Enhancing and Extending Modbus Communication Code for Dixell XR70CX Temperature Controller

M. Masuod

June 28, 2024

Abstract

This report discusses the original implementation of a Modbus communication code for the Dixell XR70CX temperature controller, outlines the enhancements made, and details how these improvements enable more versatile and generic functionality for various tasks and register readings.

1 Introduction

1.1 Objective

The original code aimed to interface with the Dixell XR70CX temperature controller using Modbus communication over RS-485 and Bluetooth. It also includes implementation for Neo-6M GPS module for positioning.

1.2 Initial Setup

The initial implementation included basic functionalities for sending Modbus requests, receiving responses, and processing data.

2 Original Code Explanation

2.1 Libraries and GPIO Pins

Included libraries:

- `Arduino.h`
- `SoftwareSerial.h`
- `ArduinoJson.h`
- `BluetoothSerial.h`
- `TinyGPSPlus.h`

Defined GPIO pins:

- `TX_PIN -> 15,3`
- `RX_PIN -> 27,4`
- `DE_RE_PIN -> 33`
- `LED_PIN -> 13`

2.2 Initialization

The `setup()` function initialized serial communications (`Serial`, `Soft_Serial`, `SerialBT`), set pin modes for RS-485 control and LED, and printed debug messages.

2.3 Modbus Communication

2.3.1 Modbus Request Structure

The original code used a predefined byte array for the Modbus request.

2.3.2 CRC Calculation

Implemented a function to calculate CRC for the Modbus request.

2.3.3 Sending and Receiving Data

Transmitted the Modbus request over RS-485 and read the response.

2.4 Data Handling

2.4.1 JSON Formatting

Converted the received Modbus response into a JSON object and sent it over Bluetooth.

3 Enhancements and Additions

3.1 Dynamic CRC Calculation

The improved code maintains dynamic CRC calculation to ensure data integrity. A function `ModbusCalcCRC` calculates CRC for any given Modbus request dynamically, enhancing flexibility.

3.2 Struct for Modbus Request

Introduced a `ModbusRequest` struct to store and manage Modbus request parameters, making the code more organized and easier to modify.

3.3 Versatility

3.3.1 Generic Request Handling

The code now supports different Modbus requests by allowing dynamic assignment of request parameters, making it adaptable to various tasks.

3.3.2 Reading Different Registers

Modified the code to enable reading different types of registers based on user input or configuration, making it more versatile.

3.3.3 GPS Module Addition

Added GPS location functionality to the ESP, ensuring geotagging of attached host for the server.

3.4 Additional Functionalities

3.4.1 Debugging and Logging

Added functions for better debugging and logging, such as `printModbusRequest` to print Modbus request details.

3.4.2 Error Handling

Improved error detection and handling, ensuring robust communication with the Modbus device.

3.4.3 User Input

Enabled the code to accept user input or configuration files to dynamically change Modbus request parameters.

4 Implementation

4.1 Code Walkthrough

4.1.1 ModbusRequest Struct

```
typedef struct {
    unsigned char SlaveAddress;
    unsigned char FunctionCode;
    unsigned char RegisterAddressMSB;
    unsigned char RegisterAddressLSB;
    unsigned char NumberOfRegistersMSB;
    unsigned char NumberOfRegistersLSB;
    unsigned char CRC_LSB;
    unsigned char CRC_MSB;
} ModbusRequest;
```

4.1.2 Dynamic CRC Calculation

```
void ModbusCalcCRC(unsigned char* Frame, unsigned char LenFrame) {
    unsigned char CntByte;
    unsigned char j;
    unsigned char bitVal;
    CRC = 0xFFFF;

    for (CntByte = 0; CntByte < LenFrame; CntByte++) {
        CRC ^= Frame[CntByte];

        for (j = 0; j < 8; j++) {
            bitVal = CRC & 0x0001;
            CRC >>= 1;

            if (bitVal) {
                CRC ^= MODBUS_GENERATOR;
            }
        }
    }
}
```

4.1.3 Generic Modbus Request Handling

```
void setup() {
    // Initialize serial communication
    Serial.begin(115200);
    Soft_Serial.begin(9600);
    SerialBT.begin("ESP32_Serial_BT");
    pinMode(DE_RE_PIN, OUTPUT);
    pinMode(LED_PIN, OUTPUT);
    digitalWrite(LED_PIN, HIGH);

    // Assign values to each field
    request.SlaveAddress = 0x01;
    request.FunctionCode = 0x03;
    request.RegisterAddressMSB = 0x01;
    request.RegisterAddressLSB = 0x01;
    request.NumberOfRegistersMSB = 0x00;
    request.NumberOfRegistersLSB = 0x05;
    request.CRC_LSB = 0x00;
    request.CRC_MSB = 0x00;

    unsigned char* requestBytes = (unsigned char*)&request;
    ModbusCalcCRC(requestBytes, 6);

    request.CRC_LSB = CRC & 0xFF;
    request.CRC_MSB = (CRC >> 8) & 0xFF;

    printModbusRequest(&request);
}
```

4.1.4 Sending and Receiving Data

```
void read_Modbus(ModbusRequest request) {
    digitalWrite(DE_RE_PIN, HIGH);
    unsigned char sendBuffer[] = {
        request.SlaveAddress,
        request.FunctionCode,
        request.RegisterAddressMSB,
        request.RegisterAddressLSB,
        request.NumberOfRegistersMSB,
        request.NumberOfRegistersLSB,
        request.CRC_LSB,
        request.CRC_MSB
    };
    Soft_Serial.write(sendBuffer, sizeof(sendBuffer));
    delay(100);

    digitalWrite(DE_RE_PIN, LOW);
    delay(100);
    no_Byte = 0;
    while (Soft_Serial.available() > 0) {
        byte temp = Soft_Serial.read();
        incomingByte[no_Byte] = temp;
        no_Byte++;
    }

    if (no_Byte > 0) {
        createJsonObject(incomingByte, no_Byte, request.SlaveAddress);
    } else {
        Serial.println("No response received");
    }
}
```

