Федеральное государственное автономное
образовательное учреждение высшего образования
«Научно-образовательная корпорация ИТМО»

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Отчёт по лабораторной работе №4

По дисциплине «Системы ввода-вывода» (семестр 6)

Студент:
Дениченко Александр P3312
Разинкин Александр P3307
Балин Артём P3312
Практик:
Табунщик Сергей Михайлович

Санкт-Петербург
2025 г.

Цель

Изучение протоколов передачи данных между устройствами. Познакомится с принципами обмена данными между устройствами, алгоритмами обмена и форматами передачи данных на примере интерфейсов I2C, SPI, 1-Wire.

1 Задачи

.

1. Написать программу для микроконтроллера Atmega328, принимающую и отправляющую пакеты по интерфейсу UART в соответствии с обозначенным форматом пакета. Драйвер UART должен быть реализован с использованием операций ввода/вывода в регистры аппаратного контроллера UART.

2. Контроллер должен принимать данные с ПК, проверять их на корректность и отправлять обратно корректные пакеты. Если пакет пришел с ошибкой, то он отбрасывается.

3. Контроллер должен раз в секунду передавать данные с датчика, указанного в варианте задания.

4. Написать клиентскую программу на ПК для приема и отправки пакетов к микроконтроллеру по интерфейсу UART, моделирующей как корректную отправку пакетов, так и случаи с ошибками: неправильная длина, отсутствие синхробайта, недостаточное количество данных.

5. Подключить микроконтроллер к ПК и протестировать работоспособность написанных программ

6. Снять осциллограмму передачи любого пакета по интерфейсу UART

7. Оформить отчет по работе в электронном формате

2 Вариант 2

Датчик BMP280, SPI температура и давление – Скорость UART 38400 – Четность odd parity – Кол-во стоповых бит 2

3 Выполнение

Листинг 1: sketch

```
#include <Wire.h>
#include <Adafruit_BMP280.h>

#define BMP_CS 10

Adafruit_BMP280 bmp(BMP_CS);

void UART_Init();
void UART_Transmit(char data);
void UART_SendString_P(const char* str);
void UART_SendInt(int num);
void UART_SendFloat(float num);

void setup() {
    UART_Init();
    delay(100);

    unsigned status;
    status = bmp.begin();
    if (!status) {
        UART_SendString_P(PSTR("Could not find BMP280 sensor!\r\n"));
        while(1);
    }

    bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,
                    Adafruit_BMP280::SAMPLING_X2,
```

```
                        Adafruit_BMP280::SAMPLING_X16,
                        Adafruit_BMP280::FILTER_X16,
                        Adafruit_BMP280::STANDBY_MS_500);
}

void loop() {
    UART_SendString_P(PSTR("Temperature = "));
    UART_SendFloat(bmp.readTemperature());
    UART_SendString_P(PSTR(" *C\r\n"));

    UART_SendString_P(PSTR("Pressure = "));
    UART_SendLong((long)bmp.readPressure());
    UART_SendString_P(PSTR(" Pa\r\n\r\n"));

    delay(2000);
}

void UART_Init() {
    uint16_t ubrr = F_CPU / 16 / 9600 - 1;
    UBRR0H = (uint8_t)(ubrr >> 8);
    UBRR0L = (uint8_t)ubrr;
    UCSR0B = (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void UART_Transmit(char data) {
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}

void UART_SendString_P(const char* str) {
    char c;
    while ((c = pgm_read_byte(str++))) {
        UART_Transmit(c);
    }
}

void UART_SendInt(int num) {
    if (num < 0) {
        UART_Transmit('-');
        num = -num;
    }
    char buffer[10];
    int i = 0;

    do {
        buffer[i++] = num % 10 + '0';
        num /= 10;
    } while (num > 0);

    while (i > 0) {
        UART_Transmit(buffer[--i]);
    }
}
void UART_SendLong(long num) {
    if (num < 0) {
```

```
83            UART_Transmit('-');
84            num = -num;
85        }
86        char buffer[12];
87        int i = 0;
88
89        do {
90            buffer[i++] = num % 10 + '0';
91            num /= 10;
92        } while (num > 0);
93
94        while (i > 0) {
95            UART_Transmit(buffer[--i]);
96        }
97    }
98
99    void UART_SendFloat(float num) {
100       if (num < 0) {
101           UART_Transmit('-');
102           num = -num;
103       }
104
105       long integerPart = (long)num;
106       UART_SendLong(integerPart);
107       UART_Transmit('.');
108
109       int decimalPart = (int)((num - integerPart) * 100 + 0.5);
110       if (decimalPart < 10) UART_Transmit('0');
111       UART_SendInt(decimalPart);
112   }
```

