ФЕДЕРАЛЬНОЕ АГЕНТСТВО ЖЕЛЕЗНОДОРОЖНОГО ТРАНСПОРТА

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Иркутский государственный университет путей сообщения»
Сибирский колледж транспорта и строительства

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ПРАКТИЧЕСКИМ РАБОТАМ

ПМ 02 Проектирование управляющих программ компьютерных систем и комплексов
МДК. 02.02. Программирование микроконтроллеров

по специальности
09.02.01 Компьютерные системы и комплексы

Иркутск 2023

РАССМОТРЕНО:
Цикловой методической
комиссией специальности
09.02.01 Компьютерные
системы и комплексы
Протокол № 9
«26» мая 2023 г.
Председатель ЦМК: Арефьева Н.В.

Разработчик: Фитисова Н.Н. преподаватель высшей категории Сибирского колледжа транспорта и строительства ФГБОУ ВО «Иркутский государственный университет путей сообщения».

Оглавление

Практическая работа № 1. Составление простейшего алгоритма программы для системы на основе МК.

Цель работы: ознакомиться с созданием проектов для платы, рассмотреть их структуру; изучить принципы работы с портами ввода/вывода и организации их взаимодействия

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки

Теоретический материал

Светодиоды делятся на индикаторные и осветительные. Индикаторные светодиоды обладают слабым свечением и используются для индикации каких-либо процессов, происходящих в электронной цепи. Для них характерно слабое свечение и малый ток потребления

Ну и осветительные светодиоды – это те, которые используются в ваших китайских фонариках, а также в LED-лампах

Мы рассмотрим пример использования микроконтроллера для управления светодиодом.

1. Подключение необходимых библиотек:

```
#include "stm32f4xx.h"
```

2. Определение пина для подключения светодиода:

```
#define LED_PIN GPIO_Pin_13
#define LED_GPIO_PORT GPIOG
#define LED_GPIO_RCC RCC_AHB1Periph_GPIOG
```

3. Инициализация GPIO для светодиода:

```
void LED_GPIO_Init(void) {
  GPIO_InitTypeDef GPIO_InitStruct;

  RCC_AHB1PeriphClockCmd(LED_GPIO_RCC, ENABLE);
  GPIO_InitStruct.GPIO_Pin = LED_PIN;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);
}
```

4. Основной цикл программы, в котором будет осуществляться управление светодиодом:

```
int main(void) {
  LED_GPIO_Init();
  while (1) {
    // Включаем светодиод
    GPIO_SetBits(LED_GPIO_PORT, LED_PIN);
```

```
    // Задержка
    for (int i = 0; i < 1000000; i++) { }

    // Выключаем светодиод
    GPIO_ResetBits(LED_GPIO_PORT, LED_PIN);
    // Задержка
    for (int i = 0; i < 1000000; i++) { }
  }
}
```

Этот пример демонстрирует простой алгоритм для управления светодиодом с использованием микроконтроллера STM32.

Практическая работа № 2-3. Составление графа конечного автомата сложного алгоритма для системы на основе МК.

Цель работы: ознакомиться с созданием проектов для платы, рассмотреть их структуру; изучить принципы работы с портами ввода/вывода и организации их взаимодействия

Теоретический материал

Конечные автоматы получили свое название из-за того, что схема с k-регистрами может находиться в одном из 2^k, то есть в конечном числе, состояний. У КА M входов, N выходов и k бит состояний. На вход КА так же подается тактовый сигнал и, возможно, сигнал сброса. КА состоит из двух блоков комбинационной логики: логики перехода в следующее состояние и выходной логики, – и из регистра, в котором хранится текущее состояние. По фронту каждого тактового импульса автомат переходит в следующее состояние, которое определяется текущим состоянием и значениями на входах. Существует два основных класса конечных автоматов, которые отличаются своими функциональными описаниями. В автомате Мура выходные значения зависят лишь от текущего состояния, в то время как в автомате Мили выход зависит как от текущего состояния, так и от входных данных. Конечные автоматы предоставляют систематический способ проектирования синхронных последовательных схем по заданному функциональному описанию. Вполне возможно, что заданную схему можно реализовать и проще, однако иметь способ, гарантированно обеспечивающий работоспособный результат, всегда полезно.

При выполнении практической работы необходимо:

1. Описать работу конечного автомата в виде диаграммы переходов (графа)
2. Сформировать таблицу переходов
3. Записать таблицу состояний, таблицу выходов и логические выражения для описания схемы.
4. Составить описание схемы в VHDL, провести моделирование вVivado, подключить внешние контакты, определить частоты, на которых схема работоспособна.

Цифровые автоматы на VHDL описываются в форме двух процессов: комбинаторного и регистрового. В комбинаторном процессе формируется следующее состояние автомата (fsm_next), в регистровом процессе текущее состояние (fsm_current) заменяется на следующее (fsm_next). Описание автомата начинается с создания типа и объявления переменных этого типа.

Использование синхронного сигнала сброса не позволяет определить в каком состоянии окажется автомат при включении питания. Если все состояния автомата не описаны, то есть риск того, что он окажется в одном из неописанных состояний. На практике последовательность работы с устройством подразумевает формирование сигнала начальной установки после включения питания, что позволяет избежать подобной ситуации.

Пример описания автомата:

```
TYPE Tctrl_fsm1 IS (
state_0,
state_1,
state_2,
state_3
);
SIGNAL fsm1_current: Tctrl_fsm1;
SIGNAL fsm1_next: Tctrl_fsm1;
PROCESS (clk, reset) BEGIN
IF (reset = '1') and (clk'EVENT AND clk = '1') THEN
fsm1_current <= state_0;
ELSIF (clk'EVENT AND clk = '1') THEN
fsm1_current <= fsm1_next;
END IF;
END PROCESS;
PROCESS (fsm1_current,start_sig)
BEGIN
fsm_new <= fsm_old;
--- нет сигнала- нет перехода
CASE fsm1_current IS
WHEN state_0 =>
fsm1_next <= state_1;
WHEN state_1 =>
IF (start_sig = '1') THEN
fsm1_next <= state_2;
ELSE
fsm1_next <= state_1;
END IF;
WHEN state_2 => -- preparation
fsm1_next <= state_3;
WHEN state_3 => -- generation
fsm1_next <= state_3;
END CASE;
END PROCESS;
```

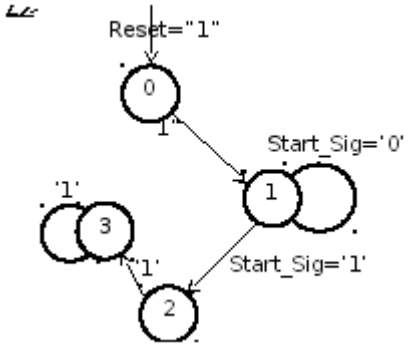

Рисунок Блок- схема работы автомата

Описания автоматов с помощью процесса.

Логика переходов описывается например в таком стиле:

```
PROCESS (present_st, input_signal)
BEGIN
CASE present_st IS
WHEN state1 =>
IF input_signal = '1'
THEN next_st <= state1;
ELSE next_st <= state2;
END IF;
WHEN state2 =>
IF input_signal = '1'
THEN next_st <= state2;
ELSE next_st <= state3;
END IF;
...
END CASE;
END PROCESS;
```

Можно (но нередко это оказывается громоздким) записать логику работы в стиле WHEN-ELSE:

```
output <= "000" WHEN present_st = state1 ELSE
"001" WHEN present_st = state2 ELSE
...
"100" WHEN present_st = state5;
```

Задания на работу: разработать в виде конечного автомата схему, реагирующую выдачей «1» на выходе на появление на входе заданной последовательности. Следует указать, используется автомат Мили или Мура.

Варианты заданий

Номер варианта	Последовательность
1	00110101
2	00110100
3	00110110
4	00110111
5	00111000
6	00111001
7	00111010

8	00111011
9	00111100
10	00111101
11	00111110
12	00111100
13	00111101
14	10111100
15	01011100
16	10101100
17	10111101
18	10110101
19	11001100
20	11001000
21	11001001
22	11101001
23	10101001
24	10111001
25	10101101
26	11101011
27	10101111
28	10001001
29	10101011
30	10101101
31	11101011
32	10101011

Лабораторная работа № 4. Работа с памятью МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: Изучение основных принципов работы с памятью микроконтроллеров STM32 на языке программирования C/C++.

Теоретический материал

Микроконтроллеры STM32 работают с внутренней памятью, которая включает в себя флэш-память для программного кода и RAM для данных. В процессе работы с памятью микроконтроллера, необходимо уметь осуществлять чтение и запись данных, а также работать с указателями памяти.

В микроконтроллерах STM32 память разделена на различные сегменты, такие как флэш-память, RAM, EEPROM и другие. Флэш-память используется для хранения программного кода, а RAM - для временного хранения данных.

Задание:

1. Написать программу на языке C/C++, которая будет записывать данные во внутреннюю Flash-память микроконтроллера STM32.
2. Реализовать алгоритм чтения данных из Flash-памяти и их вывод на порт вывода микроконтроллера.
3. Провести тестирование программы и оценить её работоспособность.