4.1.5 Creating JSON Object

```
void createJsonObject(byte* registers, int numRegisters, int slaveId) {
    const size_t capacity = JSON_OBJECT_SIZE(5);
    DynamicJsonDocument jsonDoc(capacity);

    float probe1Temp = registers[6] / 10.0;
    float probe2Temp = registers[8] / 10.0;
    float probe3Temp = registers[10] / 10.0;

    jsonDoc["slaveId"] = slaveId;
    jsonDoc["probe1Temp"] = probe1Temp;
    jsonDoc["probe2Temp"] = probe2Temp;
    jsonDoc["probe3Temp"] = probe3Temp;
    jsonDoc["timestamp"] = millis();

    serializeJson(jsonDoc, jsonString);

    Serial.println("JSON Object:");
    Serial.println(jsonString);
}
```

5 Overview of GPS Integration

In this project, GPS functionality is integrated into an ESP-based system using the TinyGPSPlus library. The system communicates with a GPS module via SoftwareSerial, allowing the ESP to receive and parse NMEA (National Marine Electronics Association) sentences from the GPS device. These sentences contain essential information such as geographic coordinates, date, and time.

5.1 Hardware Configuration

The GPS module is connected to the ESP through SoftwareSerial communication, with the module's RX pin connected to ESP's TX pin (pin 4), and the module's TX pin connected to ESP's RX pin (pin 3). This setup enables bidirectional serial communication between the ESP and the GPS module.

5.2 Software Implementation

5.2.1 Initialization and Setup

Listing 1: ESP Setup Code for GPS Integration

```
Serial.begin(115200);
ss.begin(GPSBaud); // Initialize SoftwareSerial for GPS communication
```

5.2.2 Initialization Operation

Listing 2: ESP Loop Code for GPS Data Reception

```
void loop() {
    while (ss.available() > 0) {
        if (gps.encode(ss.read())) {
            displayInfo(); // Display GPS information when a new valid
                sentence is received
        }
    }

    // Handle GPS detection error
    if (millis() > 5000 && gps.charsProcessed() < 10) {
        Serial.println(F("No GPS detected: check wiring."));
        while (true); // Stay in an infinite loop if GPS communication fails
    }
}
```

5.2.3 Data Parsing and Display

Listing 3: ESP Code for GPS Data Parsing and Display

```
void displayInfo() {
    // Display location, date, and time information
    Serial.print(F("Location: "));
    if (gps.location.isValid()) {
        Serial.print(gps.location.lat(), 6);
        Serial.print(F(","));
        Serial.print(gps.location.lng(), 6);
    } else {
        Serial.print(F("INVALID"));
    }

    Serial.print(F(" Date/Time: "));
    if (gps.date.isValid()) {
        Serial.print(gps.date.month());
        Serial.print(F("/"));
        Serial.print(gps.date.day());
        Serial.print(F("/"));
        Serial.print(gps.date.year());
    } else {
        Serial.print(F("INVALID"));
    }

    Serial.print(F(" "));
    if (gps.time.isValid()) {
        if (gps.time.hour() < 10) Serial.print(F("0"));
        Serial.print(gps.time.hour());
        Serial.print(F(":"));
        if (gps.time.minute() < 10) Serial.print(F("0"));
        Serial.print(gps.time.minute());
        Serial.print(F(":"));
        if (gps.time.second() < 10) Serial.print(F("0"));
        Serial.print(gps.time.second());
        Serial.print(F("."));
        if (gps.time.centisecond() < 10) Serial.print(F("0"));
        Serial.print(gps.time.centisecond());
    } else {
        Serial.print(F("INVALID"));
    }

    Serial.println();
}
```

6 Conclusion

The improvements made to the original code enhance its flexibility and versatility, allowing it to handle various Modbus requests dynamically. By introducing a `ModbusRequest` struct, improving CRC calculation, and enabling dynamic parameter assignment, the code can now be adapted for different tasks and register readings. These enhancements make the code more robust, user-friendly, and suitable for a wider range of applications. This implementation also demonstrates how to effectively integrate and utilize GPS functionality on an ESP using the TinyGPSPlus library and SoftwareSerial communication. It provides real-time location and time information, essential for various applications such as tracking, navigation, and geotagging.

Licensing Information

This code is developed for VirtuPharma and is to be used with the Dixell device, particularly the Dixell XR70CX temperature controller. Libraries used here are licensed as follows (scripts taken from each source file themselves):

- **Arduino.h**

  Main include file for the Arduino SDK
  ©2005-2013 Arduino Team. All rights reserved.

  This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

- **SoftwareSerial.h**

  Implementation of the Arduino software serial for ESP8266/ESP32.
  ©2015-2016 Peter Lerup. All rights reserved.
  ©2018-2019 Dirk O. Kaar. All rights reserved.

  This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

- **ArduinoJson.h**

  ArduinoJson - `https://arduinojson.org`
  ©2014-2024, Benoit BLANCHON
  MIT License

- **BluetoothSerial.h**

  ©2018 Evandro Luis Copercini

  Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at `http://www.apache.org/licenses/LICENSE-2.0`.

- **TinyGPSPlus.h**

  TinyGPSPlus - a small GPS library for Arduino providing universal NMEA parsing
  Copyright (C) 2008-2013 Mikal Hart
  All rights reserved.

  This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.