4 + client mode

Листинг 2: update ino

```
1   /**
2   * UART communication program for Atmega328
3   * - Receives packets with format: [SYNC(0x5A)][LENGTH][DATA][CHECKSUM]
4   * - Validates packets and echoes back valid ones
5   * - Sends sensor data every second
6   * - Uses odd parity and 2 stop bits
7   */
8
9   // Constants
10  #define F_CPU 16000000UL  // 16 MHz clock
11  #define BAUD 38400        // Baud rate
12  #define SYNC_BYTE 0x5A    // Sync byte for packet validation
13
14  // Calculated UBRR value for UART
15  #define UBRR_VAL ((F_CPU / (16UL * BAUD)) - 1)
16
17  // Timer constants for 1-second interval
18  #define TIMER1_PRESCALER 256
19  #define TIMER1_COMPARE_VALUE (F_CPU / TIMER1_PRESCALER) // For 1 second
20
```

```
// Buffer sizes
#define MAX_PACKET_SIZE 64
#define MAX_DATA_SIZE (MAX_PACKET_SIZE - 3) // Subtract sync, length, checksum

// Packet parsing states
typedef enum {
  STATE_WAITING_SYNC,
  STATE_READING_LENGTH,
  STATE_READING_DATA,
  STATE_READING_CHECKSUM
} PacketState;

// Global variables
volatile uint8_t rxBuffer[MAX_PACKET_SIZE]; // Buffer for incoming data
volatile uint8_t dataBuffer[MAX_DATA_SIZE]; // Buffer for packet data
volatile uint8_t dataLength = 0;            // Length of current data
volatile PacketState state = STATE_WAITING_SYNC;
volatile uint8_t bytesRead = 0;             // Bytes read in current state
volatile uint8_t sendSensorData = 0;        // Flag for sending sensor data

// Function prototypes
void uartInit(void);
void timerInit(void);
uint8_t calculateChecksum(uint8_t *data, uint8_t length);
void sendPacket(uint8_t *data, uint8_t length);
uint16_t readSensor(void);

void setup() {
  // Initialize UART with direct register access
  uartInit();

  // Initialize timer for 1-second sensor data transmission
  timerInit();

  // Initialize ADC for sensor readings
  // Enable ADC with prescaler 128 (16MHz/128 = 125kHz)
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
  ADMUX = (1 << REFS0); // Use AVCC as reference, ADC0 as input
}

void loop() {
  // Check if it's time to send sensor data
  if (sendSensorData) {
    sendSensorData = 0;

    // Read from sensor
    uint16_t sensorValue = readSensor();

    // Prepare packet with sensor data
    uint8_t sensorPacket[3]; // 2 bytes for the sensor value + 1 for packet type
    sensorPacket[0] = 0x01;  // Packet type: Sensor data
    sensorPacket[1] = (uint8_t)(sensorValue >> 8);  // High byte
    sensorPacket[2] = (uint8_t)(sensorValue & 0xFF); // Low byte

    // Send the packet
    sendPacket(sensorPacket, sizeof(sensorPacket));
```

```
77      }
78    }
79
80    // Initialize UART with direct register access
81    void uartInit(void) {
82      // Set baud rate
83      UBRR0H = (uint8_t)(UBRR_VAL >> 8);
84      UBRR0L = (uint8_t)UBRR_VAL;
85
86      // Enable transmitter and receiver, and receive interrupt
87      UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);
88
89      // Set frame format: 8 data bits, odd parity, 2 stop bits
90      UCSR0C = (1 << UCSZ01) | (1 << UCSZ00) | // 8-bit data
91               (1 << UPM01) | (1 << UPM00) |    // Odd parity
92               (1 << USBS0);                    // 2 stop bits
93    }
94
95    // Initialize timer for 1-second intervals
96    void timerInit(void) {
97      // Set timer1 to CTC mode
98      TCCR1A = 0;
99      TCCR1B = (1 << WGM12) | (1 << CS12); // CTC mode, prescaler 256
100
101     // Set compare value for 1 second
102     OCR1A = TIMER1_COMPARE_VALUE;
103
104     // Enable timer compare interrupt
105     TIMSK1 = (1 << OCIE1A);
106
107     // Enable global interrupts
108     sei();
109   }
110   // UART receive interrupt
111 ISR(USART_RX_vect) {
112     // Read received byte
113     uint8_t receivedByte = UDR0;
114
115     // Check for UART errors (frame error, parity error, data overrun)
116     uint8_t error = UCSR0A & ((1 << FE0) | (1 << UPE0) | (1 << DOR0));
117
118     if (error) {
119       // Reset state machine if there's an error
120       state = STATE_WAITING_SYNC;
121       return;
122     }
123
124     // Process based on current state
125     switch (state) {
126       case STATE_WAITING_SYNC:
127         if (receivedByte == SYNC_BYTE) {
128           rxBuffer[0] = SYNC_BYTE;
129           state = STATE_READING_LENGTH;
130         }
131         break;
132
```

```
133     case STATE_READING_LENGTH:
134       dataLength = receivedByte;
135       rxBuffer[1] = dataLength;
136 
137       if (dataLength > MAX_DATA_SIZE) {
138         // Invalid length, reset state machine
139         state = STATE_WAITING_SYNC;
140       } else {
141         bytesRead = 0;
142         state = STATE_READING_DATA;
143       }
144       break;
145 
146     case STATE_READING_DATA:
147       // Store data byte
148       dataBuffer[bytesRead] = receivedByte;
149       rxBuffer[2 + bytesRead] = receivedByte;
150       bytesRead++;
151 
152       if (bytesRead >= dataLength) {
153         state = STATE_READING_CHECKSUM;
154       }
155       break;
156 
157     case STATE_READING_CHECKSUM:
158       // Store and check checksum
159       uint8_t receivedChecksum = receivedByte;
160       uint8_t calculatedChecksum = calculateChecksum(dataBuffer, dataLength);
161 
162       if (receivedChecksum == calculatedChecksum) {
163         // Valid packet received, echo it back
164         sendPacket(dataBuffer, dataLength);
165       }
166 
167       // Reset for next packet
168       state = STATE_WAITING_SYNC;
169       break;
170   }
171 }
172 
173 // Timer1 compare interrupt for periodic sensor reading
174 ISR(TIMER1_COMPA_vect) {
175   sendSensorData = 1;
176 }
177 
178 // Calculate checksum (XOR of all data bytes)
179 uint8_t calculateChecksum(uint8_t *data, uint8_t length) {
180   uint8_t checksum = 0;
181   for (uint8_t i = 0; i < length; i++) {
182     checksum ^= data[i];
183   }
184   return checksum;
185 }
186 
187 // Send a packet with the standard format
188 void sendPacket(uint8_t *data, uint8_t length) {
```

```
  // Wait until transmit buffer is empty
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = SYNC_BYTE;

  // Wait until transmit buffer is empty
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = length;

  // Send data bytes
  for (uint8_t i = 0; i < length; i++) {
    // Wait until transmit buffer is empty
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data[i];
  }

  // Send checksum
  uint8_t checksum = calculateChecksum(data, length);
  while (!(UCSR0A & (1 << UDRE0)));
  UDR0 = checksum;
}

// Read sensor data from ADC
uint16_t readSensor(void) {
  // Start conversion
  ADCSRA |= (1 << ADSC);

  // Wait for conversion to complete
  while (ADCSRA & (1 << ADSC));

  // Return result
  return ADC;
}
```

