1. **Тестируемое устройство**

Снитко Д.А. 250501

Для тестирования был взят курсовой проект по дисциплине “Схемотехника” – виджет мониторинга параметров окружающей среды.

Устройство должно быть способно на:

– измерение температуры;

– измерение влажности;

– измерение уровня шума;

– отображение полученных данных на устройство вывода;

– настройку параметров через органы управления;

– отображение "настроения" на основе полученных данных.

В состав устройства входят:

– датчик температуры и влажности DHT11;

– датчик уровня шума KY-037;

– OLED дисплей 0.96" 128x64;

– Arduino NANO V3.0 CH340;

– 3 кнопки управления.

# Требования к системе диагностирования

Питаясь от внешнего источника питания 5V через USB, устройство может функционировать постоянно, при условии, что температура окружающей среды находится в границах от 0 до +50 градусов Цельсия (ограничение датчика DHT11).

Работу системы диагностирования можно подразделить на отдельные этапы:

* физический, на котором происходит проверка проводов и физического подключения элементов устройства
* аппаратный, на котором проводится диагностика отдельных элементов устройства
* программный, целью которого является выявление неисправностей программного обеспечения устройства
* пользовательский (или прикладной), верхний уровень абстракции и последний этап диагностики, в котором тестируется все устройство целиком в реальных сценариях

# Аппаратный этап

* 1. **Диагностика датчика DHT11**

Данный датчик проверяется путем сравнения его показателей с эталоном (термометром и гигрометром). После чего датчик проверяется на пороговых значениях соответствующих параметров (0-50°C для температуры и 20-80% для влажности) и, сравнивая значения, можно заключить об исправности или неисправности DHT11.

# 3.2 Диагностика датчика шума KY-037

Для проверки датчика шума создаем различные уровни звука (тишина, щелчок, хлопок) и проверяем соответствующие показания на аналоговом выходе датчика. Также проверяем работу цифрового выхода, регулируя порог срабатывания с помощью потенциометра на датчике.

# 3.3 Диагностика OLED дисплея и кнопок

Тестирование OLED дисплея с использованием различных паттернов и проверка работы кнопок управления.

#include <Wire.h>

#include <Adafruit\_GFX.h>

#include <Adafruit\_SSD1306.h>

#define SCREEN\_WIDTH 128

#define SCREEN\_HEIGHT 64

#define OLED\_RESET -1

#define SCREEN\_ADDRESS 0x3C

Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET);

const int BUTTON\_PINS[] = {2, 3, 4}; // Пины для кнопок

void setup() {

Serial.begin(9600);

if(!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) {

Serial.println(F("SSD1306 allocation failed"));

for(;;);

}

for(int i = 0; i < 3; i++) {

pinMode(BUTTON\_PINS[i], INPUT\_PULLUP);

}

Serial.println(F("Starting OLED display test..."));

testOLEDDisplay();

}

void loop() {

checkButtons();

}

void testOLEDDisplay() {

// Прогон 1: Плавное заполнение сверху вниз и обратно

Serial.println(F("Pass 1: Smooth fill top to bottom and back"));

smoothFillTest();

// Прогон 2: Шахматный паттерн сверху вниз

Serial.println(F("Pass 2: Checkerboard pattern top to bottom"));

checkerboardTest(true);

// Прогон 3: Инвертированный шахматный паттерн

Serial.println(F("Pass 3: Inverted checkerboard pattern"));

checkerboardTest(false);

Serial.println(F("OLED display test completed"));

Serial.println(F("Press buttons to test..."));

}

void smoothFillTest() {

// Заполнение сверху вниз

for(int16\_t y=0; y<SCREEN\_HEIGHT; y++) {

for(int16\_t x=0; x<SCREEN\_WIDTH; x++) {

display.drawPixel(x, y, SSD1306\_WHITE);

}

display.display();

delay(50); // Небольшая задержка для плавности

}

delay(1000);

// Заполнение снизу вверх (очистка)

for(int16\_t y=SCREEN\_HEIGHT-1; y>=0; y--) {

for(int16\_t x=0; x<SCREEN\_WIDTH; x++) {

display.drawPixel(x, y, SSD1306\_BLACK);

}

display.display();

delay(50); // Небольшая задержка для плавности

}

delay(1000);

}

void checkerboardTest(bool startWhite) {

for(int16\_t y=0; y<SCREEN\_HEIGHT; y++) {

for(int16\_t x=0; x<SCREEN\_WIDTH; x++) {

bool isWhite = ((x + y) % 2 == 0) ^ !startWhite;

display.drawPixel(x, y, isWhite ? SSD1306\_WHITE : SSD1306\_BLACK);

}

display.display(); // Обновляем дисплей после каждой строки

checkRow(y, startWhite);

}

delay(2000);

