з міток безконтактним способом.

RFID-мітка містить унікальний ідентифікатор (UID), який можна зчитати за допомогою RFID-модуля. Кожен електронний ключ (контактний або безконтактний) містить ідентифікатор, який може бути в переліку дозволених для доступу ідентифікаторів у системі контролю доступу. Відповідно особа, що володіє даним ключем, матиме доступ до приміщення, яке захищається системою контролю доступу. В протилежному випадку особа не зможе потрапити до нього. Як правило, в таких системах використовують соленоїдні засувки, що при подачі на обмотку напруги живлення надають або закривають можливість відкрити двері. Для зв'язку Arduino з соленоїдним замком використовують реле або транзистори.

RFID-мітка — це електронний пристрій, який використовується для радіочастотної ідентифікації об'єктів. Вона складається з двох основних компонентів:

1. Антена. Котушка дроту або друкована схема, яка використовується для прийому та передачі радіосигналів. Антена призначена для прийому енергії від зчитувача RFID, яка потім використовується для живлення мікрочіпа.

2. Мікрочіп (інтегральна схема). Мікросхема, яка зберігає унікальний ідентифікаційний код (UID) та інші дані. Мікросхема обробляє радіосигнали, модулює дані та передає їх назад до зчитувача. В залежності від типу мітки, мікросхема може мати різний обсяг пам'яті для зберігання даних.

RFID-модуль RC522 з мікросхемою MFRC522 працює на частоті 13.56 МГц і використовує протокол SPI для зв'язку з головним мікропроцесором. Бібліотека MFRC522 (автор - Мігель Балбоа), дозволяє взаємодіяти з RFID-модулем RC522 на платформі Arduino. Вона надає зручний інтерфейс для взаємодії з модулем, спрощуючи процес зчитування та запису даних на RFID-мітки. Основні функції та можливості бібліотеки наступні:

1. Ініціалізація модуля: Бібліотека дозволяє ініціалізувати модуль RC522, налаштовуючи необхідні параметри для його роботи.

2. Зчитування UID міток: Основна функція бібліотеки - зчитування

унікального ідентифікатора (UID) RFID-міток. Вона дозволяє отримати UID мітки, який можна використовувати для ідентифікації об'єктів.

3. Зчитування та запис даних: Бібліотека надає функції для зчитування та запису даних на RFID-мітки. Це дозволяє зберігати та отримувати інформацію з міток.

4. Аутентифікація: Бібліотека підтримує аутентифікацію для захищеного доступу до даних на RFID-мітках.

План роботи

1. Ознайомитись зі схемою підключення компонентів.

2. Запрограмувати зчитування всього вмісту RFID-мітки та передачу його через послідовний інтерфейс на комп'ютер.

3. Реалізувати систему контролю доступу на базі RFID.

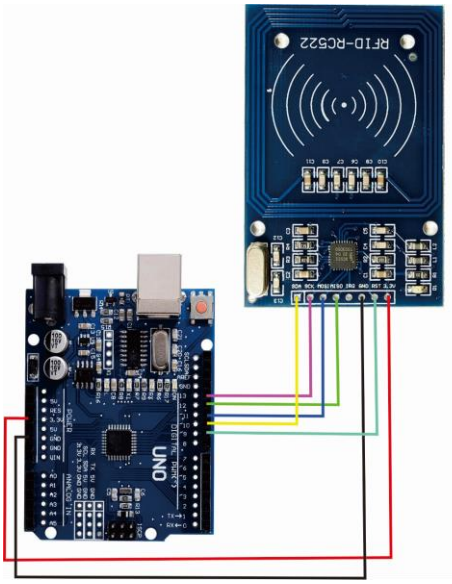
Порядок виконання роботи

1. Завантажити бібліотеку для роботи з RFID-модулем за посиланням https://github.com/miguelbalboa/rfid. Підключити завантажений ZIP-архів у середовище Arduino IDE в меню Скетч-Додати бібліотеку-Додати ZIP-бібліотеку. Після підключення бібліотеки в прикладах знайти DumpInfo та вивантажити в плату Arduino Uno.

2. Підключити до Arduino RFID-модуль на базі MFRC522 так, як показано на рис. 9. Для підключення RFID-модуля MFRC522 може використовуватись інтерфейс I^2C або SPI. В цій схемі використовується 4-провідний інтерфейс SPI та окремий провідник для сигналу скидання Reset (RST).

3. Зчитати RFID-мітку (картку або брелок), спостерігаючи виведений вміст картки у моніторі порту. Знайти UID мітки.

4. Розробити програму, що зчитуватиме UID та порівнюватиме його із заданим. Якщо ідентифікатори не співпадають — запалювати червоний світлодіод (“доступ заборонено”), а якщо співпадають — зелений (“доступ дозволено”). Одночасно доступ повинен сигналізуватись бузером.



Arduino	RFID-RC522
SDA	10
SCK	13
MOSI	11
MISO	12
GND	GND
RST	9
3.3V	3.3V

Рис. 9. Підключення модуля MFRC522 до Arduino Uno

5. Розробити програму, яка при прикладанні мітки зчитуватиме перший байт пам'яті мітки, зменшуватиме це число на 1 та записуватиме назад ("зніматиме" одну поїздку).

6. Зробити висновки. Звіт повинен містити: титульний лист; тему, мету роботи; порядок виконання; створені програми; висновки.