Листинг 3: client

```
import serial
import time
import random
import struct

class UARTClient:
    # Standard packet format:
    # [SYNC_BYTE(1)] [LENGTH(1)] [DATA(n)] [CHECKSUM(1)]
    SYNC_BYTE = 0x5A

    def init(self, port='COM5', baudrate=38400):
        self.ser = serial.Serial(
            port=port,
            baudrate=baudrate,
            bytesize=serial.EIGHTBITS,
            parity=serial.PARITY_ODD,  # Odd parity
            stopbits=serial.STOPBITS_TWO,  # 2 stop bits
            timeout=1
        )
        if not self.ser.is_open:
            self.ser.open()
        print(f"Connected to {port} at {baudrate} baud, odd parity, 2 stop bits")
```

```
        time.sleep(2)  # Give time for Arduino to reset

    def del(self):
        if hasattr(self, 'ser') and self.ser.is_open:
            self.ser.close()
            print("Serial port closed")

    def calculate_checksum(self, data):
        """Calculate simple XOR checksum of all data bytes"""
        checksum = 0
        for byte in data:
            checksum ^= byte
        return checksum

    def send_packet(self, data, corrupt_type=None):
        """
        Send a packet with optional corruption for testing

        corrupt_type options:
        - None: Send normal, valid packet
        - 'no_sync': Omit the sync byte
        - 'wrong_length': Set incorrect length
        - 'wrong_checksum': Use incorrect checksum
        - 'short_data': Send less data than specified in length
        """
        # Prepare basic packet
        packet = bytearray()

        # Add sync byte (unless corrupting)
        if corrupt_type != 'no_sync':
            packet.append(self.SYNC_BYTE)
        else:
            packet.append(0x00)  # Invalid sync byte

        # Add length byte
        if corrupt_type == 'wrong_length':
            packet.append(len(data) + 2)  # Incorrect length
        else:
            packet.append(len(data))  # Correct length

        # Add data
        if corrupt_type == 'short_data':
            # Only add half the data
            packet.extend(data[:len(data)//2])
        else:
            packet.extend(data)

        # Add checksum
        if corrupt_type == 'wrong_checksum':
            packet.append((self.calculate_checksum(data) + 1) % 256)  # Incorrect checksum
        else:
            packet.append(self.calculate_checksum(data))  # Correct checksum

        # Send the packet
        self.ser.write(packet)