}

void checkRow(int16\_t y, bool startWhite) {

for(int16\_t x=0; x<SCREEN\_WIDTH; x++) {

bool shouldBeWhite = ((x + y) % 2 == 0) ^ !startWhite;

uint16\_t expectedColor = shouldBeWhite ? SSD1306\_WHITE : SSD1306\_BLACK;

if(display.getPixel(x, y) != expectedColor) {

Serial.print(F("Checkerboard error at ("));

Serial.print(x);

Serial.print(F(","));

Serial.print(y);

Serial.print(F("): Expected "));

Serial.println(shouldBeWhite ? F("WHITE") : F("BLACK"));

}

}

}

void checkButtons() {

for(int i = 0; i < 3; i++) {

if(digitalRead(BUTTON\_PINS[i]) == LOW) {

display.clearDisplay();

display.setTextSize(2);

display.setTextColor(SSD1306\_WHITE);

display.setCursor(0,0);

display.print(F("Button "));

display.print(i+1);

display.println(F(" pressed"));

display.display();

delay(500); // Debounce

while(digitalRead(BUTTON\_PINS[i]) == LOW); // Ждем отпускания кнопки

}

}

Результаты тестирования:

Starting OLED display test...

Pass 1: Smooth fill top to bottom and back

Starting OLED display test...

Pass 1: Smooth fill top to bottom and back

Pass 2: Checkerboard pattern top to bottom

Pass 3: Inverted checkerboard pattern

OLED display test completed

Press buttons to test...

Button 1 pressed

Button 2 pressed

Button 3 pressed

OLED дисплей успешно прошел все тесты, включая отображение различных паттернов и текста. Все кнопки управления корректно обрабатываются, дребезг контактов отсутствует.

**3.4 Диагностика платы Arduino NANO**

Проверяем напряжение на выводах питания. Тестируем цифровые и аналоговые входы/выходы с помощью простых тестовых скетчей. Проверяем работу интерфейса I2C для связи с OLED дисплеем.

**3.5 Диагностика АЛУ**

Тестирование основных арифметических и логических операций, таких как сложение, вычитание, умножение, деление, а также побитовые операции (AND, OR, XOR, NOT) и операции сравнения (равенство, больше, меньше).

#include <Arduino.h>

// Прототипы функций АЛУ

int add(int a, int b) { return a + b; }

int subtract(int a, int b) { return a - b; }

int multiply(int a, int b) { return a \* b; }

int divide(int a, int b) { return b != 0 ? a / b : 0; }

int bitwiseAnd(int a, int b) { return a & b; }

int bitwiseOr(int a, int b) { return a | b; }

int bitwiseXor(int a, int b) { return a ^ b; }

int bitwiseNot(int a) { return ~a; }

bool isEqual(int a, int b) { return a == b; }

bool isGreater(int a, int b) { return a > b; }

bool isLess(int a, int b) { return a < b; }

// Функция для выполнения теста

template <typename T>

void performTest(const char\* testName, T result, T expected) {

Serial.print(testName);

Serial.print(" -> ");

Serial.println(result == expected ? "Pass" : "Fail");

}

// Функция setup для инициализации тестов

void setup() {

Serial.begin(9600);

delay(1000); // Задержка для инициализации Serial

// Арифметические операции

performTest("Addition", add(5, 3), 8);

performTest("Subtraction", subtract(5, 3), 2);

performTest("Multiplication", multiply(5, 3), 15);

performTest("Division", divide(6, 3), 2);

// Побитовые операции

performTest("Bitwise AND", bitwiseAnd(5, 3), 1);

performTest("Bitwise OR", bitwiseOr(5, 3), 7);

performTest("Bitwise XOR", bitwiseXor(5, 3), 6);

performTest("Bitwise NOT", bitwiseNot(5), ~5);

// Операции сравнения

performTest("Equal", isEqual(5, 5), true);

performTest("Greater", isGreater(5, 3), true);

performTest("Less", isLess(3, 5), true);

}

// Функция loop не используется в тестах

void loop() {

// Пусто

}

# Результаты тестирования:

# Addition -> Pass

# Subtraction -> Pass

# Multiplication -> Pass

# Division -> Pass

# Bitwise AND -> Pass

# Bitwise OR -> Pass

# Bitwise XOR -> Pass

# Bitwise NOT -> Pass

# Equal -> Pass

# Greater -> Pass

# Less -> Pass

# Все тесты для АЛУ прошли успешно, подтверждая корректность выполнения арифметических и логических операций.

**3.6 Тестирование оперативной памяти микропроцессора**

Для тестирования оперативной памяти микропроцессора был использован алгоритм MATS+ (Modified Algorithmic Test Sequence Plus). Этот алгоритм является разновидностью маршевого теста и позволяет выявить дефекты памяти путем последовательного выполнения операций записи и чтения. Код пытается выделить как можно больше доступной памяти с помощью malloc и затем выполняет тест MATS+ на выделенной памяти.