Контрольні запитання

1. Який принцип роботи RFID-технології?
2. Які основні компоненти RFID-системи?
3. На якій частоті працює RFID-модуль RC522?
4. Який протокол використовується для зв'язку між RFID-модулем та Arduino?
5. Як зчитати UID RFID-мітки за допомогою Arduino та RFID-модуля RC522?
6. Де на практиці використовуються RFID технології?

Лабораторна робота №15. Програмування MQTT-клієнта на мікроконтролері ESP8266

Мета: навчитись передавати й отримувати дані з мікропроцесорного пристрою з використанням протоколу MQTT.

Теоретичні відомості

MQTT (Message Queuing Telemetry Transport) — це легкий протокол обміну повідомленнями за моделлю видавець/підписник, розроблений для обміну повідомленнями між пристроями у мережах з низькою пропускною здатністю, високою затримкою або ненадійних мережах. Він часто використовується в Інтернеті речей (IoT) через його ефективність і низьке енергоспоживання.

MQTT широко використовується в таких сферах:

1. Інтернет речей (IoT): MQTT ідеально підходить для сенсорних мереж і пристроїв, які потребують мінімальних ресурсів для обміну даними.

2. Розумний дім: Використовується для зв'язку між різними домашніми пристроями, такими як розумні термостати, датчики та розумні побутові прилади.

3. Промисловий IoT: Використовується в системах моніторингу та управління виробничими процесами.

4. Мобільні додатки: Використовується для передачі повідомлень у режимі реального часу в додатках з низькою пропускною здатністю.

5. Міжмашинна взаємодія (M2M): Використовується для обміну даними між різними машинами в автоматизованих системах.

Протокол MQTT виділяють з-поміж інших такі переваги:

1. Легкість: Протокол має малу надлишковість, що дозволяє ефективно використовувати його навіть на пристроях з обмеженими ресурсами.

2. Ефективне використання пропускної здатності: Завдяки мінімальним обсягам даних, що передаються, MQTT може ефективно працювати в мережах з низькою пропускною здатністю.

3. Надійність: Протокол підтримує різні рівні якості обслуговування (QoS), що дозволяє забезпечити надійну доставку повідомлень навіть у ненадійних мережах.

4. Публікація/підписка: Модель "публікація/підписка" дозволяє ефективно обмінюватися даними між багатьма пристроями без необхідності встановлення прямих з'єднань між кожним пристроєм.

5. Масштабованість: Протокол добре масштабується, дозволяючи легко додавати нові пристрої в існуючу мережу.

6. Нижче енергоспоживання: Завдяки оптимізованому використанню ресурсів, MQTT краще підходить для пристроїв з батарейним живленням, ніж HTTP.

7. Безпека: MQTT підтримує шифрування через TLS і механізми аутентифікації та ACL для забезпечення безпечної передачі даних.

8. MQTT є стандартом де-факто для багатьох IoT-додатків.

Порядок виконання роботи

1. Запустити Arduino IDE. Додати підтримку плат з мікроконтролерами ESP8266 згідно інструкцій https://github.com/esp8266/Arduino?tab=readme-ov-file#installing-with-boards-manager

2. Встановити бібліотеку PubSubClient (автор Nick O'Leary) через Менеджер бібліотек Arduino.

3. Відкрити приклад mqtt_esp8266. Ознайомитись із прийомами роботи з елементами бібліотеки.

4. Модифікувати назви топіків за шаблоном номер_аудиторії/Прізвище/inTopic та номер_аудиторії/Прізвище/outTopic, задати SSID, пароль WiFi-мережі, брокер test.mosquitto.org, період надсилання повідомлення 5 с.

5. Скомпілювати й прошити плату з WiFi-модулем. Використовуючи монітор послідовного порта, переконатись у підключенні до брокера.

6. Встановити MQTT-клієнт на смартфон, наприклад, IoT MQTT Panel. Налаштувати його для підключення до брокера test.mosquitto.org. Створити віджети (“панелі”) типу Text Log та Switch для підписки на топік номер_аудиторії/Прізвище/outTopic та публікації в топік номер_аудиторії/Прізвище/inTopic відповідно.

7. Перевірити відповідність реакції запрограмованого пристрою на команди з смартфону та прийом даних.

8. Модифікувати вихідний код програми так, щоб в топік номер_аудиторії/Прізвище/inTopic можна було зі смартфону надіслати яскравість світлодіода [0…255], підключеного до GPIO 12, 14 або 15, і

отримане повідомлення з числом у функції callback перетворювалось у цілочисельну змінну й використовувалось як аргумент функції analogWrite(). Для цього використати функцію atoi або методи класу String:

String str = String(charBuf);

value_from_mqtt = str.toInt();.

9. Прошити плату й перевірити зміну яскравості світлодіода. На смартфоні для задання величини 0…255 можна додати елемент Slider.

10. Додати передачу напруги з фоторезистора або термістора, підключеного до контакту A0 за схемою подільника напруги. Значення АЦП публікувати в топік номер_аудиторії/Прізвище/A0.

11. Запрограмувати плату й перевірити прийом смартфоном результату АЦП і його зміну залежно від освітленості або температури. Зробити скріншот.

12. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст фінальної програми, скріншот екрану смартфону з попереднього пункту, висновок.

Контрольні запитання

1. Яка модель обміну повідомленнями в MQTT?

2. Чим відрізняються різні QoS в MQTT?

3. Чи потрібно мікропроцесорному пристрою мати публічну IP-адресу для передачі даних MQTT-брокеру?

4. Як задати функцію, що викликається при отриманні MQTT-повідомлення у бібліотеці PubSubClient?