```

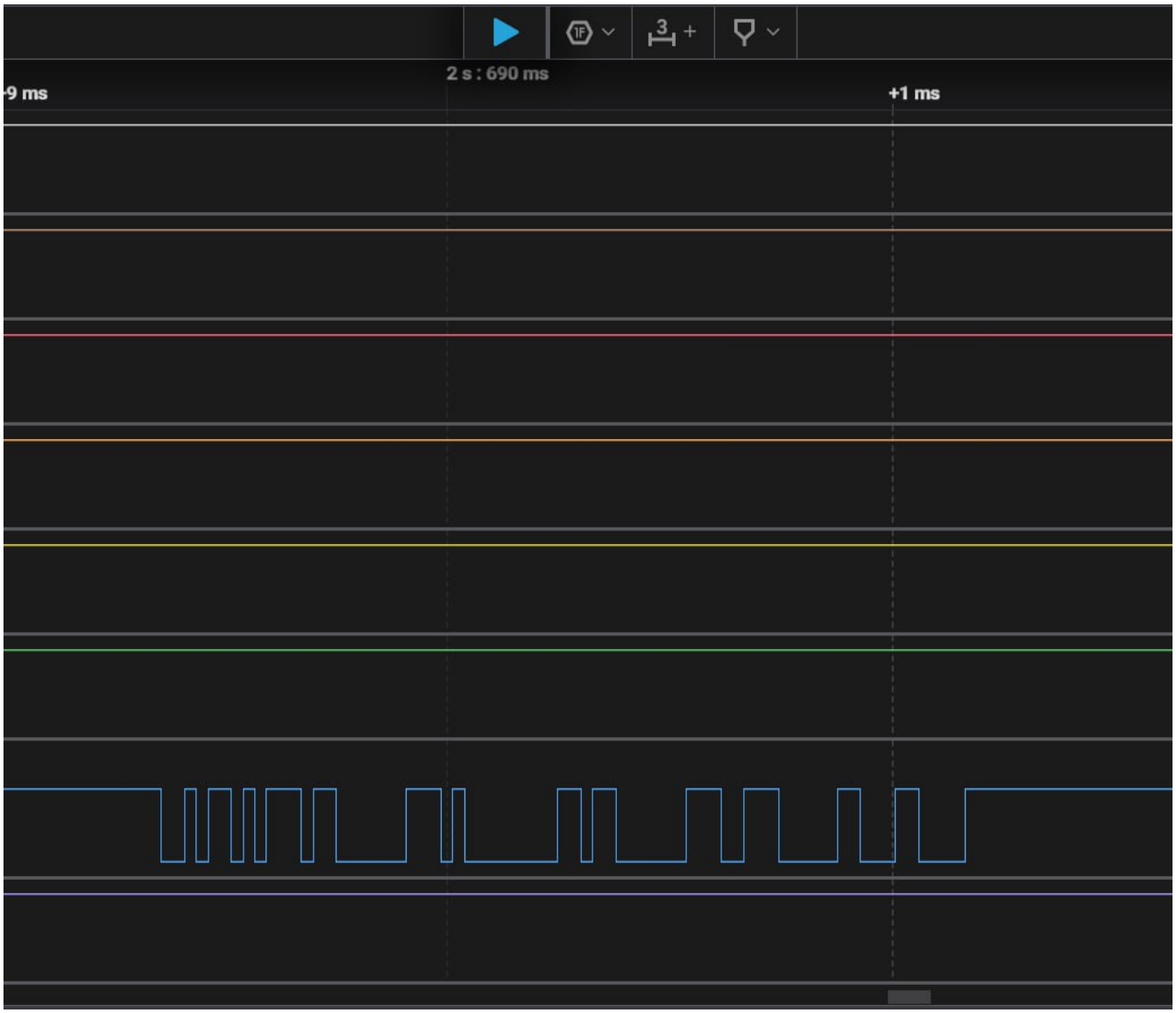
```
79            # Print what was sent
80            print(f"Sent: {' '.join([f'{b:02X}' for b in packet])}")
81
82            return len(packet)
83            def receive_packet(self, timeout=1.0):
84            """Receive and parse response packet with timeout"""
85            start_time = time.time()
86
87            # Wait for sync byte
88            while (time.time() - start_time) < timeout:
89                if self.ser.in_waiting > 0:
90                    sync = self.ser.read(1)
91                    if sync and sync[0] == self.SYNC_BYTE:
92                        break
93            else:
94                print("Timeout waiting for sync byte")
95                return None
96
97            # Read length
98            if self.ser.in_waiting < 1:
99                time.sleep(0.1)  # Small delay to ensure data arrival
100               if self.ser.in_waiting < 1:
101                   print("Failed to receive length byte")
102                   return None
103
104           length = self.ser.read(1)[0]
105
106           # Read data
107           data = bytearray()
108           if length > 0:
109               wait_time = 0
110               while len(data) < length and wait_time < timeout:
111                   if self.ser.in_waiting > 0:
112                       data.extend(self.ser.read(min(length - len(data),
                              self.ser.in_waiting)))
113                   else:
114                       time.sleep(0.01)
115                       wait_time += 0.01
116
117               if len(data) < length:
118                   print(f"Incomplete data: received {len(data)}/{length} bytes")
119                   return None
120
121           # Read checksum
122           if self.ser.in_waiting < 1:
123               time.sleep(0.1)
124               if self.ser.in_waiting < 1:
125                   print("Failed to receive checksum byte")
126                   return None
127
128           checksum = self.ser.read(1)[0]
129
130           # Verify checksum
131           calculated_checksum = self.calculate_checksum(data)
132           if checksum != calculated_checksum:
133               print(f"Checksum error: received {checksum:02X}, calculated
```

```
                {calculated_checksum:02X}")
            return None

        # Packet received successfully
        packet = bytearray([self.SYNC_BYTE, length]) + data + bytearray([checksum])
        print(f"Received: {' '.join([f'{b:02X}' for b in packet])}")

        return data

    def test_communication(self):
        """Run a series of tests including normal and corrupted packets"""
        tests = [
            ("Normal valid packet", None),
            ("Missing sync byte", "no_sync"),
            ("Incorrect length", "wrong_length"),
            ("Incorrect checksum", "wrong_checksum"),
            ("Incomplete data", "short_data")
        ]

        for test_name, corruption in tests:
            print(f"\n=== Testing: {test_name} ===")
            # Create random test data (3-10 bytes)
            data = bytearray([random.randint(0, 255) for _ in range(random.randint(3,
                10))])
            print(f"Test data: {' '.join([f'{b:02X}' for b in data])}")

            # Send the packet with specified corruption
            self.send_packet(data, corruption)