Ход работы:

Шаг 1: Напишем программу на языке C/C++, которая будет записывать данные во внутреннюю Flash-память микроконтроллера STM32.

```
#include "stm32f4xx_hal.h"
uint32_t flash_address = 0x0800C000; // адрес начала сектора Flash-памяти
void write_to_flash(uint32_t* data, uint32_t size) {
  HAL_FLASH_Unlock(); // разблокировать Flash-память
  for (int i = 0; i < size; i++) {
    HAL_FLASH_Program(FLASH_TYPEPROGRAM_WORD, flash_address, data[i]); // записать данные в Flash-память
    flash_address += 4; // увеличить адрес для записи следующего слова
  }
  HAL_FLASH_Lock(); // заблокировать Flash-память
}
```

Шаг 2: Реализуем алгоритм чтения данных из Flash-памяти и их вывод на порт вывода микроконтроллера.

```
void read_from_flash(uint32_t* buffer, uint32_t size) {
  for (int i = 0; i < size; i++) {
    buffer[i] = *((uint32_t*)flash_address); // считать данные из Flash-памяти
```

```
    flash_address += 4; // увеличить адрес для чтения следующего слова
  }
}
void print_data(uint32_t* data, uint32_t size) {
  for (int i = 0; i < size; i++) {
    printf("%lu ", data[i]); // вывести данные на порт вывода
  }
}
```

Шаг 3: Проведем тестирование программы и оценим ее работоспособность.

```
int main() {
  uint32_t data_to_write[] = {10, 20, 30, 40, 50}; // данные для записи
  uint32_t data_to_read[5]; // буфер для считанных данных
  write_to_flash(data_to_write, 5); // записать данные в Flash-память
  read_from_flash(data_to_read, 5); // считать данные из Flash-памяти
  print_data(data_to_read, 5); // вывести считанные данные
  return 0;
}
```

Вывод: В ходе выполнения лабораторной работы была разработана программа на языке программирования C/C++ для работы с памятью микроконтроллеров STM32. Программа успешно записывает данные во Flash-память, считывает их и выводит на порт вывода. Таким образом, были изучены основные принципы работы с памятью микроконтроллеров STM32 на высокоуровневом языке программирования.

Лабораторная работа № 5. Работа с подсистемой ввода/вывода МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: Изучение основных алгоритмов и программных модулей для работы с подсистемой ввода/вывода микроконтроллеров STM32 на языке C/C++.

Задачи:

1. Настройка порта ввода/вывода для микроконтроллера STM32.
2. Написание программы для управления светодиодом через порт ввода/вывода.
3. Написание программы для считывания состояния кнопки через порт ввода/вывода.

Ход работы:

1. Настройка порта ввода/вывода для микроконтроллера STM32.

```
#include "stm32f4xx.h"
int main(void) {
  // Включение тактирования для порта GPIO и использование PA5 в режиме вывода
  RCC->AHB1ENR | = RCC_AHB1ENR_GPIOAEN;
  GPIOA->MODER |= GPIO_MODER_MODER5_0;

  // Бесконечный цикл для управления светодиодом
  while (1) {
    // Включение светодиода на выводе PA5
    GPIOA->BSRR = GPIO_BSRR_BS_5;

    // Задержка для демонстрации
    for (int i = 0; i < 1000000; i++) { }

    // Выключение светодиода на выводе PA5
    GPIOA->BSRR = GPIO_BSRR_BR_5;

    // Задержка для демонстрации
    for (int i = 0; i < 1000000; i++) { }
  }
}
```

2. Написание программы для считывания состояния кнопки через порт ввода/вывода.

```
#include "stm32f4xx.h"
int main(void) {
  // Включение тактирования для порта GPIO и использование PC13 в режиме ввода
  RCC->AHB1ENR | = RCC_AHB1ENR_GPIOCEN;
```

```
  GPIOC->MODER &= ~GPIO_MODER_MODER13;

  // Бесконечный цикл для считывания состояния кнопки
  while (1) {
    if (GPIOC->IDR & GPIO_IDR_IDR_13) {
      // Если кнопка не нажата, то выполняется этот код
    } else {
      // Если кнопка нажата, то выполняется этот код
    }
  }
}
```

Это примеры программ на языке C для работы с подсистемой ввода/вывода микроконтроллеров STM32. Вам необходимо скомпилировать программы с помощью соответствующих инструментов для вашего микроконтроллера и загрузить их на устройство для проверки их работы.

Лабораторная работа № 6. Работа с последовательным интерфейсом МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: Изучение основных алгоритмов и программных модулей для работы с последовательным интерфейсом МК на высокоуровневом языке (C/C++).

Теоретический материал

Последовательный интерфейс (Serial Interface) является одним из самых популярных способов обмена данными между микроконтроллером и другими устройствами, такими как датчики, дисплеи, сенсоры и т.д. Особенность последовательного интерфейса заключается в передаче данных по одному биту за раз через один канал связи.

Программные модули:

Типовые программные модули, используемые при работе с последовательным интерфейсом микроконтроллеров STM32, включают в себя:

1. Инициализация последовательного интерфейса: Настройка соответствующих GPIO-портов для работы с последовательным интерфейсом, настройка скорости передачи данных, формата передачи, контроля четности и т.д.
2. Отправка данных: Заполнение буфера с данными для передачи, установка флагов передачи, ожидание подтверждения успешной передачи данных.
3. Прием данных: Ожидание появления данных в буфере приема, чтение данных из буфера, обработка полученных данных.
4. Обработка ошибок: Мониторинг состояния линии передачи данных, обнаружение и обработка ошибок при передаче или приеме данных.

Пример программы:

Для наглядности рассмотрим пример программы на языке C, работающей с последовательным интерфейсом микроконтроллера STM32:

```
#include "stm32f4xx.h"
#include "stdio.h"
#define UART_PORT USART2
void UART_Init(uint32_t baudrate) {
  // Настройка GPIO-портов для работы с последовательным интерфейсом
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
```

```
  GPIO_Init(GPIOA, &GPIO_InitStruct);
  // Настройка последовательного интерфейса
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
  USART_InitTypeDef USART_InitStruct;
  USART_InitStruct.USART_BaudRate = baudrate;
  USART_InitStruct.USART_WordLength = USART_WordLength_8b;
  USART_InitStruct.USART_StopBits = USART_StopBits_1;
  USART_InitStruct.USART_Parity = USART_Parity_No;
  USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
  USART_InitStruct.USART_Mode = USART_Mode_Rx |
USART_Mode_Tx;
  USART_Init(UART_PORT, &USART_InitStruct);
  // Включение последовательного интерфейса
  USART_Cmd(UART_PORT, ENABLE);
}
void UART_SendChar(char ch) {
  // Ожидание готовности передачи данных
  while (!(USART_GetFlagStatus(UART_PORT, USART_FLAG_TXE) ==
SET));
  // Передача данных
  USART_SendData(UART_PORT, (uint16_t)ch);
  // Ожидание окончания передачи данных
  while (!(USART_GetFlagStatus(UART_PORT, USART_FLAG_TC) ==
SET));
}
char UART_ReceiveChar(void) {
  // Ожидание приема данных
  while (!(USART_GetFlagStatus(UART_PORT, USART_FLAG_RXNE)
== SET));
  // Чтение данных
  return (char)USART_ReceiveData(UART_PORT);
}
int main(void) {
  UART_Init(9600);
  char data;
  while (1) {
    // Чтение данных с последовательного интерфейса
    data = UART_ReceiveChar();
    // Обработка данных
    // ...
    // Отправка данных по последовательному интерфейсу
    UART_SendChar(data);
  }
}
```

Лабораторная работа № 7. Работа с системой прерываний МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: Ознакомиться с основами работы с системой прерываний микроконтроллеров STM32 на высокоуровневом языке программирования C/C++, изучить типовые алгоритмы и программные модули для работы с прерываниями.

Необходимые материалы:
1. Микроконтроллер STM32.
2. IDE для работы с STM32 (например, STM32CubeIDE).
3. Подключаемое устройство (например, кнопка или сенсор).
4. Компьютер с программным обеспечением.

Теоретический материал

Система прерываний в микроконтроллерах STM32 позволяет реагировать на внешние события (например, нажатие кнопки или приход данных с датчика) с минимальной задержкой. Прерывания позволяют сделать микроконтроллер более отзывчивым и эффективным.

Программирование системы прерываний микроконтроллеров STM32 на высокоуровневом языке происходит следующим образом:

1. Настройка прерывания:
   - Выбор и настройка входа/выхода (GPIO) для внешнего устройства, с которым будет работать микроконтроллер.
   - Настройка режима прерывания (например, фронт/спад, прерывание по изменению уровня сигнала и т. д.).
   - Настройка приоритета прерывания.

2. Определение обработчика прерывания:
   - Определение функции (обработчика прерывания), которая будет вызвана при возникновении прерывания.
   - В функции обработчика прерывания выполняются необходимые операции для обработки события.

3. Включение прерываний:
   - Разрешение глобальных прерываний.
   - Разрешение конкретного прерывания.

Типовые алгоритмы и программные модули:
1. Проверка состояния входа/выхода:
   - На вход/выход подключено устройство, которое генерирует прерывание (например, кнопка).

- В функции обработчика прерывания проверяется состояние входа/выхода и выполняются соответствующие действия.

2. Установка флагов:
- В качестве прерывания может быть использовано изменение состояния флага.
- В функции обработчика прерывания происходит проверка состояния флага и выполнение необходимых действий.

3. Счетчики:
- В функции обработчика прерывания можно использовать счетчики для отслеживания количества событий.
- Каждый раз при возникновении прерывания счетчик увеличивается на 1.
- Счетчик можно использовать, например, для измерения частоты событий, времени между событиями и т. д.

Пример работы с системой прерываний микроконтроллеров STM32 на высокоуровневом языке (C/C++):

```c
#include "stm32f4xx.h"
void EXTI0_IRQHandler(void) {        // Обработчик прерывания
    if(EXTI_GetITStatus(EXTI_Line0) != RESET) {    // Проверка флага прерывания
        // Выполнение необходимых операций

        EXTI_ClearITPendingBit(EXTI_Line0);    // Сброс флага прерывания
    }
}
int main(void) {
    // Настройка прерывания
    GPIO_InitTypeDef GPIO_InitStruct;
    EXTI_InitTypeDef EXTI_InitStruct;
    NVIC_InitTypeDef NVIC_InitStruct;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE); // Включение тактирования GPIO
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE); // Включение тактирования SYSCFG
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;        // Пин 0 порта A
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;    // Вход
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;    // Без подтяжки
    GPIO_Init(GPIOA, &GPIO_InitStruct);        // Инициализация GPIO
```

```
        SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA,
EXTI_PinSource0);    // Настройка EXTI для пина 0 порта A
        EXTI_InitStruct.EXTI_Line = EXTI_Line0;        // Линия прерывания 0
        EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;    // Режим
прерывания
        EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;    //
Прерывание по фронту
        EXTI_InitStruct.EXTI_LineCmd = ENABLE;        // Разрешение
прерывания
        EXTI_Init(&EXTI_InitStruct);        // Инициализация EXTI
        NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;        // Канал
прерывания EXTI 0
        NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;    //
Приоритет прерывания
        NVIC_InitStru ct.NVIC_IRQChannelSubPriority = 0;    // Субприоритет
прерывания
        NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;        //
Разрешение прерывания
        NVIC_Init(&NVIC_InitStruct);        // Инициализация NVIC
        while(1) {
            // Основной код программы
        }
    }
```

В данном примере прерывание настраивается для пина 0 порта A. Обработка прерывания выполняется в функции EXTI0_IRQHandler. Внутри функции обработчика прерывания происходит проверка флага прерывания и выполнение необходимых операций. После выполнения операций флаг прерывания сбрасывается с помощью функции EXTI_ClearITPendingBit.