5. Які пристрої можуть бути MQTT-клієнтами?

Лабораторна робота №16. Реалізація веб-інтерфейсу мікропроцесорного пристрою

Мета: навчитись програмувати мікропроцесорний пристрій для надання веб-інтерфейсу користувачу.

Теоретичні відомості

SDK для ESP8266 містить реалізацію простого веб-сервера. Найпростіший HTTP-сервер працює за принципом обробки запитів клієнтів (браузерів або інших пристроїв) та повернення відповідей. HTTP-сервер у відповідь на запит надає ресурси, такі як HTML-сторінки, зображення, або виконує певні дії (наприклад, зберігання даних). Для додавання веб-інтерфейсу в ESP8266 потрібно виконати наступне:

1. Додати підключення до заданої Wi-Fi мережі з використанням SSID та пароля.

2. Створити об’єкт сервера: у конструктор ESP8266WebServer передається номер порту (зазвичай 80).

3. Визначити шляхи, які сервер обробляє, й відповідні функції-обробники. Наприклад, кореневий каталог ("/") для відображення головної сторінки, /img.jpg для надсилання файлу зображення тощо.

4. Додати циклічну обробку запитів: в loop() циклічно викликається server.handleClient(), що відповідає за прослуховування сервером запитів від клієнтів та їх обробку, функції-обробники запитів повертають відповідь.

Сервер починає працювати і обробляти запити клієнтів.

Порядок виконання роботи

1. Запустити Arduino IDE. Додати підтримку плат з мікроконтролерами ESP8266 згідно інструкцій https://github.com/esp8266/Arduino?tab=readme-ov-file#installing-with-boards-manager

2. Відкрити приклад ESP8266WebServer. Розібрати призначення кожного рядка.

3. Скомпілювати й завантажити програму в плату.

4. На основі прикладу розробити програму, яка надаватиме можливість користувачеві ввести яскравість світлодіода (підключеного до GPIO 12, 14 або 15) в браузері.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
```

```
const char* ssid = "Your_SSID";          // Введіть назву вашої Wi-Fi мережі
const char* password = "Your_PASSWORD"; // Введіть пароль від вашої Wi-Fi мережі
ESP8266WebServer server(80);
String inputValue = "";  // Змінна для збереження введених даних

void setup() {
  Serial.begin(115200);
  delay(10);
  // Підключення до Wi-Fi
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
   WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  // Налаштування веб-сервера
  server.on("/", handleRoot);
  server.on("/submit", handleSubmit);
  server.begin();
  Serial.println("HTTP server started");
}

void handleRoot() {
  String html = "<html>\
   <body>\
```

```
  <h1>ESP8266 Web Form</h1>\
  <form action='/submit' method='POST'>\
          Enter something: <input type='text' name='inputValue'><br>\
          <input type='submit' value='Submit'>\
  </form>\
  <p>Last input: " + inputValue + "</p>\
  </body>\
 </html>";
 server.send(200, "text/html", html);
}

void handleSubmit() {
 if (server.hasArg("inputValue")) {
  inputValue = server.arg("inputValue");
 if (inputValue != "") {
  int angle = inputValue.toInt();
  …
  } else {inputValue = "Empty Data";}
 } else {
  inputValue = "No data";
 }
 server.sendHeader("Location", "/");
 server.send(303);
}

void loop() {
 server.handleClient();
}
```

5. Замість світлодіода підключити схему керування двигуном постійного струму. Внести відповідні зміни в програму. Перевірити зміну швидкості обертання двигуна залежного від значення, введеного в браузері. Зробити скріншот веб-сторінки.

6. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст фінальної програми, скріншот веб-сторінки з попереднього пункту, висновок.

Контрольні запитання

1. Який мережевий протокол використовується для взаємодії пристрою з браузером користувача?
2. Який HTML-тег використовується для виведення форми на сторінці веб-інтерфейсу?
3. Який HTML-тег використовується для виведення тексту на сторінці веб-інтерфейсу?
4. Що означають коди відповіді 200 і 303 в коді програми?
5. Який метод класу ESP8266WebServer задає функцію-обробник переходу за певною адресою на сайті?
6. Як задати мережевий порт, на якому веб-сервер очікуватиме вхідних з'єднань?

Лабораторна робота №17. Програмування Telegram-бота на мікроконтролері ESP32

Мета: Навчитися програмувати ESP32 для взаємодії з Telegram API. Розробити Telegram-бот, що керує дискретним електричним навантаженням.

Теоретичні відомості

ESP32 — це серія мікроконтролерів типу «система на кристалі», що мають інтегровані контролери Wi-Fi і Bluetooth, низьке енергоспоживання і невисоку ціну. У серії ESP32 використовується ядро Tensilica Xtensa LX6 в двоядерних та одноядерних варіаціях та є вбудовані антенні перемикачі, радіочастотний тракт, підсилювач потужності, приймач з низьким рівнем шумів, фільтри та модулі керування живленням. Має широкий набір периферійних інтерфейсів (GPIO, UART, SPI, I2C, ADC, DAC тощо). Також є версії з архітектурою RISC-V.

ESP32 широко використовується в IoT-пристроях, домашній автоматизації, носімих пристроях та інших застосуваннях. Завдяки вбудованому Wi-Fi, ESP32 добре підходить для підключення до мережі

Інтернет розроблюваного пристрою. Bluetooth дозволяє взаємодіяти з іншими Bluetooth-пристроями.

ESP32 можна програмувати за допомогою Arduino IDE, ESP-IDF (Espressif IoT Development Framework) та інших інструментів.

Telegram-боти

Telegram-боти — це спеціальні облікові записи в месенджері Telegram, керовані програмним кодом. Вони автоматизують взаємодію з користувачами, надаючи різноманітні сервіси: від простих сповіщень до складних інтерактивних сервісів.

Боти можуть використовуватись для таких функцій як:

- Надсилання та отримання повідомлень.
- Обробка команд користувачів.
- Надання інформації (новини, прогнози погоди тощо).
- Інтеграція з іншими сервісами та API.
- Керування IoT-пристроями.

Для створення та управління ботами використовується Telegram Bot API — набір HTTP-методів. API дозволяє ботам отримувати оновлення (повідомлення, команди) та надсилати відповіді. Для взаємодії з API необхідно отримати токен бота — унікальний ідентифікатор, що надається BotFather.

BotFather — це офіційний бот Telegram, призначений для створення та управління іншими ботами. За допомогою BotFather можна:

- Створити нового бота.
- Отримати токен доступу.
- Налаштувати профіль бота (ім'я, опис, аватар).
- Керувати командами бота.

Приклад програми, що реалізує простий Telegram-бот

