Development Report of Temperature Acquisition System Based on STM32

Zhang Yanpeng, Peng Yilin, Hu Yanbing
School of Electrical Engineering, Malaysia University of Science and Technology

Abstract

This report introduces the design and implementation of a temperature acquisition system based on the STM32 microcontroller. The system integrates temperature sensors, data acquisition, processing and monitoring components, and communication capabilities to achieve accurate and real-time temperature measurement. The collected data is displayed on a screen and can be retrieved and monitored by a computer using the MODBUS communication protocol, ensuring efficient and accurate temperature management.

1. Introduction

The objective of this project is to design a temperature acquisition system based on the STM32 microcontroller, which can achieve accurate real-time temperature measurement. The system integrates the DS18B20 temperature sensor, local data acquisition, processing and monitoring components, and communication capabilities. Temperature sensor data is read via the OneWire protocol and displayed on the GAPASS1602A screen via the I2C interface. Meanwhile, the system supports data retrieval and monitoring on the computer via MODBUS communication using USART2.

2. Software/Tools Setup

2.1 Integrated Development Environment (IDE)

This project uses Keil5 (µVision) as the integrated development environment (IDE). Install Keil5 and ensure it is updated to the latest version for optimal performance and functionality.

2.2 STM32CubeMX

STM32CubeMX is used to configure the STM32 microcontroller. Download and install STM32CubeMX, which is a graphical tool for configuring the STM32 microcontroller.

2.3 Toolchain

Ensure that the Arm Compiler is installed in Keil5 for compiling the STM32 firmware.

2.4 Drivers and Libraries

Download the required drivers and libraries from STMicroelectronics, particularly the libraries for the STM32F446 microcontroller and peripherals such as I2C, USART, and OneWire protocol.

3. Configuration Steps

3.1 Creating a Project with STM32CubeMX

1. Open STM32CubeMX and create a new project.
2. Select the STM32F446 microcontroller.
3. Configure the system clock according to the design requirements.

3.2 Peripheral Configuration

1. Enable and configure the I2C interface for the GAPASS1602A display.
2. Enable and configure USART2 for MODBUS communication.
3. Set up GPIO pins for the OneWire protocol to read data from the DS18B20 temperature sensor.

3.3 Generating Code

In the Project Settings, select Keil MDK-ARM V5 as the toolchain/IDE, generate the project code, and open the generated project in Keil5.

4. Firmware Development Steps

4.1 Configuring the Project in Keil5

Open the generated project file and configure the project settings, such as target device and compiler options.

4.2 Initialization Code

In `main.c`, initialize the system clock, GPIO, and configured peripherals.

4.3 OneWire Protocol Implementation

Implement or include the OneWire protocol code to read temperature data from the DS18B20 sensor.

4.4 Temperature Data Processing

Implement the logic to process the acquired temperature data, ensuring real-time data processing to maintain system accuracy.

4.5 Display and Communication

Write code to display the temperature data on the GAPASS1602A screen via the I2C interface and implement MODBUS communication using USART2 for data exchange with a computer.

4.6 Main Loop

In the main loop, continuously read the temperature data, process the data, and update the display, ensuring proper error handling and system stability.

5. Hardware Development Steps

5.1 Component Selection

Select the STM32F446 microcontroller, DS18B20 temperature sensor, GAPASS1602A display, and necessary communication interfaces.

5.2 Circuit Design

Design the circuit diagram, including the microcontroller, sensor, display, and communication modules. Ensure proper connections for power, ground, and signal lines.

5.3 PCB Layout

Use software such as Altium Designer or KiCad to design the PCB layout, paying special attention to signal line routing, especially for high-speed communication interfaces like I2C and USART.

5.4 Prototype Assembly

Assemble components on a breadboard or prototype board, verify connections, and ensure the system powers up correctly.

5.5 Testing and Debugging

Individually test each component to ensure they work properly, debug any issues using logic analyzers or oscilloscopes, and perform integration tests to ensure all components work seamlessly together.

5.6 Final Assembly

After testing, assemble the final hardware, ensuring secure connections and appropriate enclosures to protect the components.

6. Code Implementation

6.1 Initialization and Configuration

```
#include "main.h"
#include "usart.h"
#include "gpio.h"
#include <stdio.h>
#include "LCD1602A.h"
#include "DS18B20.h"

void SystemClock_Config(void);

void floatToStr(char* str, float num) {
   sprintf(str, "%.2f", num);  // Convert float to string with 2 decimal places
}
```

6.2 Main Function

```
int main(void)
{
   HAL_Init();
   SystemClock_Config();
   MX_GPIO_Init();
   MX_USART2_UART_Init();

   LCD_Init();
   LCD_Clear();
   LCD_SetCursor(0, 0);
   LCD_SendString("Hello");
   LCD_SetCursor(1, 0);
   LCD_SendString("World");
```

```
    float temperature;
    char tempStr[10];
    DS18B20_Init();

    if (DS18B20_Reset() == 1) {
        LCD_SetCursor(0, 0);
        LCD_SendString("Sensor OK");
    } else {
        while (1) {
            LCD_SetCursor(1, 0);
            LCD_SendString("Sensor Fail");
        }
    }

    while (1) {
        DS18B20_StartConversion();
        HAL_Delay(750);

        temperature = DS18B20_ReadTemperature();
        floatToStr(tempStr, temperature);

        LCD_Clear();
        LCD_SetCursor(0, 0);
        LCD_SendString("Temp: ");
        LCD_SendString(tempStr);
        LCD_SendString(" C");

        HAL_Delay(2000);
    }
}
```

6.3 System Clock Configuration

```
void SystemClock_Config(void) {
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

  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 = 8;
  RCC_OscInitStruct.PLL.PLLN = 72;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
    Error_Handler();
  }

  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_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
}
```

Conclusion

Through the above steps, we successfully implemented a temperature acquisition system based on the STM32 microcontroller. The system can accurately and in real-time measure the temperature, and display the temperature data on the screen. In addition, the system also realizes the function of communication with the computer via the MODBUS protocol, supporting remote temperature monitoring and management.