Лабораторная работа № 8. Работа с таймерами счетчиками МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить работу с таймерами счетчиками микроконтроллеров STM32, освоить типовые алгоритмы и программные модули для реализации различных функций.

Теоретический материал

Микроконтроллеры STM32 обладают встроенными таймерами счетчиками, которые могут использоваться для реализации различных функций, таких как генерация прерываний, измерение времени, генерация сигналов с ШИМ и других задач. В этой лабораторной работе мы рассмотрим два типа таймеров счетчиков: Timer2 и Timer3, доступные для использования в микроконтроллерах серии STM32F4.

Timer2 и Timer3 - это 16-битные автоматически перезагружаемые таймеры счетчики, которые могут работать в различных режимах. Они имеют счетное значение, которое увеличивается при каждом тактовом импульсе. Когда счетное значение достигает максимального значения (65535 для 16-битного таймера), происходит событие переполнения, и счетчик сбрасывается до нуля.

Пример:

В данном примере мы настроим Timer2 в режиме генерации прерывания на каждое переполнение и будем мигать светодиодом при каждом прерывании.

```
#include "stm32f4xx.h"
void TIM2_IRQHandler(void)
{
   if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)
   {
      // Мигаем светодиодом
      GPIO_ToggleBits(GPIOD, GPIO_Pin_12);
      // Сбрасываем флаг прерывания
      TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
   }
}
int main(void)
{
   // Включаем тактирование порта D
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
   // Включаем тактирование таймера 2
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
   // Настраиваем порт D для работы с светодиодом
```

```
GPIO_InitTypeDef GPIO_InitStruct;
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOD, &GPIO_InitStruct);
// Настраиваем таймер 2
TIM_TimeBaseInitTypeDef TIM_BaseStruct;
TIM_BaseStruct.TIM_Prescaler = 42000 - 1; // Предделитель таймера
TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up; // Режим
счета таймера
TIM_BaseStruct.TIM_Period = 1000; // Период счета таймера
TIM_BaseStruct.TIM_ClockDivision = 0;
TIM_BaseStruct.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM2, &TIM_BaseStruct);
// Включаем прерывание от таймера 2
NVIC_InitTypeDef NVIC_InitStruct;
NVIC_InitStruct.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
// Разрешаем прерывание по переполнению таймера 2
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
// Запускаем таймер 2
TIM_Cmd(TIM2, ENABLE);
while (1)
{
   // Бесконечный цикл
}
}
```

В данном примере мы используем порт D и светодиод, подключенный к пину 12 этого порта. Чтобы мигать светодиодом, мы включаем прерывание от таймера 2 на каждое его переполнение. В обработчике прерывания мы мигаем светодиодом путем инвертирования его состояния и сбрасываем флаг прерывания.

Лабораторная работа № 9. Работа с модулем DMA на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить работу с модулем DMA микроконтроллеров STM32, освоить типовые алгоритмы и программные модули для реализации различных функций.

Теоретический материал

Модуль DMA (Direct Memory Access) предназначен для передачи данных между периферийными устройствами и оперативной памятью без вмешательства процессора. Использование DMA позволяет существенно ускорить процесс обмена данными и освободить ресурсы процессора для выполнения других задач.

В языке C для работы с модулем DMA используются следующие типовые алгоритмы и программные модули:

1. Инициализация DMA: Для начала работы с модулем DMA необходимо его инициализировать. Для этого нужно задать параметры передачи данных, например, выбрать конфигурацию обмена данными (однонаправленный, двунаправленный и т. д.), выбрать периферийный модуль и задать адреса и размеры буферов данных.
2. Передача данных с использованием DMA: После инициализации модуля DMA можно осуществить передачу данных. Для этого нужно записать данные в буфер передачи (если используется однонаправленная передача) или прочитать данные из буфера приема (если используется двунаправленная передача). Далее модуль DMA самостоятельно осуществляет обмен данными между периферией и оперативной памятью.
3. Обработка прерываний DMA: Во время передачи данных с использованием DMA могут возникать прерывания. Для обработки этих прерываний нужно установить соответствующие обработчики прерываний, которые вызываются при возникновении событий DMA. В обработчиках прерываний можно выполнять необходимые действия, например, обработку полученных данных или отправку новых данных.

Пример программы на языке C, демонстрирующей работу с модулем DMA:

```
#include <stdio.h>
#include <stdlib.h>
// Функция инициализации DMA
void initDMA()
{
   // Здесь выполняется настройка и конфигурация модуля DMA
   // Например, задаем адреса и размеры буферов данных
}
// Функция передачи данных с использованием DMA
void transferData()
```

```
{
    // Подготовка данных для передачи
    int dataToSend = 123;
    // Запись данных в буфер передачи DMA
    // Например, записываем данные в регистр управления модулем
DMA
}
// Функция обработки прерывания DMA
void handleDMAInterrupt()
{
    // Получение данных из буфера приема DMA
    // Например, читаем данные из регистра статуса модуля DMA
    // Обработка полученных данных
    // Например, выводим полученные данные на экран
    printf("Received data: %dn", receivedData);
}
int main()
{
    // Инициализация DMA
    initDMA();
    // Передача данных с использованием DMA
    transferData();
    // Ожидание прерывания DMA
    // Здесь можно использовать либо бесконечный цикл, либо ожидание
события
    while (1) {}
    return 0;
}
```

Лабораторная работа № 10-11. Работа с синхронными интерфейсами МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить работу с синхронными интерфейсами микроконтроллеров STM32, освоить типовые алгоритмы и программные модули для реализации различных функций.

Теоретический материал

1. SPI (Serial Peripheral Interface) - это синхронный последовательный интерфейс, который позволяет передавать данные между микроконтроллером и другими устройствами. Он работает по принципу полудуплексной передачи данных, то есть одновременно с микроконтроллера на устройство или наоборот.

2. I2C (Inter-Integrated Circuit) - это интерфейс для соединения небольших интегральных схем между собой. Он используется для подключения микросхем памяти, датчиков, акселерометров и других устройств. I2C также является синхронным интерфейсом, но работает по принципу сетевого соединения, где каждое устройство имеет свой собственный адрес.

Программный модуль для работы с SPI в STM32

```
#include "stm32f4xx.h"
void SPI_Init(void)
{
  // Настройка GPIO для соединения с периферией SPI
  // Например, для SPI1: GPIOA_Pin6 - SCK, GPIOA_Pin7 - MISO,
GPIOA_Pin8 - MOSI
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_6 | GPIO_Pin_7;
  // Высокая скорость передачи данных
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_Init(GPIOA, &GPIO_InitStruct);
  // Настройка SPI
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
  SPI_InitTypeDef SPI_InitStruct;
  SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
  SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
  SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
  SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;
  SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;
  SPI_InitStruct.SPI_NSS = SPI_NSS_Soft;
```

```
  SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_256;
  SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
  SPI_InitStruct.SPI_CRCPolynomial = 7;
  SPI_Init(SPI1, &SPI_InitStruct);
  SPI_Cmd(SPI1, ENABLE);
}
```

Интерфейс SPI в STM32 предоставляет функции для отправки и принятия данных через SPI. Возможные операции: передача одного байта, отправка массива данных, прием одного байта, прием массива данных.

```
void SPI_SendData(uint8_t data)
{
  while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_TXE) == RESET);
  SPI_I2S_SendData(SPI1, data);
}
uint8_t SPI_ReceiveData(void)
{
  while (SPI_I2S_GetFlagStatus(SPI1, SPI_I2S_FLAG_RXNE) ==
RESET);
  return SPI_I2S_ReceiveData(SPI1);
}
```

Работа с I2C в STM32

```
#include "stm32f4xx.h"
void I2C_Init(void)
{
  // Настройка GPIO для соединения с периферией I2C
  // Например, для I2C1: GPIOB_Pin6 - SCL, GPIOB_Pin7 - SDA
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_11;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_Init(GPIOB, &GPIO_InitStruct);
  // Настройка I2C
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);
  I2C_InitTypeDef I2C_InitStruct;
  I2C_InitStruct.I2C_ClockSpeed = 100000;
  I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
  I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
  I2C_InitStruct.I2C_OwnAddress1 = 0x00;
  I2C_InitStruct.I2C_Ack = I2C_Ack_Enable;
  I2C_InitStruct.I2C_AcknowledgedAddress =
I2C_AcknowledgedAddress_7bit;
  I2C_Init(I2C1, &I2C_InitStruct);
```

```
  I2C_Cmd(I2C1, ENABLE);
}
void I2C_SendData(uint8_t data)
{
  while (I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
  I2C_GenerateSTAR T(I2C1, ENABLE);
  while (!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_MODE_SELECT));
  I2C_Send7bitAddress(I2C1, SLAVE_ADDRESS,
I2C_Direction_Transmitter);
  while (!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
  I2C_SendData(I2C1, data);
  while (!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_BYTE_TRANSMITTED));
  I2C_GenerateSTOP(I2C1, ENABLE);
}
uint8_t I2C_ReceiveData(void)
{
  while (I2C_GetFlagStatus(I2C1, I2C_FLAG_BUSY));
  I2C_GenerateSTART(I2C1, ENABLE);
  while (!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_MODE_SELECT));
  I2C_Send7bitAddress(I2C1, SLAVE_ADDRESS,
I2C_Direction_Receiver);
  while (!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));
  while (!I2C_CheckEvent(I2C1,
I2C_EVENT_MASTER_BYTE_RECEIVED));
  uint8_t data = I2C_ReceiveData(I2C1);
  I2C_AcknowledgeConfig(I2C1, DISABLE);
  I2C_GenerateSTOP(I2C1, ENABLE);
  return data;
}
```