```
#include <WiFi.h>
#include <UniversalTelegramBot.h>

// Дані Wi-Fi
const char* ssid = "your_ssid";
const char* password = "your_password";

// Токен Telegram-бота
```

```
const char* telegram_token = "your_telegram_token";

// Ініціалізація бота
UniversalTelegramBot bot(telegram_token);

void handleNewMessages(int numNewMessages) {
  for (int i = 0; i < numNewMessages; i++) {
    String chat_id = String(bot.messages[i].chat_id);
    String text = bot.messages[i].text;

    if (text == "/start") {
      bot.sendMessage(chat_id, "Привіт! Я ваш ESP32 бот.");
    } else if (text == "/info") {
      bot.sendMessage(chat_id, "ESP32 бот, розроблений для лабораторної роботи.");
    } else {
      bot.sendMessage(chat_id, "Невідома команда.");
    }
  }
}

void setup() {
  Serial.begin(115200);

  // Підключення до Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Підключення до Wi-Fi...");
  }
  Serial.println("Підключено до Wi-Fi");

  // Ініціалізація бота
  bot.begin();
}
```

```
void loop() {
 // Отримання нових повідомлень
 int numNewMessages = bot.getUpdates(bot.last_message_received + 1);

 // Обробка повідомлень
 if (numNewMessages) {
  handleNewMessages(numNewMessages);
 }