#include <Arduino.h>

#define MEMORY\_SIZE 256 // Размер тестируемой памяти

byte memory[MEMORY\_SIZE]; // Массив, представляющий память

// Функция для выполнения теста MATS+

void matsPlusTest() {

Serial.println("Starting MATS+ Test");

// Шаг 1: Запись 0 во все ячейки

for (int i = 0; i < MEMORY\_SIZE; i++) {

memory[i] = 0;

}

// Шаг 2: Чтение 0 и запись 1

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] != 0) {

Serial.print("Error at step 2, index ");

Serial.println(i);

}

memory[i] = 1;

}

// Шаг 3: Чтение 1 и запись 0

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] != 1) {

Serial.print("Error at step 3, index ");

Serial.println(i);

}

memory[i] = 0;

}

// Шаг 4: Чтение 0

for (int i = 0; i < MEMORY\_SIZE; i++) {

if (memory[i] != 0) {

Serial.print("Error at step 4, index ");

Serial.println(i);

}

}

Serial.println("MATS+ Test Completed");

}

void setup() {

Serial.begin(9600);

delay(1000); // Задержка для инициализации Serial

matsPlusTest();

}

void loop() {

// Пусто

}

# Результаты тестирования:

# Starting MATS+ Test

# MATS+ Test Completed

# 3.7 Тестирование оперативной памяти микропроцессора

# Для тестирования внешней энергонезависимой памяти EEPROM был использован алгоритм MATS+ (Modified Algorithmic Test Sequence Plus). Этот алгоритм позволяет выявить дефекты памяти путем последовательного выполнения операций записи и чтения.

# #include <EEPROM.h>

# void setup() {

# Serial.begin(9600);

# while (!Serial) {

# ; // Ожидание подключения Serial

# }

# Serial.println("Starting EEPROM MATS+ Test");

# const int eepromSize = EEPROM.length();

# // Шаг 1: Запись 0 во все ячейки

# for (int i = 0; i < eepromSize; i++) {

# EEPROM.write(i, 0);

# }

# // Шаг 2: Чтение 0 и запись 1

# for (int i = 0; i < eepromSize; i++) {

# byte value = EEPROM.read(i);

# if (value != 0) {

# Serial.print("Error at step 2, index ");

# Serial.println(i);

# }

# EEPROM.write(i, 1);

# }

# // Шаг 3: Чтение 1 и запись 0

# for (int i = 0; i < eepromSize; i++) {

# byte value = EEPROM.read(i);

# if (value != 1) {

# Serial.print("Error at step 3, index ");

# Serial.println(i);

# }

# EEPROM.write(i, 0);

# }

# // Шаг 4: Чтение 0

# for (int i = 0; i < eepromSize; i++) {

# byte value = EEPROM.read(i);

# if (value != 0) {

# Serial.print("Error at step 4, index ");

# Serial.println(i);

# }

# }

# Serial.println("EEPROM MATS+ Test Completed");

# }

# void loop() {

# // Пусто

# }

# Результаты тестирования:

# Starting EEPROM MATS+ Test

# EEPROM MATS+ Test Completed

# Тестирование внешней энергонезависимой памяти EEPROM с использованием алгоритма MATS+ было выполнено успешно. В ходе теста не было обнаружено ошибок, что свидетельствует о корректной работе памяти.

# 4 Программный этап

Диагностика программного обеспечения проводится на протяжении всего процесса разработки. Проверка корректности считывания данных с датчиков

Тестирование алгоритмов обработки данных. Проверка работы меню и настроек. Тестирование отображения информации на дисплее. Проверка обработки ошибок и граничных случаев

# 5 Прикладной этап

Диагностирование верхнего уровня абстракции связано с непосредственным использованием устройства в реальных условиях. Устройство тестируется в различных помещениях, при разных условиях окружающей среды. Проверяется удобство использования, точность измерений, стабильность работы в течение длительного времени.

# 6 Вывод

Система диагностирования, обеспечивает комплексную проверку работоспособности устройства мониторинга параметров окружающей среды. Поэтапный подход, начиная с физического уровня и заканчивая прикладным, позволяет всесторонне оценить состояние всех компонентов устройства.

Аппаратный этап включает диагностику каждого отдельного компонента, такого как датчики, дисплей и кнопки, что позволяет выявить аппаратные неисправности. Программный этап направлен на проверку корректности работы программного обеспечения, включая алгоритмы обработки данных и взаимодействие с пользователем. Прикладной этап тестирует устройство в реальных условиях эксплуатации, что позволяет оценить его стабильность и удобство использования.

Результаты тестирования показали, что устройство функционирует корректно, все компоненты работают в соответствии с заданными параметрами, а программное обеспечение надежно обрабатывает данные и управляет устройством.