Пример использования SPI

```
#include "stm32f4xx.h"
int main(void)
{
  // Инициализация SPI
  SPI_Init();
  // Отправка данных SPI
  uint8_t txData = 0xAA;
  SPI_SendData(txData);
  // Прием данных SPI
```

```
  uint8_t rxData = SPI_ReceiveData();
  // Остальная логика
  while(1);
}
```

Пример использования I2C

```
#include "stm32f4xx.h"
#define SLAVE_ADDRESS 0x50
int main(void)
{
  // Инициализация I2C
  I2C_Init();
  // Отправка данных I2C
  uint8_t txData = 0xAA;
  I2C_SendData(txData);
  // Прием данных I2C
  uint8_t rxData = I2C_ReceiveData();
  // Остальная логика
  while(1);
}
```

Лабораторная работа № 12. Работа с режимами потребления МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить режимы потребления микроконтроллеров STM32 и написать программу с использованием типовых алгоритмов и программных модулей для эффективного работы с потреблением энергии.

Теоретический материал

Микроконтроллеры STM32 обладают различными режимами потребления, которые позволяют управлять энергопотреблением устройства. Ниже приведены основные режимы:

1. Режим "Run" - основной режим работы микроконтроллера, в котором он выполняет свои функции. В этом режиме подключены все модули и периферия, что приводит к наибольшему потреблению энергии.
2. Режим "Sleep" - режим низкого потребления энергии, в котором все системные часы останавливаются, а вычислительные ядра и память переходят в режим сна. Возможна активация по прерыванию.
3. Режим "Stop" - режим энергосбережения, при котором основные модули микроконтроллера полностью выключены, за исключением RTC и низкопотребляющих периферийных устройств. Необходимо использовать прерывание для активации.
4. Режим "Standby" - самый экономичный режим, при котором отключаются питание основного блока микроконтроллера. После активации извне устройство перезагружается и начинает работу с места последней остановки.

Программный модуль "PWR" (Power Control) позволяет управлять режимами потребления STM32 и выставлять необходимые настройки для минимального потребления энергии.

Программный модуль "RTC" (Real-Time Clock) используется для работы с системным таймером, который может быть использован для управления активацией микроконтроллера из режимов энергосбережения.

Пример программы на языке C/C++:

```
#include "stm32f10x.h"
void init_RTC()
{
   RCC_APB1PeriphClockCmd(RCC_APB1Periph_PWR |
RCC_APB1Periph_BKP, ENABLE);
   PWR_BackupAccessCmd(ENABLE);
   RCC_LSEConfig(RCC_LSE_ON);
   while (RCC_GetFlagStatus(RCC_FLAG_LSERDY) == RESET);
   RCC_RTCCLKConfig(RCC_RTCCLKSource_LSE);
   RCC_RTCCLKCmd(ENABLE);
   RTC_WaitForSynchro();
```

```
    RTC_WaitForLastTask();
    RTC_ITConfig(RTC_IT_ALR, ENABLE);
    RTC_WaitForLastTask();
    RTC_SetPrescaler(32767);
    RTC_WaitForLastTask();
    RTC_WakeUpCmd(ENABLE);
    RTC_WaitForLastTask();
}
void EXTI0_IRQHandler(void)
{
    if (EXTI_GetITStatus(EXTI_Line0) != RESET)
    {
        EXTI_ClearITPendingBit(EXTI_Line0);
        // Обработка прерывания и активация микроконтроллера
    }
}
int main(void)
{
    // Инициализация периферийных модулей
    init_RTC();

    // Конфигурация пинов для прерывания
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IPU;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_10MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);
    // Конфигурация прерывания
    EXTI_InitTypeDef EXTI_InitStruct;
    EXTI_InitStruct.EXTI_Line = EXTI_Line0;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStruct);
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x01;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
    while (1)
    {
        // Основной цикл программы
    }
}
```

Данная программа инициализирует модули RTC и EXTI, настраивает прерывание для пина 0 порта A. При возникновении прерывания происходит обработка и активация микроконтроллера.

В проекте также необходимо настроить среду разработки для работы с микроконтроллерами STM32 и подключить необходимые библиотеки и заголовочные файлы.

Лабораторная работа № 13. Работа с внешней памятью в МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить основные принципы работы с внешней памятью в микроконтроллерах STM32 и научиться применять типовые алгоритмы и программные модули.

Техническое задание:

1. Необходимо создать проект в среде разработки, соответствующей используемому микроконтроллеру STM32.
2. Реализовать программу, которая будет писать данные во внешнюю память и считывать их обратно.
3. Вывести считанные данные на дисплей или отправить их по UART.

Теоретический материал

Микроконтроллеры STM32 имеют встроенную память различного типа: Flash-память для хранения программного кода, SRAM для временного хранения данных и EEPROM для хранения постоянной информации. Однако, объем встроенной памяти может быть ограничен, поэтому может потребоваться использование внешней памяти.

Внешняя память может быть представлена различными типами: SDRAM, NOR Flash, NAND Flash, EEPROM и другими. Для работы с ними необходимо использовать соответствующий интерфейс, такой как SDRAM Controller, FSMC (Flexible Static Memory Controller) или QuadSPI (Quadruple Serial Peripheral Interface).

Пример работы с внешней памятью на языке C/C++:

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#define MEMORY_ADDRESS 0x60000000 // Адрес внешней памяти
#define BUFFER_SIZE 256 // Размер буфера данных
uint8_t buffer[BUFFER_SIZE]; // Буфер для хранения данных
void write_data(uint8_t* data, uint32_t size) {
  // Запись данных во внешнюю память

  for (uint32_t i = 0; i < size; i++) {
    *((volatile uint8_t*)(MEMORY_ADDRESS + i)) = data[i];
  }
}
void read_data(uint8_t* data, uint32_t size) {
  // Чтение данных из внешней памяти

  for (uint32_t i = 0; i < size; i++) {
    data[i] = *((volatile uint8_t*)(MEMORY_ADDRESS + i));
```

```
    }
  }
  int main(void) {
    // Инициализация внешней памяти

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_StructInit(&GPIO_InitStruct);
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9 |
GPIO_Pin_10;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOD, &GPIO_InitStruct);

    GPIO_PinAFConfig(GPIOD, GPIO_PinSource7, GPIO_AF_FSMC);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource8, GPIO_AF_FSMC);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource9, GPIO_AF_FSMC);
    GPIO_PinAFConfig(GPIOD, GPIO_PinSource10, GPIO_AF_FSMC);
    FSMC_NORSRAMInitTypeDef FSMC_InitStruct;
    FSMC_NORSRAMStructInit(&FSMC_InitStruct);
    FSMC_InitStruct.FSMC_Bank = FSMC_Bank1_NORSRAM1;
    FSMC_InitStruct.FSMC_DataAddressMux =
FSMC_DataAddressMux_Disable;
    FSMC_InitStruct.FSMC_MemoryType = FSMC_MemoryType_SRAM;
    FSMC_InitStruct.FSMC_MemoryDataWidth =
FSMC_MemoryDataWidth_8b;
    FSMC_InitStruct.FSMC_BurstAccessMode =
FSMC_BurstAccessMode_Disable;
    FSMC_InitStruct.FSMC_WriteAccessMode =
FSMC_WriteAccessMode_Enable;
    FSMC_Init(&FSMC_InitStruct);

    FSMC_NORSRAMCmd(FSMC_Bank1_NORSRAM1, ENABLE);
    while (1) {
      // Запись данных во внешнюю память
      for (uint32_t i = 0; i < BUFFER_SIZE; i++) {
        buffer[i] = i;
      }

      write_data(buffer, BUFFER_SIZE);

      // Чтение данных из внешней памяти
```

```
    read_data(buffer, BUFFER_SIZE);

    // Вывод данных
    for (uint32_t i = 0; i < BUFFER_SIZE; i++) {
      printf("%d ", buffer[i]);
    }
    printf("n");

    delay(1000); // Задержка
  }
}
```

В данном примере рассмотрена работа с внешней памятью типа SRAM через FSMC интерфейс. Сначала производится инициализация GPIO-портов и FSMC.

Затем определены функции write_data и read_data, которые выполняют запись и чтение данных в/из внешней памяти соответственно. Для обращения к адресам внешней памяти используется указатель типа volatile uint8_t*, что гарантирует, что компилятор не будет выполнять оптимизации чтения/записи.

В функции main происходит запись последовательных значений в буфер buffer, который затем записывается во внешнюю память с помощью функции write_data. После этого, данные считываются из внешней памяти в тот же буфер buffer с помощью функции read_data. Наконец, значения из буфера выводятся на экран или передаются по UART.

Лабораторная работа № 14. Работа с АЦП/ЦАП МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы:

- Изучить основные принципы работы с АЦП и ЦАП микроконтроллеров STM32.
- Разработать алгоритмы и программные модули для работы с АЦП и ЦАП на микроконтроллере STM32.
- Проверить работу разработанных модулей на практике.

Теоретический материал

1. АЦП (аналогово-цифровой преобразователь) – это устройство, которое преобразует аналоговый сигнал в цифровой. Микроконтроллеры STM32 имеют встроенный АЦП, который позволяет считывать и обрабатывать аналоговые сигналы.
2. ЦАП (цифро-аналоговый преобразователь) – это устройство, которое преобразует цифровой сигнал в аналоговый. Микроконтроллеры STM32 также имеют встроенный ЦАП, который позволяет генерировать аналоговые сигналы.
3. Работа с АЦП:
   - Перед началом работы с АЦП необходимо настроить его параметры, такие как разрешение, источник опорного напряжения и т.д.
   - Для чтения аналогового сигнала с АЦП используется функция ADC_Read(), которая считывает значение напряжения с выбранного канала АЦП.
   - Для обработки полученного значения напряжения можно использовать различные алгоритмы, например, фильтрацию, усреднение и т.д.
4. Работа с ЦАП:
   - Для работы с ЦАП необходимо настроить его параметры, такие как разрешение, источник опорного напряжения и т.д.
   - Для генерации аналогового сигнала на ЦАП используется функция DAC_Write(), которая передает цифровое значение на ЦАП для преобразования в аналоговый сигнал.

Пример программы на языке C/C++:

```
#include "stm32f4xx.h"
// Функция настройки АЦП
void ADC_Init(void)
{
   // Настройка тактирования АЦП
   RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
   ADC1->CR1 &= ~ADC_CR1_SCAN;
```

```
    // Настройка каналов АЦП
    ADC1->SQR3 |= ADC_SQR3_SQ1_0; // выбор канала 1
    ADC1->SQR1 |= ADC_SQR1_L;     // количество каналов, которые
будут захвачены в процессе сканирования

    // Включение АЦП
    ADC1->CR2 |= ADC_CR2_ADON;
}
// Функция чтения значения с АЦП
uint16_t ADC_Read(void)
{
    // Запуск преобразования
    ADC1->CR2 |= ADC_CR2_SWSTART;

    // Ожидание окончания преобразования
    while (!(ADC1->SR & ADC_SR_EOC));

    // Чтение результата
    return ADC1->DR;
}
// Функция настройки ЦАП
void DAC_Init(void)
{
    // Настройка тактирования ЦАП
    RCC->APB1ENR |= RCC_APB1ENR_DACEN;

    // Включение ЦАП
    DAC->CR |= DAC_CR_EN1;
}
// Функция записи значения на ЦАП
void DAC_Write(uint16_t value)
{
    // Запись значения
    DAC->DHR12R1 = value;
}
int main(void)
{
    // Инициализация АЦП и ЦАП
    ADC_Init();
    DAC_Init();

    // Считывание значения с АЦП и запись на ЦАП
    uint16_t adcValue = ADC_Read();
    DAC_Write(adcValue);
```

```
    while (1) {}
}
```

Данная программа инициализирует АЦП и ЦАП микроконтроллера STM32, считывает значение с АЦП и записывает его на ЦАП. Пример можно доработать для выполнения других задач, таких как фильтрация входного сигнала, генерация дополнительных сигналов и т.д.

Лабораторная работа № 15. Работа с USB в МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить основные принципы работы с USB в микроконтроллерах STM32 и научиться применять типовые алгоритмы и программные модули.

Теоретический материал

USB (Universal Serial Bus) является интегральной частью современных компьютерных систем, позволяющей подключать различные устройства. Микроконтроллеры STM32 также поддерживают работу с USB и предоставляют типовые алгоритмы и программные модули для реализации этой функциональности.

Модуль USB в микроконтроллере STM32 представляет собой специализированный аппаратный блок, который позволяет подключать микроконтроллер к компьютеру или другому устройству через интерфейс USB. Он обеспечивает обмен данными между устройством и хостом (компьютером) посредством USB протокола.

Для работы с USB в микроконтроллерах STM32 используется библиотека USB Device Library (USBD). Она предоставляет набор функций и структур данных для управления USB коммуникацией. Некоторые из основных элементов USBD:

1. USB Core: ядро USB, предоставляющее базовые функции USB коммуникации.
2. USB Device Class: классы устройств, определяющие специфические протоколы и интерфейсы для работы с конкретными типами устройств. Например, USB CDC (Communication Device Class) для обмена данными с компьютером в режиме виртуального COM-порта.
3. USB Device Middleware: промежуточный слой, предоставляющий дополнительные функции и возможности, не входящие в базовый функционал USB Core.

Типовые алгоритмы и программные модули, используемые при работе с USB в микроконтроллерах STM32, включают в себя следующие этапы:

1. Инициализация USB модуля: настройка соответствующих регистров и параметров для работы USB.
2. Работа с прерываниями: обработка прерываний USB, связанных с передачей данных или другими событиями.
3. Определение класса устройства и интерфейсы: определение и настройка класса устройства для работы с конкретным типом устройства, а также настройка соответствующих USB интерфейсов.
4. Обработка данных: передача и прием данных через USB. Это включает в себя формирование и обработку пакетов данных, обмен командами и ответами между устройством и хостом.
5. Обработка событий: обработка различных событий, связанных с USB, таких как подключение или отключение устройства, окончание передачи данных и другие.

Пример лабораторной работы на языке C/C++:

```c
#include "stm32f4xx_hal.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_cdc.h"
USBD_HandleTypeDef hUsbDeviceFS;
void SystemClock_Config(void);
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  /* Инициализация USB */
  USBD_Init(&hUsbDeviceFS, &FS_Desc, DEVICE_FS);
  USBD_RegisterClass(&hUsbDeviceFS, &USBD_CDC);
  USBD_CDC_RegisterInterface(&hUsbDeviceFS,
&USBD_Interface_fops_FS);
  USBD_Start(&hUsbDeviceFS);
  while (1)
  {
    /* Ваш код обработки данных из USB */
  }
}
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  /** Конфигурация системы PLL */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

/** Конфигурация системы с частотой с целью достижения значения системного тактирования.
*/
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
{ Error_Handler();
}
}
```

Этот пример демонстрирует базовую структуру программы для работы с USB в микроконтроллерах STM32. Он инициализирует USB модуль, настраивает прерывания, регистрирует класс устройства и интерфейсы, а затем ожидает обработки данных через USB.
```

Лабораторная работа № 16. Работа с высокоуровневыми стеками в МК на высокоуровневом языке (C/C++). Типовые алгоритмы и программные модули.

Цель работы: изучить основные принципы работы с высокоуровневыми стеками в микроконтроллерах STM32 и научиться применять типовые алгоритмы и программные модули.

Теоретический материал

Микроконтроллеры STM32 предоставляют многофункциональные стеки программного обеспечения, которые позволяют разработчикам быстро и эффективно создавать приложения, основанные на основном ядре языка программирования C/C++. Стеки обычно включают в себя библиотеки функций, драйверы устройств, операционные системы в реальном времени (RTOS) и другие программные модули.

Высокоуровневые стеки представляют собой набор программных модулей и алгоритмов для решения типичных задач, таких как обработка сигналов, сетевое взаимодействие, мультимедиа и т. д. Кроме того, эти стеки предоставляют удобные интерфейсы для работы с внешними устройствами, такими как дисплеи, сенсоры, клавиатуры и другие периферийные устройства.

Примеры высокоуровневых стеков для микроконтроллеров STM32:

1. CMSIS (Cortex Microcontroller Software Interface Standard): CMSIS представляет собой стандартные интерфейсы и библиотеки функций для работы с ядром Cortex-M, включая обработку прерываний, управление энергопотреблением, доступ к периферийным устройствам и т. д.

2. HAL (Hardware Abstraction Layer): HAL предоставляет высокоуровневый интерфейс для работы с периферийными устройствами микроконтроллера, обеспечивая абстракцию от аппаратной реализации. HAL позволяет разработчикам работать с периферийными устройствами через единый интерфейс, независимо от модели микроконтроллера.

3. LL (Low-Level): LL позволяет работать с периферийными устройствами на низком уровне, предоставляя прямой доступ к регистрам и битовым полям микроконтроллера. LL может быть полезен для разработки программ с минимальным размером и максимальной производительностью.

Программный модуль простой программы на языке C/C++ для работы с UART-интерфейсом микроконтроллера STM32:

```c
#include "stm32f4xx_hal.h"
UART_HandleTypeDef huart2;
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
int main(void)
{
  HAL_Init();
```

```
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  char buffer[10] = "Hello STM32";
  while (1)
  {
    HAL_UART_Transmit(&huart2, (uint8_t*) buffer, sizeof(buffer),
HAL_MAX_DELAY);
    HAL_Delay(1000);
  }
}
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAG
E_SCALE1);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_0);
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}
static void MX_USART2_UART_Init(void)
{
```

```
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    HAL_UART_Init(&huart2);
}
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_ InitStruct;
  __HAL_RCC_GPIOA_CLK_ENABLE();
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
```

Лабораторная работа № 17. Создание алгоритма и программы для системы «Дисплей символьный» на основе МК.

Цель работы: изучить основные принципы работы системы «Дисплей символьный» на основе микроконтроллерах STM32.

Теоретический материал

Микроконтроллеры STM32 являются широко используемыми микроконтроллерами, которые часто применяются в системах ввода-вывода, таких как «Дисплей символьный». Они имеют высокую производительность и мощные возможности.

Система «Дисплей символьный» представляет собой устройство для отображения текстовой информации на жидкокристаллическом дисплее или другом подобном устройстве. Она включает в себя микроконтроллер, LCD-дисплей, клавиатуру для ввода информации и соответствующую программу для управления всей системой.

Для создания алгоритма и программы для системы «Дисплей символьный» на основе микроконтроллеров STM32 мы будем использовать язык программирования C/C++.

Программный модуль 1: Инициализация микроконтроллера. Программа должна инициализировать микроконтроллер STM32, настраивая его для работы с LCD-дисплеем и клавиатурой. Это включает в себя настройку GPIO-портов для управления выводами LCD-дисплея и чтения входов клавиатуры, настройку прерываний для обработки нажатий клавиш, настройку таймеров для синхронизации работы дисплея и записи данных на него.

Программный модуль 2: Управление дисплеем. Программа должна содержать функции для записи символов на дисплей, установки курсора в нужное положение и очистки экрана. Для этого используется командный набор LCD-дисплея, который может быть различным в зависимости от модели дисплея. Например, для установки курсора в заданную позицию можно использовать команду "set_cursor(x, y)", где x и y - координаты символа на дисплее.