            time.sleep(0.5)

            # Try to receive response
            response = self.receive_packet()
            if response:
                print(f"Response received: {' '.join([f'{b:02X}' for b in response])}")
            else:
                print("No valid response received (as expected for corrupted packets)")

            time.sleep(1)  # Pause between tests
            def receive_packet(self, timeout=1.0):
            """Receive and parse response packet with timeout"""
            start_time = time.time()

            # Wait for sync byte
            while (time.time() - start_time) < timeout:
                if self.ser.in_waiting > 0:
                    sync = self.ser.read(1)
                    if sync and sync[0] == self.SYNC_BYTE:
                        break
            else:
                print("Timeout waiting for sync byte")
                return None

            # Read length
            if self.ser.in_waiting < 1:
                time.sleep(0.1)  # Small delay to ensure data arrival
                if self.ser.in_waiting < 1:
```

```
                print("Failed to receive length byte")
                return None

            length = self.ser.read(1)[0]

            # Read data
            data = bytearray()
            if length > 0:
                wait_time = 0
                while len(data) < length and wait_time < timeout:
                    if self.ser.in_waiting > 0:
                        data.extend(self.ser.read(min(length - len(data),
                            self.ser.in_waiting)))
                    else:
                        time.sleep(0.01)
                        wait_time += 0.01

                if len(data) < length:
                    print(f"Incomplete data: received {len(data)}/{length} bytes")
                    return None

            # Read checksum
            if self.ser.in_waiting < 1:
                time.sleep(0.1)
                if self.ser.in_waiting < 1:
                    print("Failed to receive checksum byte")
                    return None

            checksum = self.ser.read(1)[0]

            # Verify checksum
            calculated_checksum = self.calculate_checksum(data)
            if checksum != calculated_checksum:
                print(f"Checksum error: received {checksum:02X}, calculated
                    {calculated_checksum:02X}")
                return None

            # Packet received successfully
            packet = bytearray([self.SYNC_BYTE, length]) + data + bytearray([checksum])
            print(f"Received: {' '.join([f'{b:02X}' for b in packet])}")

            return data

    def test_communication(self):
        """Run a series of tests including normal and corrupted packets"""
        tests = [
            ("Normal valid packet", None),
            ("Missing sync byte", "no_sync"),
            ("Incorrect length", "wrong_length"),
            ("Incorrect checksum", "wrong_checksum"),
            ("Incomplete data", "short_data")
        ]

        for test_name, corruption in tests:
            print(f"\n=== Testing: {test_name} ===")
            # Create random test data (3-10 bytes)
```

```
242                 data = bytearray([random.randint(0, 255) for _ in range(random.randint(3,
                        10))])