 delay(1000);
}
```

Бібліотека UniversalTelegramBot

Ініціалізація та налаштування

Конструктор:

UniversalTelegramBot bot(botToken, client);

- botToken: Ваш унікальний токен бота, отриманий від BotFather.
- client: Об'єкт WiFiClientSecure або ESP8266WiFiClientSecure, який використовується для захищеного HTTPS з'єднання з серверами Telegram.

bot.longPoll = seconds;

- Встановлює час (у секундах), протягом якого бот буде чекати нових повідомлень, перш ніж повернути відсутність повідомлень. Це зменшує кількість запитів та використання даних, але може заблокувати виконання коду на ESP8266 під час очікування.
- Приклад: bot.longPoll = 60; (чекати 60 секунд)

Отримання повідомлень:

int bot.getUpdates(long offset);

- Отримує всі нові (непрочитані) повідомлення від Telegram.
- offset: Ідентифікатор останнього обробленого повідомлення. Це важливо для того, щоб не отримувати одні й ті ж повідомлення повторно. Зазвичай, після обробки повідомлення, offset оновлюється до message.update_id + 1.
- Повертає кількість отриманих повідомлень.

- Отримані повідомлення зберігаються у внутрішньому масиві bot.messages. Ви можете перебирати їх за допомогою індексу (наприклад, bot.messages[i]).

Структура *message* (для отриманих повідомлень). Кожне отримане повідомлення зберігається в об'єкті типу *message*, який містить такі властивості:

- message.update_id: Унікальний ідентифікатор оновлення.
- message.chat_id: Ідентифікатор чату, з якого надійшло повідомлення.
- message.sender_id: Ідентифікатор відправника повідомлення.
- message.from_id: Ідентифікатор відправника повідомлення (застаріле, краще використовувати sender_id).
- message.from_name: Ім'я відправника повідомлення.
- message.text: Текст повідомлення.
- message.date: Час надходження повідомлення (у форматі Unix timestamp).
- message.type: Тип повідомлення (наприклад, "text", "photo", "sticker" тощо).
- message.photo[0...N].file_id: Якщо повідомлення містить фото, тут будуть ідентифікатори файлів різних розмірів.
- message.photo[0...N].width: Ширина фото.
- message.photo[0...N].height: Висота фото.
- message.document.file_id: Ідентифікатор файлу, якщо повідомлення містить документ.
- message.callback_query_id: Ідентифікатор для обробки натискань на inline-кнопки.
- message.callback_query_data: Дані, пов'язані з натиснутою inline-кнопкою.

Надсилання повідомлень:

bool bot.sendMessage(String chat_id, String text, String parse_mode = "", bool disable_web_page_preview = false, bool disable_notification = false, int reply_to_message_id = 0, String reply_markup = "");

- Надсилає текстове повідомлення в зазначений чат.
- chat_id: Ідентифікатор чату, куди надсилається повідомлення.
- text: Текст повідомлення.
- parse_mode (необов'язково): Форматування тексту ("Markdown", "HTML").

- disable_web_page_preview (необов'язково): Чи вимикати попередній перегляд веб-сторінок у повідомленні.
- disable_notification (необов'язково): Чи вимикати сповіщення для отримувачів.
- reply_to_message_id (необов'язково): Ідентифікатор повідомлення, на яке робиться відповідь.
- reply_markup (необов'язково): JSON-рядок для додавання клавіатури (ReplyKeyboardMarkup, InlineKeyboardMarkup).

bool bot.sendPhoto(String chat_id, String photo_file_id, String caption = "", String parse_mode = "", bool disable_notification = false, int reply_to_message_id = 0, String reply_markup = "");

- Надсилає фотографію за її file_id (якщо фото вже завантажено на сервери Telegram) або URL. Для надсилання локальних файлів потрібно використовувати інший підхід (див. sendPhoto з Stream).
- photo_file_id: Ідентифікатор файлу фотографії.
- caption (необов'язково): Підпис до фотографії.

bool bot.sendPhoto(String chat_id, Stream& photo, String filename, String caption = "", String parse_mode = "", bool disable_notification = false, int reply_to_message_id = 0, String reply_markup = "");

- Надсилає фотографію, використовуючи Stream об'єкт (наприклад, з файлової системи SPIFFS/LittleFS).
- photo: Об'єкт Stream, що містить дані зображення.
- filename: Ім'я файлу (для правильного MIME-типу).

bool bot.sendChatAction(String chat_id, String action);

- Надсилає дію чату (наприклад, "typing" - бот набирає текст, "upload_photo" - завантажує фото). Це корисно для інформування користувача, що бот щось робить.
- action: Рядок, що визначає дію (наприклад, "typing", "upload_photo", "record_video", "upload_video", "record_audio", "upload_audio", "upload_document", "find_location").

Додаткові методи:

Надсилання стікера

bool bot.sendSticker(String chat_id, String sticker_file_id, bool disable_notification = false, int reply_to_message_id = 0, String reply_markup = "");

Надсилання місцезнаходження

bool bot.sendLocation(String chat_id, float latitude, float longitude, bool disable_notification = false, int reply_to_message_id = 0, String reply_markup = "");

Надсилання документа з Stream

bool bot.sendDocument(String chat_id, Stream& document, String filename, String caption = "", bool disable_notification = false, int reply_to_message_id = 0, String reply_markup = "");

Отримання URL для завантаження файлу за його file_id

String bot.getFileID(String file_id);

Видалення повідомлення з чату

bool bot.deleteMessage(String chat_id, int message_id);

Відповідь на callback_query (натискання inline-кнопки)

bool bot.answerCallbackQuery(String callback_query_id, String text = "", bool show_alert = false, String url = "", int cache_time = 0);

- callback_query_id: Ідентифікатор callback-запиту.
- text (необов'язково): Текст, що відображається користувачеві (може бути пустим).
- show_alert (необов'язково): Чи показувати текст у вигляді спливаючого вікна.

Обробка inline-клавіатур:

Бібліотека підтримує створення inline-клавіатур, які дозволяють користувачам взаємодіяти з ботом без надсилання текстових команд.

Створення InlineKeyboardMarkup:

Створюється JSON-рядок, який описує кнопки.

```
String keyboardJson = "[[\"Кнопка 1\", \"data1\"],[\"Кнопка 2\", \"data2\"]]";
// Або складніше з ArduinoJson
DynamicJsonDocument doc(1024);
JsonArray row1 = doc.to<JsonArray>();
row1.add("Text Button 1");
row1.add("callback_data_1");
// ...
String replyMarkup = doc.as<String>();

bot.sendMessage(chat_id, "Виберіть опцію:", "", false, false, 0, replyMarkup);
```

де callback_data_1 - це дані, які будуть надіслані боту, коли користувач натисне кнопку.

Обробка Callback Query: Коли користувач натискає inline-кнопку, бот отримує повідомлення з типом callback_query. Ви можете отримати дані за допомогою message.callback_query_data та ідентифікатор запиту за допомогою message.callback_query_id. Після обробки запиту важливо

викликати bot.answerCallbackQuery() для підтвердження отримання.

```
if (bot.messages[i].type == "callback_query") {
        String callbackQueryId = bot.messages[i].callback_query_id;
        String callbackData = bot.messages[i].callback_query_data;
        String chatId = bot.messages[i].chat_id;