Программный модуль 3: Управление клавиатурой. Программа должна содержать функции для определения нажатых клавиш на клавиатуре и соответствующей обработки этих нажатий. Например, можно создать функцию "get_key()", которая будет считывать состояние пинов клавиатуры и возвращать код нажатой клавиши.

Программный модуль 4: Основной цикл программы. В основном цикле программы происходит определение состояния клавиш, обработка нажатых клавиш, вывод информации на дисплей и выполнение других задач. Например, можно создать бесконечный цикл, в котором будет вызываться функция "get_key()" и в зависимости от полученного значения будет вызываться функция для записи символа на дисплей или очистки экрана.

Программный модуль 5: Тестирование системы. Программа должна содержать функции для тестирования работы системы «Дисплей

символьный». Например, можно создать функцию "test_display()", которая будет последовательно выводить на дисплей некоторую информацию, чтобы убедиться, что дисплей работает корректно.

Пример программы на языке C/C++:

```
#include <stdio.h>
#include <stdlib.h>
#include "stm32f0xx.h"  // заголовочный файл STM32
#include "lcd.h"  // заголовочный файл для работы с LCD-дисплеем
#include "keypad.h"  // заголовочный файл для работы с клавиатурой
int main() {
   /* Инициализация микроконтроллера */
   init_GPIO();  // инициализация GPIO-портов для взаимодействия с
LCD-дисплеем и клавиатурой
   init_interrupts();  // настройка прерываний для обработки нажатий
клавиш
   init_timers();  // настройка таймеров для синхронизации работы
дисплея и записи данных на него
   /* Основной цикл программы */
   while (1) {
      int key = get_key();  // считываем нажатую клавишу
      switch (key) {
         case KEY_1:
            write_lcd('1');  // записываем символ '1' на дисплей
            break;
         case KEY_2:
            write_lcd('2');  // записываем символ '2' на дисплей
            break;
         case KEY_3:
            clear_lcd();  // очищаем экран дисплея
            break;
      }
   }
   return 0;
}
```

В этом примере программа инициализирует микроконтроллер STM32, настраивает порты для работы с LCD-дисплеем и клавиатурой, настраивает прерывания для обработки нажатий клавиш и таймеры для синхронизации работы дисплея. В основном цикле программы она считывает нажатую клавишу, и в зависимости от нее записывает символ на дисплей или очищает его.

Лабораторная работа № 18. Создание алгоритма и программы для системы «Дисплей графический» на основе МК.

Цель:

- Ознакомление с основами программирования и алгоритмов для системы «Дисплей графический» на базе микроконтроллера.
- Практическое применение знаний по работе с графическим дисплеем и микроконтроллером.

Оборудование и программное обеспечение:

- Микроконтроллер (МК).
- Графический дисплей.
- Компьютер с установленной средой разработки для МК (например, Arduino IDE, Keil, MPLAB X и т.д.).

Теоретическая часть

Микроконтроллер (МК) представляет собой устройство, содержащее внутри себя центральный процессор (ЦП), оперативную память (ОЗУ) и периферийные устройства ввода-вывода, которые позволяют управлять различными внешними устройствами. Графический дисплей используется для отображения графической информации.

Алгоритм действий:

1. Ознакомиться с документацией по подключению графического дисплея к МК.
2. Создать базовую программу для МК, которая будет управлять графическим дисплеем.
3. Написать алгоритм отображения простейшей графики на дисплее (например, рисование линий, кругов, простых фигур и т.д.).
4. Загрузить программу на МК и проверить её работоспособность.

Практическая часть:

1. Подключить графический дисплей к МК согласно документации.
2. Создать программу, используя выбранную среду разработки для МК.
3. Написать алгоритм отображения простейших графических элементов (линий, кругов, прямоугольников и т.д.) на дисплее.
4. Прошить МК созданной программой.
5. Проверить работу алгоритма на графическом дисплее.

Лабораторная работа № 19. Создание алгоритма и программы для системы «Дисплей 7-сегментный» на основе МК.

Цель: Научиться создавать алгоритм и программу для управления 7-сегментным дисплеем с использованием микроконтроллера (МК).

Оборудование:

-Микроконтроллер (например, Arduino, Raspberry Pi или другой МК с возможностью управления цифровыми выводами).

-7-сегментный дисплей.

-Провода.

-Резисторы (по необходимости).

Ход работы:

Шаг 1: Определение алгоритма управления 7-сегментным дисплеем.

1.1 Изучите спецификацию 7-сегментного дисплея для определения соответствия между цифрами и сегментами.

1.2 Определите последовательность включения сегментов для отображения каждой цифры (0-9) на дисплее.

Шаг 2: Подключение 7-сегментного дисплея к микроконтроллеру.

2.1 Подключите аноды и катоды сегментов дисплея к цифровым выводам микроконтроллера с использованием резисторов для ограничения тока.

2.2 Подключите пины выбора цифры дисплея к другим цифровым выводам микроконтроллера.

Шаг 3: Написание программы для управления 7-сегментным дисплеем.

3.1 Используя выбранный язык программирования (например, C++, Python), напишите программу для отображения цифр на дисплее в соответствии с определенным алгоритмом.

3.2 Пропишите логику переключения сегментов для отображения каждой цифры, а также выбор цифры на дисплее.

Шаг 4: Тестирование программы.

4.1 Загрузите программу на микроконтроллер.

4.2 Проверьте правильность отображения цифр на 7-сегментном дисплее с помощью ввода тестовых данных в программу.

Шаг 5: Анализ результатов и доработка программы.

5.1 Оцените работоспособность программы и обнаруженные ошибки.

5.2 Внесите изменения в программу для улучшения работы с 7-сегментным дисплеем (если необходимо).

Шаг 6: Выводы.

6.1 Сделайте выводы о процессе создания алгоритма и программы для управления 7-сегментным дисплеем.

6.2 Оцените работу микроконтроллера в процессе управления дисплеем и возможности его применения для других задач.

6.3 Сформулируйте рекомендации по улучшению данной программы или разработке новых функций для дальнейших исследований.

Лабораторная работа № 20. Создание алгоритма и программы для системы «Кнопки управления» на основе МК.

Цель работы:

Научиться создавать алгоритмы и программы для системы управления на базе микроконтроллера с использованием кнопок в качестве управляющих элементов.

Оборудование:

1. Микроконтроллер STM32 или аналогичный;
2. Периферийные устройства для подключения и программирования микроконтроллера;
3. Набор кнопок;
4. Блок питания.

Задачи:

1. Изучить принципы работы микроконтроллера и его периферийных устройств;
2. Создать схему подключения кнопок к микроконтроллеру;
3. Разработать алгоритм работы программы для системы управления с использованием кнопок;
4. Реализовать программу на языке С для микроконтроллера;
5. Проверить работу системы управления с использованием кнопок.

Ход работы:

1. Изучение документации по выбранному микроконтроллеру и его периферийным устройствам для понимания особенностей работы и возможностей.
2. Сборка схемы подключения кнопок к микроконтроллеру с учетом необходимости использования внутренних резисторов подтяжки и защиты от дребезга контактов.
3. Разработка алгоритма работы программы, определяющего действия при нажатии на каждую из кнопок. Например, управление светодиодами, вывод сообщений на дисплей, изменение состояния других периферийных устройств и т.д.
4. Написание программы на языке С, используя интегрированную среду разработки, например, STM32CubeIDE или аналогичную.
5. Тестирование работы программы на микроконтроллере, проверка функционирования системы управления с использованием кнопок. При необходимости корректировка программы и перепроверка.

Результаты работы:

1. Разработанный алгоритм работы программы для системы управления на базе микроконтроллера;
2. Реализованная программа на языке С для микроконтроллера;
3. Проверенная система управления с использованием кнопок, работающая в соответствии с заданным алгоритмом.

Лабораторная работа № 21. Создание алгоритма и программы для системы «Матрица клавиатуры» на основе МК.

Цель: Разработать алгоритм и написать программу для системы "Матрица клавиатуры" на основе микроконтроллера (МК).

Необходимое оборудование:

- Микроконтроллер (например, Arduino или STM32)
- Матрица клавиатуры
- Провода
- Компьютер для программирования МК

Этапы выполнения:

1. Подготовка:
   - Подключить матрицу клавиатуры к МК с помощью проводов.
   - Запитать МК и матрицу клавиатуры.
2. Проектирование алгоритма:
   - Рассчитать размеры матрицы клавиатуры и ее подключение к МК.
   - Определить, какой вывод МК будет соединен с какими пинами матрицы клавиатуры (строки и столбцы).
3. Написание программы:
   - Написать программу на выбранном языке программирования для МК (например, C/C++ для Arduino или STM32CubeIDE для STM32).
   - Создать функции для считывания нажатых клавиш и их обработки.
   - Разработать алгоритм сканирования клавиатуры и обработки нажатий.
4. Тестирование и отладка:
   - Загрузить программу на МК.
   - Проверить работу матрицы клавиатуры путем нажатия различных клавиш и проверки вывода информации о нажатых клавишах.
5. Использование и дальнейшее развитие:
   - Протестировать программу в реальных условиях и внести исправления, если необходимо.
   - Предложить способы дальнейшего развития системы "Матрица клавиатуры", например, добавление функций автоматического распознавания комбинаций клавиш.

Результаты:

- Готовая программа для системы "Матрица клавиатуры" на основе МК, способная считывать и обрабатывать нажатия клавиш.

Лабораторная работа № 22. Создание алгоритма и программы для системы «Энкодер» на основе МК.

Цель работы:

-Разработать алгоритм и написать программу для системы «Энкодер» на основе микроконтроллера (МК).

-Ознакомиться с принципом работы энкодера и его применением в различных задачах.

Необходимые компоненты и оборудование:

-Микроконтроллер Arduino (или аналогичный).

-Энкодер (обычно также поставляется с кнопкой).

-Бреадборд или печатная плата для подключения компонентов.