243                 print(f"Test data: {' '.join([f'{b:02X}' for b in data])}")
244
245                 # Send the packet with specified corruption
246                 self.send_packet(data, corruption)
247                 time.sleep(0.5)
248
249                 # Try to receive response
250                 response = self.receive_packet()
251                 if response:
252                     print(f"Response received: {' '.join([f'{b:02X}' for b in response])}")
253                 else:
254                     print("No valid response received (as expected for corrupted packets)")
255
256                 time.sleep(1)  # Pause between tests'
257                 if name == "main":
258                 try:
259                     client = UARTClient()
260
261                     print("\n== Running communication tests with various corruptions ==")
262                     client.test_communication()
263
264                     print("\n== Starting regular communication mode ==")
265                     while True:
266                         # Send a normal packet with random data
267                         data = bytearray([random.randint(0, 255) for _ in
                                range(random.randint(3, 8))])
268                         print(f"\nSending data: {' '.join([f'{b:02X}' for b in data])}")
269                         client.send_packet(data)
270
271                         # Receive response from microcontroller
272                         response = client.receive_packet()
273                         if response:
274                             print(f"Response received: {' '.join([f'{b:02X}' for b in
                                    response])}")
275
276                         # Wait for the 1-second sensor data that should come from
                                microcontroller
277                         print("Waiting for periodic sensor data...")
278                         sensor_data = client.receive_packet(timeout=1.5)
279                         if sensor_data:
280                             print(f"Sensor data received: {' '.join([f'{b:02X}' for b in
                                    sensor_data])}")
281
282                         time.sleep(2)
283
284                 except KeyboardInterrupt:
285                     print("\nProgram terminated by user")
286                 except serial.SerialException as e:
287                     print(f"Serial error: {e}")
288                     print("Make sure the COM5 port is available and not in use by another
                            program")
```

5 Logic 2



6 Example

```
Adafruit_BMP280 bmp(BMP_CS); // аппаратный SPI

// Прототипы функций
void UART_Init();
void UART_Transmit(char data);
void UART_SendString_P(const char* str);
void UART_SendInt(int num);
void UART_SendFloat(float num);

void setup() {
    UART_Init();
    delay(100);
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Nano' on 'COM5')

```
Temperature = 26.84 *C
Pressure = 100722 Pa

Temperature = 26.86 *C
Pressure = 100723 Pa
```

7 Client out

```
Connected to COM5 at 38400 baud, odd parity, 2 stop bits

== Running communication tests with various corruptions ==

=== Testing: Normal valid packet ===
Test data: 55 00 C5 42 4F 83
Sent: 5A 06 55 00 C5 42 4F 83 1E
Received: 5A 06 55 00 C5 42 4F 83 1E
Response received: 55 00 C5 42 4F 83
```