        if (callbackData == "data1") {
           bot.sendMessage(chatId, "Ви обрали Кнопку 1!");
        }
        // Завжди відповідайте на callback query
        bot.answerCallbackQuery(callbackQueryId, "Отримано!");
    }
```

Порядок виконання роботи

1. Запустити Arduino IDE. Додати підтримку плат з мікроконтролерами ESP32 згідно інструкцій https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html

2. Встановити бібліотеку Universal Arduino Telegram Bot.

3. Створити нового Telegram-бота за допомогою BotFather і отримати токен доступу до бота.

4. Використовуючи приклад з теоретичних відомостей, напишіть програму для ESP32 з використанням бібліотеки Universal Arduino Telegram Bot, яка дозволяє ввімкнути й вимкнути світлодіод командами /on і /off.

5. Завантажити програму в мікроконтролер і налагодити роботу програми.

6. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст програми, скріншот екрану з програмою Telegram з діалогом зі створеним Telegram-ботом, висновок.

Контрольні запитання

1. Що таке Telegram-бот?

2. Як створити Telegram-бот?

3. Як підключити ESP32 до Telegram API?

4. Які команди можна реалізувати в Telegram-боті?

5. Який протокол використовує ESP32 для взаємодії з серверами Telegram?

Лабораторна робота №18. Використання GPIO в Broadcom BCM2711 архітектури ARM Cortex-A72 для вводу-виводу дискретних сигналів

Мета роботи: навчитись розробляти програми для цифрового вводу-виводу з використанням GPIO одноплатних комп'ютерів, що працюють під керуванням ОС на базі GNU/Linux.

Теоретичні відомості

Для використання портів вводу-виводу загального призначення GPIO ядро Linux повинне бути скомпільовано з активованою їх підтримкою для потрібної плати/процесора/SoC.

Щоб взаємодіяти з портами вводу-виводу загального призначення GPIO з власної прикладної програми, як правило, використовують бібліотеки, що взаємодіють з відповідним API ядра Linux. Однією з поширених таких бібліотек для одноплатного комп'ютера Raspberry Pi є WiringPi.

Її можна отримати, завантаживши вміст її Github-репозиторію командою *git clone https://github.com/WiringPi/WiringPi.git ~/wiringpi*

Для початку потрібно підключити бібліотеку:

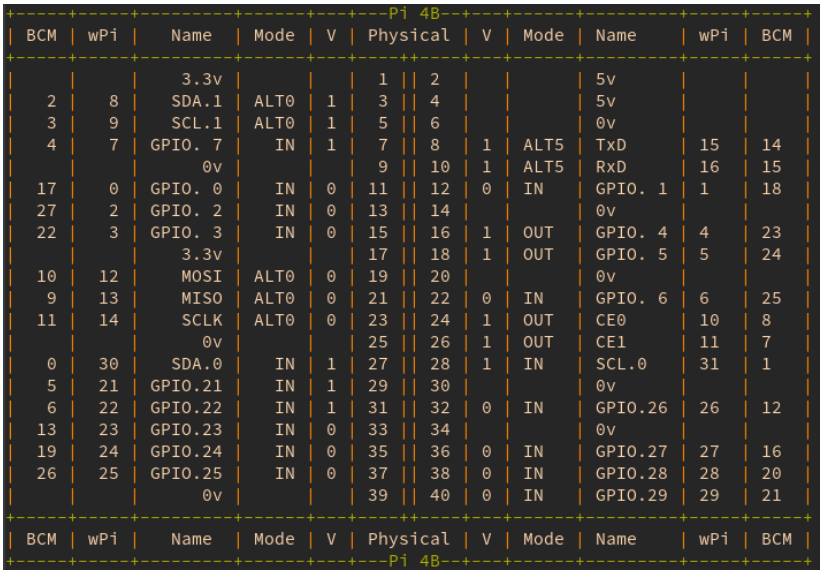
#include <wiringPi.h>

Першим кроком має бути ініціалізація. На цьому етапі визначається, яку схему нумерації пінів ви використовуватимете у програмі. Вкажіть один із цих викликів функцій, щоб ініціалізувати бібліотеку:

wiringPiSetup(); // ініціалізація з використанням спрощеної нумерації GPIO

wiringPiSetupGpio(); // ініціалізація з використанням нумерації Broadcom GPIO

У схемі нумерації Broadcom номери пінів відповідають порядку виводів GPIO на процесорі (SoC). Спрощена нумерація GPIO використовує номера від 0 до 16 для ідентифікації контактів. Побачити нумерацію фізичних контактів у різних схемах можна, виконавши команду *gpio readall* (рис. 10).



BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
					Pi 4B					
		3.3v			1 || 2			5v		
2	8	SDA.1	ALT0	1	3 || 4			5v		
3	9	SCL.1	ALT0	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	1	ALT5	TxD	15	14
		0v			9 || 10	1	ALT5	RxD	16	15
17	0	GPIO. 0	IN	0	11 || 12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13 || 14			0v		
22	3	GPIO. 3	IN	0	15 || 16	1	OUT	GPIO. 4	4	23
		3.3v			17 || 18	1	OUT	GPIO. 5	5	24
10	12	MOSI	ALT0	0	19 || 20			0v		
9	13	MISO	ALT0	0	21 || 22	0	IN	GPIO. 6	6	25
11	14	SCLK	ALT0	0	23 || 24	1	OUT	CE0	10	8
		0v			25 || 26	1	OUT	CE1	11	7
0	30	SDA.0	IN	1	27 || 28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29 || 30			0v		
6	22	GPIO.22	IN	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	IN	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37 || 38	0	IN	GPIO.28	28	20
		0v			39 || 40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
					Pi 4B					

Рис. 10. Вивід команди gpio readall з нумерацією контактів

Далі слід налаштувати режим роботи GPIO на вхід, вихід або ШІМ-вихід. Це можна зробити функцією *pinMode([pin], [mode])*. *mode* може бути *INPUT, OUTPUT* або *PWM_OUTPUT* відповідно. Наприклад, pinMode(18, PWM_OUTPUT).

Цифровий вихід

Функцію *digitalWrite([pin], [HIGH/LOW])* можна використовувати для встановлення високого або низького рівня на цифровому виході. Функція аналогічна такій в Arduino.

Наприклад, щоб встановити вивід 23 як HIGH, можна викликати:

digitalWrite(23, HIGH);

ШІМ-вихід

Для окремих контактів, які підтримують апаратний ШІМ можна використовувати функцію *pwmWrite([pin], [0-1023])*, яка встановить коефіцієнт заповнення ШІМ-сигналу від 0 до 100%, що відповідає другому аргументу функції від 0 до 1023. Наприклад, *pwmWrite(18, 723)* встановить на контакті 18 коефіцієнт заповнення ШІМ-сигналу близько 70%.

Цифровий вхід

Зчитувати цифровий вхід можна функцією *digitalRead([pin])*. Наприклад,

if (digitalRead(17)==HIGH)

printf("Pin 17 is HIGH\n");

else

printf("Pin 17 is LOW\n");

виведе стан контакту 17.

Підтягуючі резистори