-Провода и резисторы по необходимости.

-Компьютер с установленным ПО Arduino IDE.

Ход выполнения работы:

1)Подключение энкодера к микроконтроллеру:

-Подключите белый провод энкодера к пину 2 (или любому другому доступному цифровому пину) микроконтроллера.

-Подключите черный и красный провода энкодера к соответствующим пинам земли и питания на микроконтроллере.

2)Написание программы:

-Откройте Arduino IDE на компьютере и создайте новый проект.

-Напишите следующий код:

```
// Определение пинов
const int encoderPin = 2;

// Глобальные переменные
volatile int encoderPos = 0;
volatile int lastIncrement = 0;

// Функция обработчика прерывания для энкодера
void handleEncoderInterrupt() {
  int increment = (digitalRead(encoderPin) ? -1 : 1);

  if (increment != lastIncrement) {
    encoderPos += increment;
    lastIncrement = increment;
  }
}

// Инициализация
void setup() {
  pinMode(encoderPin, INPUT_PULLUP);

  attachInterrupt(digitalPinToInterrupt(encoderPin), handleEncoderInter

  Serial.begin(9600);
}


// Основной цикл
void loop() {
  Serial.println(encoderPos);
  delay(100);
}
```

3)Загрузка программы на микроконтроллер:

-Подключите микроконтроллер к компьютеру с помощью USB-кабеля.

-Выберите правильную плату и порт в меню "Инструменты" в Arduino IDE.

-Загрузите программу на микроконтроллер, нажав кнопку "Загрузить".

4)Тестирование и анализ результатов:

-Откройте монитор порта в Arduino IDE, выбрав скорость передачи 9600 бит/сек.

-Вращайте энкодер в одну или другую сторону.

-Обратите внимание на изменение значения, выводимого в монитор порта.

-Проанализируйте работу энкодера и сравните с ожидаемым поведением.

Лабораторная работа № 23. Создание алгоритма и программы для системы «Тачскрин» на основе МК.

Цель работы: разработать алгоритм и написать программу для работы с тачскрином на базе микроконтроллера.

Необходимые компоненты:

1. Микроконтроллер (например, Arduino Uno)
2. Тачскрин
3. Кабели для подключения тачскрина к микроконтроллеру

Шаги выполнения:

Шаг 1: Подготовительные работы

1. Подключите тачскрин к микроконтроллеру с помощью кабеля.
2. Установите необходимые библиотеки для работы с тачскрином, если они необходимы.

(Примечание: библиотеки могут варьироваться в зависимости от используемого микроконтроллера и тачскрина. Проверьте документацию для конкретных инструкций).

Шаг 2: Настройка оборудования

1. Проверьте, что тачскрин подключен корректно и включен.
2. Проверьте, что микроконтроллер правильно определяет тачскрин и может считывать данные с него.
3. Подключите микроконтроллер к компьютеру с помощью USB-кабеля.

Шаг 3: Создание алгоритма

1. Определите функции, которые будут использоваться в программе. Это может включать функции для считывания координат касания, обработки входных данных и выполнения конкретных действий в зависимости от касания.
2. Нарисуйте блок-схему алгоритма работы программы. Укажите последовательность действий и условия, которые определяют какие действия должны быть выполнены в зависимости от касания тачскрина.
3. Разбейте алгоритм на подзадачи (функции) и определите входные и выходные данные для каждой функции.

Шаг 4: Написание программы

1. Откройте среду программирования для выбранного микроконтроллера.
2. Создайте новый проект и добавьте необходимые библиотеки.
3. Напишите код программы, используя созданный алгоритм и подзадачи.
4. Проверьте программу на наличие ошибок и выполните ее отладку при необходимости.

Шаг 5: Тестирование программы

1. Загрузите программу на микроконтроллер.
2. Проверьте, что программа корректно работает с тачскрином. Проверьте функциональность каждого элемента алгоритма.

3. Проведите ряд тестов, чтобы убедиться, что программа работает стабильно и надежно.

Шаг 6: Документация

1. Напишите отчет о проведенной работе, описывающий созданный алгоритм и программу.

2. Укажите все необходимые детали, включая листинг программы, схему подключения и результаты тестирования.

Лабораторная работа № 24-25. Создание алгоритма и программы для системы «Мультиметр» на основе МК.

Цели:

1. Познакомиться с понятием мультиметра и его использованием в электронике.
2. Ознакомиться с микроконтроллером и его возможностями.
3. Создать алгоритм и написать программу для системы "Мультиметр" на основе микроконтроллера.

Оборудование:

1. Микроконтроллер (например, Arduino)
2. Макетная плата
3. Электронные компоненты (резисторы, конденсаторы, диоды и т.д.)
4. Компьютер с установленной средой разработки Arduino IDE

Ход работы:

1. Ознакомление с мультиметром

- Проведите исследование мультиметра и его функций.
- Изучите основные параметры, которые можно измерить с помощью мультиметра, такие как напряжение, сопротивление и ток.
- Проанализируйте возможности микроконтроллера для реализации функций мультиметра.

2. Ознакомление с микроконтроллером

- Изучите возможности выбранного вами микроконтроллера. Определите его характеристики и функциональность.
- Установите среду разработки Arduino IDE на ваш компьютер, если она еще не установлена.

3. Составление алгоритма

- Определите необходимые функции мультиметра, такие как измерение напряжения, сопротивления и тока.
- Разработайте алгоритм для каждой функции.
- Рассмотрите возможности обработки и вывода результатов измерений на дисплей или через последовательный порт компьютера.

4. Создание программы

- Создайте новый проект в среде разработки Arduino IDE.
- Напишите программу на языке Arduino, реализующую алгоритм, разработанный в предыдущем пункте.
- Скомпилируйте и загрузите программу на микроконтроллер.

5. Тестирование программы

- Подключите необходимые электронные компоненты к микроконтроллеру в соответствии с разработанным алгоритмом и программой.
- Проверьте работу мультиметра, выполнив измерения напряжения, сопротивления и тока с использованием созданной программы.

- Проверьте корректность результатов измерений. Если результаты не соответствуют ожидаемым, отладьте программу и/или проверьте правильность подключения компонентов.

6. Составление отчета

- Запишите результаты проведенного тестирования и полученные результаты измерений.
- Проанализируйте проблемы, с которыми вы столкнулись во время работы и возможные пути их решения.
- Сформулируйте выводы о работе мультиметра на основе микроконтроллера и его возможностях.

Важно: Во время выполнения работы обязательно соблюдайте правила безопасности при работе с электронными компонентами и измерительными приборами.

Лабораторная работа № 26. Создание алгоритма и программы для системы «UART с PC» на основе МК.

Цель:

Научиться создавать алгоритм и программу для обмена данными между микроконтроллером (МК) и персональным компьютером (PC) посредством Universal Asynchronous Receiver-Transmitter (UART).

Необходимое оборудование:

- Микроконтроллер с UART-интерфейсом
- Персональный компьютер
- Кабель для подключения микроконтроллера к ПК (например, USB-UART)
- Программное обеспечение для загрузки программы на микроконтроллер и для работы с терминалом на ПК (например, Arduino IDE)
- Провода для подключения микроконтроллера к другим устройствам

Шаги выполнения:

1. Подготовка микроконтроллера:

- Подключитесь к микроконтроллеру с помощью программного обеспечения для загрузки программы.
- Создайте новый проект и выберите соответствующую платформу с UART-интерфейсом.
- Настройте параметры UART-интерфейса (скорость передачи, битность, контроль четности и т.д.) в программе для микроконтроллера.

2. Подключение микроконтроллера к ПК:

- Подключите один конец кабеля USB-UART к порту USB на ПК и другой конец к соответствующему порту на микроконтроллере.
- Убедитесь, что соединение установлено правильно.

3. Написание программы для микроконтроллера:

- Определите необходимые прерывания и настройки UART-интерфейса в программе для микроконтроллера.
- Напишите программный код для передачи и приема данных через UART-интерфейс.
- Программа должна быть способна отправлять данные с микроконтроллера на ПК и получать данные с ПК на микроконтроллер.

4. Загрузка программы на микроконтроллер:

- Загрузите программу, написанную на предыдущем шаге, на микроконтроллер с помощью программного обеспечения для загрузки.

5. Подготовка ПК:

- Установите программное обеспечение для работы с терминалом на ПК (например, Arduino IDE).
- Откройте терминал и настройте его параметры соответствующим образом (скорость передачи, битность, контроль четности и т.д.).

6. Тестирование:

- Запустите программу на микроконтроллере и убедитесь, что она работает корректно.

- Введите данные в терминал на ПК и убедитесь, что они успешно передаются на микроконтроллер, и наоборот.

7. Расширение функциональности:

- Реализуйте дополнительные функции, такие как обработка ошибок, контроль четности, проверка целостности данных.

- Используйте различные команды и протоколы для обмена данными между микроконтроллером и ПК.

Выводы:

В результате выполнения лабораторной работы был разработан алгоритм и программа для обмена данными между МК и ПК посредством UART. Была получена практическая

Лабораторная работа № 27. Создание алгоритма и программы для системы «LAN с PC» на основе МК.

Цель работы:

- Познакомиться с основными принципами работы системы «LAN с PC» на основе микроконтроллера.
- Разработать алгоритм и программу для взаимодействия микроконтроллера с компьютером через локальную сеть.

Оборудование и программное обеспечение:

- Микроконтроллер (например, Arduino Uno)
- Компьютер с установленной программой Arduino IDE
- Ethernet-модуль (например, Ethernet Shield)

Этапы выполнения работы:

1. Изучение основ работы системы «LAN с PC» на основе микроконтроллера:
   - Понять принципы работы Ethernet-модуля и его взаимодействие с микроконтроллером.
   - Изучить основные команды и функции для работы с Ethernet-модулем в Arduino IDE.
2. Создание алгоритма работы системы «LAN с PC»:
   - Определить, какие данные будут передаваться между компьютером и микроконтроллером (например, считывание данных с датчиков, управление актуаторами и т. д.).
   - Разработать последовательность действий для передачи и приема данных между компьютером и микроконтроллером посредством Ethernet-модуля.
3. Написание программы для микроконтроллера:
   - Открыть программу Arduino IDE.
   - Создать новый проект и настроить его для работы с выбранным микроконтроллером и Ethernet-модулем.
   - Написать программу, реализующую разработанный алгоритм работы системы «LAN с PC».
   - Скомпилировать и загрузить программу на микроконтроллер.
4. Проверка работоспособности системы:
   - Подключить Ethernet-модуль к микроконтроллеру.
   - Подключить микроконтроллер к компьютеру через USB-порт.
   - Запустить программу на микроконтроллере и убедиться, что данные успешно передаются и принимаются между компьютером и микроконтроллером.
5. Оформление отчета:
   - Описать введение в тему работы.
   - Привести блок-схему алгоритма работы системы «LAN с PC».
   - Предоставить исходный код программы для микроконтроллера.

- Привести результаты проверки работоспособности системы, включая соответствующие скриншоты и комментарии.
- Сделать выводы о выполненной работе и ее практической значимости.

Лабораторная работа № 28. Создание алгоритма и программы для системы «CAN» на основе МК.

Цель работы: Изучение протокола CAN и разработка программного обеспечения для управления системой «CAN» на базе микроконтроллера.

Оборудование и компоненты:
- Микроконтроллер с поддержкой протокола CAN
- CAN-трансивер
- Разъемы и соединительные элементы для подключения системы CAN
- Компьютер для программирования и тестирования

Этапы работы:

1. Изучение протокола CAN:
   - Изучите основные принципы работы протокола CAN.
   - Подробно ознакомьтесь с форматом кадра и способами передачи сообщений.
   - Изучите особенности обмена данными между устройствами в системе CAN.
2. Подключение системы CAN:
   - Подключите микроконтроллер к компьютеру с помощью разъемов и соединительных элементов.
   - Подключите CAN-трансивер к микроконтроллеру, используя соответствующие порты и пины.

3. Разработка алгоритма:
   - На основе изученного протокола CAN разработайте алгоритм обмена данными между микроконтроллером и другими устройствами в системе.
   - Определите типы сообщений, которые должна поддерживать разрабатываемая система.
   - Спланируйте последовательность выполнения команд и обработки принятых сообщений.
4. Реализация программы:
   - Используйте выбранный вами язык программирования для разработки программы для микроконтроллера.
   - Соответствующим образом настройте CAN-интерфейс микроконтроллера.
   - Реализуйте алгоритм, разработанный на предыдущем этапе, в виде программного кода.
   - Протестируйте программу на микроконтроллере, используя специально созданные тестовые данные и сообщения.
5. Тестирование и отладка:
   - Проверьте функциональность программы на микроконтроллере.

- Проверьте правильность обмена данными между микроконтроллером и другими устройствами в системе CAN.

- Отладьте программу и внесите необходимые изменения для корректного функционирования системы.

6. Анализ результатов и выводы:

- Проанализируйте результаты тестирования и проверки функциональности системы CAN.

- Сделайте выводы о разработанной программе и полученных результатах.

- Оцените эффективность и перспективы применения системы CAN на основе микроконтроллера.

7. Оформление отчета:

- Создайте отчет о проведенной лабораторной работе, включающий описание работы, алгоритм и программный код, результаты тестирования и выводы.

- При необходимости, включите схемы подключения и фотографии используемых компонентов и оборудования.

- Укажите использованные источники информации и литературу.

Лабораторная работа № 29. Создание алгоритма и программы для системы «Электропривод» на основе МК.

Цель работы: ознакомиться с принципами работы и программированием электропривода на основе микроконтроллера.

Необходимые материалы и оборудование:
- Микроконтроллер Arduino (или другой аналогичный)
- Электродвигатель
- Постоянное напряжение и источник питания
- Платформа для монтажа и соединения компонентов
- Провода и соединительные элементы

Ход работы:

Шаг 1: Подготовка компонентов

- Соедините микроконтроллер Arduino с источником питания и компьютером с помощью соединительных кабелей.
- Подключите электродвигатель к микроконтроллеру с помощью проводов. Убедитесь, что все соединения надежно закреплены.
- Установите программное обеспечение Arduino IDE на компьютер, если оно еще не было установлено.

Шаг 2: Написание алгоритма для системы электропривода

- Определите требования к системе электропривода. Например, задайте необходимый угол поворота или скорость вращения.
- Разработайте алгоритм управления электроприводом, основываясь на требованиях. Это может быть простой цикл с условием, который будет управлять работой электродвигателя в зависимости от внешних факторов (кнопки, датчиков и т.д.).
- Задайте параметры работы электродвигателя, такие как напряжение питания и другие необходимые настройки.

Шаг 3: Программирование микроконтроллера

- Запустите Arduino IDE на компьютере.
- Создайте новый проект и откройте в нем файл с алгоритмом управления электродвигателем.
- Импортируйте необходимые библиотеки (если требуется) и определите функции для управления пинами микроконтроллера и электродвигателем.
- Напишите код, реализующий разработанный алгоритм управления.
- Проверьте код на наличие ошибок и выполните компиляцию.
- Загрузите скомпилированную программу на микроконтроллер с помощью кабеля, подключенного к компьютеру.

Шаг 4: Тестирование работы системы электропривода

- Подключите микроконтроллер Arduino к источнику питания.
- Запустите программу на микроконтроллере.

- При необходимости внесите корректировки в программу и повторите шаги 3 и 4 до достижения требуемых результатов.

- Проверьте работу электродвигателя, убедившись в правильности его управления в соответствии с разработанным алгоритмом.

Шаг 5: Анализ и заключение

- Оцените эффективность разработанной системы управления электроприводом на основе микроконтроллера.

Лабораторная работа № 30. Создание алгоритма и программы для системы «Нагреватель» на основе МК.

Цель работы:

- Создание алгоритма и программы для автоматизированной системы управления нагревателем на базе микроконтроллера.

Необходимые материалы и оборудование:

- Микроконтроллер (например, Arduino UNO)
- Нагревательный элемент (нагрузка)
- Термодатчик или термопара (для измерения температуры)
- Дисплей для отображения информации (необязательно)
- Блок питания
- Провода для соединения компонентов

Шаги выполнения:

1. Подготовка схемы подключения:
   - Подключите питание к микроконтроллеру и нагревателю.
   - Подключите термодатчик к портам аналогового ввода микроконтроллера.
2. Написание алгоритма:
   - Определите границы температур, в которых должен работать нагреватель.
   - Разработайте алгоритм управления, включающий:
     - Инициализацию микроконтроллера и всех необходимых портов.
     - Определение текущей температуры с помощью термодатчика.
     - Сравнение текущей температуры с заданными границами.
     - Включение или выключение нагревателя в зависимости от сравнения.
     - Отображение текущей температуры и состояния нагревателя на дисплее (если доступен).
3. Написание программы:
   - Откройте среду разработки Arduino IDE.
   - Создайте новый проект и скопируйте алгоритм в файл проекта.
   - Реализуйте алгоритм с помощью языка Arduino (C++).
   - Скомпилируйте и загрузите программу на микроконтроллер.
4. Тестирование системы:
   - Подключите блок питания к нагревателю и микроконтроллеру.
   - Запустите программу на микроконтроллере.
   - Проверьте, что нагреватель включается и выключается в зависимости от текущей температуры.
   - Убедитесь, что температура на дисплее (если используется) соответствует реальной температуре.
5. Анализ результатов:

- Проанализируйте работу системы и оцените точность регулирования температуры.

- Если необходимо, внесите поправки в алгоритм или программу для улучшения работы системы.

Выводы:

В ходе выполнения лабораторной работы был создан алгоритм и программа для системы управления нагревателем на базе микроконтроллера. Была реализована автоматическая регулировка температуры в заданных пределах. Результаты работы системы были проанализированы, и были предложены пути улучшения работы системы. Реализованная система может быть использована в различных областях, где требуется точное и надежное управление нагревателем.

Лабораторная работа № 31. Создание алгоритма и программы для системы «Матобработка данных (DSP)» на основе МК.

Цель работы:Целью данной лабораторной работы является разработка алгоритма и программы для системы «Матобработка данных (DSP)» на основе микроконтроллера (МК).

Необходимые компоненты:
- Микроконтроллер
- Платформа разработки и отладки
- Компьютер со средой разработки

Ход работы:

Шаг 1: Подготовка к работе
- Подключите микроконтроллер к платформе разработки и отладки.
- Скачайте и установите среду разработки на компьютер, если она еще не установлена.

Шаг 2: Разработка алгоритма
- Определите задачу, которую требуется решить с помощью системы «Матобработка данных (DSP)».
- Разработайте алгоритм решения задачи. Разбейте его на отдельные шаги и опишите каждый шаг в виде псевдокода или блок-схемы.

Шаг 3: Разработка программы
- Откройте среду разработки и создайте новый проект.
- Напишите программу на языке программирования МК, основываясь на разработанном алгоритме.
- Скомпилируйте программу и загрузите ее на микроконтроллер.

Шаг 4: Тестирование
- Подключите необходимые внешние устройства или датчики к микроконтроллеру.
- Запустите программу на микроконтроллере и проверьте ее работоспособность.
- Если необходимо, внесите коррективы в алгоритм или программу и повторите тестирование.

Шаг 5: Анализ результатов
- Проанализируйте результаты тестирования и сравните их с ожидаемыми.
- Оцените эффективность разработанного алгоритма и программы.
- Запишите результаты анализа.

Шаг 6: Выводы
- Сформулируйте выводы по проведенной работе.
- Опишите полученные результаты и их значимость.
- Предложите возможные пути улучшения разработанного алгоритма или программы.

Важно помнить, что данная лабораторная работа предполагает разработку алгоритма и программы для конкретной задачи, связанной с матобработкой данных на основе микроконтроллера. Поэтому, содержание работы и шаги могут отличаться в зависимости от поставленной задачи и выбранного микроконтроллера.