До цифрових входів можуть бути підключені вбудовані підтягуючі резистори (активована підтяжка до землі або до напруги живлення). Це забезпечує функція *pullUpDnControl([pin], [PUD_OFF, PUD_DOWN, PUD_UP])*, другим аргументом якої є вид підтяжки: вимкнута, до землі, до напруги живлення відповідно.

Наприклад, якщо до контакту 22 підключена кнопка і потрібно підтягнути її до напруги живлення:

pullUpDnControl(17, PUD_UP);

Затримки

WiringPi включає дві функції затримки: *delay([milliseconds])* і *delayMicroseconds([microseconds])*, прототипи яких аналогічні Arduino. Звичайна затримка delay() призупиняє виконання програми на певну кількість мілісекунд. Якщо потрібно відкласти, наприклад, на 2 секунди, виконання дії, використовується *delay(2000).*

Порядок виконання роботи

1. Перевірити доступність Raspberry Pi командою *ping ri41rpi.local*

2. Підключитись до Raspberry Pi командою *ssh student@ri41rpi.local*

3. Створити каталог з вашим прізвищем.

4. Склонувати репозиторій https://github.com/WiringPi/WiringPi.git у створений каталог.

5. Перейти в каталог wiringPi та скомпілювати бібліотеку WiringPi командою *./build*

6. Відкрити приклад Blink, розібрати ключові елементи програми.

7. Скомпілювати приклад компілятором gcc, вказавши додатково ключ -lwiringPi, щоб використати отриману бібліотеку. Запустити програму й перевірити стан світлодіода.

8. Відповідно до порядкового номеру в групі розробити програму для періодичного включення-виключення навантаження на вказаному контакті (таблиця 3).

Таблиця 3

Варіанти завдань для лабораторної роботи 18

Варіант	Номер фізичного контакту	Номер GPIO в WiringPi
1	29	21
2	31	22
3	33	23
4	35	24
5	37	25
6	11	0
7	7	17
8	22	6
9	38	28
10	3	8

9. Знайти одну з кнопок на CrowPi та визначити номер контакту/GPIO, на який надходить сигнал з неї. Модифікувати програму так, щоб світлодіод блимав лише при натиснутій кнопці.

10. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, тексти розроблених програм, висновок.

Контрольні запитання

1. Як підключитись до віддаленого вузла по SSH за його іменем?

2. Яка бібліотека може використовуватись для взаємодії з GPIO плати Raspberry Pi 4?

3. Яка функція WiringPi дозволяє задати режим нумерації контактів плати Raspberry Pi?

4. Як налаштувати контакт як цифровий вхід?

5. Як налаштувати контакт як цифровий вихід?

6. Як налаштувати контакт як ШІМ-вихід?

7. Якою функцією можна зчитати стан цифрового входу?

8. Якою функцією можна змінити стан цифрового виходу?

9. Якою функцією можна вивести ШІМ-сигнал 50%?

10. Як можна увімкнути вбудований підтягуючий резистор до напруги живлення?

11. Як можна увімкнути вбудований підтягуючий резистор до землі?

Лабораторна робота №19. Розробка програми з GUI для зчитування сигналу з датчиків та керування дискретним електричним навантаженням для Raspberry Pi 4B

Мета: Ознайомитися з основами розробки програм з графічним інтерфейсом (GUI) для Raspberry Pi 4B, навчитися зчитувати сигнали з датчиків та керувати дискретним електричним навантаженням.

Теоретичні відомості

Raspberry Pi має GPIO (General Purpose Input/Output), які дозволяють підключати різні датчики та керувати зовнішніми пристроями. Існує кілька способів підключення датчиків до Raspberry Pi, залежно від типу датчика та його інтерфейсу.

1. Цифрові датчики:

Цифрові датчики передають дані у вигляді цифрових сигналів (0 або 1). Зазвичай підключаються до GPIO напряму або через перетворювач логічних рівнів. Деякі датчики використовують інтерфейси I2C або SPI, реалізовані в Raspberry Pi апаратно, а датчики з іншими інтерфейсами (напр., 1-wire) повинні підключатись із використанням програмної реалізації відповідного інтерфейсу в Raspberry Pi.

2. Аналогові датчики:

Raspberry Pi не має вбудованого аналого-цифрового перетворювача (АЦП), тому для підключення аналогових датчиків необхідно використовувати зовнішній АЦП. АЦП підключається до Raspberry Pi через інтерфейс I2C або SPI.

Tkinter

Tkinter — це поширена бібліотека Python для створення графічних інтерфейсів користувача (GUI). Вона надає набір інструментів для створення вікон, кнопок, текстових полів, міток та інших елементів GUI.

Основні концепції Tkinter:

1. Віджети (Widgets):

Віджети — це основні будівельні блоки GUI.

Існують різні типи віджетів, такі як:

- Tk(): Головне вікно програми.
- Label: Мітка для відображення тексту або зображення.
- Button: Кнопка для виконання дій.
- Entry: Текстове поле для введення даних.
- Text: Багаторядкове текстове поле.
- Canvas: Область для малювання графіки.
- Frame: Контейнер для організації інших віджетів.

2. Геометричні менеджери (Geometry Managers):

Геометричні менеджери використовуються для розміщення віджетів у вікні.

Основні геометричні менеджери:

- pack(): Розміщує віджети у вікні один за одним.
- grid(): Розміщує віджети у вигляді таблиці.
- place(): Розміщує віджети в заданих координатах.

3. Події (Events):

Події — це дії користувача, такі як натискання кнопки, введення тексту або переміщення миші.

Tkinter дозволяє прив'язувати функції до подій для обробки дій користувача.

Розробка GUI з Tkinter передбачає:

1. Імпорт бібліотеки Tkinter:

```
import tkinter as tk
```

2. Створення головного вікна:

```
window = tk.Tk()
```

3. Створення віджетів:

```
label = tk.Label(window, text="Привіт, світ!")
```

```
button = tk.Button(window, text="Натисніть мене", command=my_function)
```

4. Розміщення віджетів у вікні:

```
label.pack()
button.pack()
```

5. Запуск головного циклу обробки подій:

```
window.mainloop()
```

Tkinter є корисним інструментом для швидкого створення простих графічних інтерфейсів. Для створення складніших GUI можна використовувати інші бібліотеки, такі як PyQt або Kivy.

Приклад програми на Python, яка використовує Tkinter для створення простого графічного інтерфейсу та RPi.GPIO для керування світлодіодом

```
import tkinter as tk
import RPi.GPIO as GPIO

# Налаштування GPIO
GPIO.setmode(GPIO.BCM)
led_pin = 18  # Пін GPIO, до якого підключено світлодіод
GPIO.setup(led_pin, GPIO.OUT)

# Функція для ввімкнення/вимкнення світлодіода
def toggle_led():
    if GPIO.input(led_pin):
        GPIO.output(led_pin, GPIO.LOW)
        led_button.config(text="Ввімкнути світлодіод")
    else:
        GPIO.output(led_pin, GPIO.HIGH)
        led_button.config(text="Вимкнути світлодіод")

# Створення вікна Tkinter
window = tk.Tk()
window.title("Керування світлодіодом")
```

```
# Створення кнопки для ввімкнення/вимкнення світлодіода
led_button = tk.Button(window, text="Ввімкнути світлодіод", command=toggle_led)
led_button.pack(pady=20)

# Запуск головного циклу Tkinter
window.mainloop()

# Очищення GPIO після завершення програми
GPIO.cleanup()
```

Порядок виконання роботи

1. За позначеннями на CrowPi, визначити до якого конакту Raspberry Pi 4B підключено пасивний інфрачервоний датчик руху.

2. Розробити на базі прикладу з теоретичних відомостей програму на Python, яка зчитує сигнал з датчика руху та вмикає світлодіод при виявленні руху.

3. Запустити й налагодити програму.

4. За позначеннями на CrowPi, визначити до якого конакту Raspberry Pi 4B підключено електромагнітне реле. Занотувати у звіт перелік електронних компонентів, задіяних у керуванні реле.

5. Змінити номер контакту в програмі на визначений у попередньому пункті. Перевірити роботу програми.

6. Оформити звіт, що повинен містити титульну сторінку, тему, мету роботи, порядок виконання, текст програми, скріншот екрану з програмою Telegram з діалогом зі створеним Telegram-ботом, висновок.

Контрольні запитання

1. Як підключити PIR-датчик до Raspberry Pi 4B?

2. Як керувати реле через GPIO Raspberry Pi (апаратна і прогамна сторони)?

3. Як відобразити дані з датчика в GUI?

4. Як реалізувати керування реле через GUI?

Список рекомендованої літератури

1. Проектування мікропроцесорних систем керування : навчальний посібник / І. Р. Козбур, П. О. Марущак, В. Р. Медвідь, В. Б. Савків, В. П. Пісьціо. Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2022. 324 с.

2. Мікропроцесори та мікроконтролери : навч. посіб. / Д. Д. Татарчук, Ю. В. Діденко. Київ : КПІ ім. Ігоря Сікорського, 2020. 238 с.

3. ATmega328/P AVR MCU with picoPower Technology Data Sheet [Електронний ресурс] / Microchip Technology Inc, 2018. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/ATmega328_P%20AVR%20MCU%20with%20picoPower%20Technology%20Data%20Sheet%2040001984A.pdf .

4. RM0091 Reference manual [Електронний ресурс] / STMicroelectronics, 2017. URL: https://www.st.com/resource/en/reference_manual/dm00031936-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

Інформаційні ресурси

1. FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions. URL: http://www.freertos.org/.

2. Arduino Docs | Arduino Documentation. URL: https://docs.arduino.cc/.

3. ESP8266 Arduino Core’s documentation. URL: https://arduino-esp8266.readthedocs.io/en/latest/.

4. ESP32 Arduino Core’s documentation. URL: https://docs.espressif.com/projects/arduino-esp32/en/latest/.

5. Raspberry Pi Documentation. URL: https://www.raspberrypi.com/documentation/.

6. Documentation WiringPi-library. URL: https://github.com/WiringPi/WiringPi/blob/master/documentation/english/functions.md.

7. Raspberry-gpio-python Wiki. URL: https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/.