Sistema medidor de temperatura

Proyecto de microcontroladores

Alumno: Jorge Ángel Juárez Vázquez - 2213026247

Profesor: Luis Ángel Alarcón Ramos

Trimestre: 24 - I

Resumen

En este proyecto se desarrolló un sistema de medición de temperatura utilizando un Arduino UNO con sensores NTC y DS18B20. La temperatura promedio se muestra en un display de siete segmentos, y la frecuencia de medición es ajustable mediante un potenciómetro. Se integraron códigos en C++ y ensamblador para optimizar el rendimiento y se exploró el funcionamiento interno del microcontrolador ATmega328p, proporcionando una comprensión profunda tanto de alto como de bajo nivel en sistemas embebidos.

Descripción

El objetivo de esta práctica es diseñar y construir un dispositivo que mida la temperatura ambiente utilizando sensores y presente los resultados en un par de displays de 7 segmentos. Para cumplir con los requerimientos, se emplearán al menos dos tipos diferentes de sensores de temperatura, un potenciómetro, resistencias, un par de displays de 7 segmentos y un expansor de pines digitales con protocolo de comunicación I^2C.

El dispositivo debe operar de la siguiente manera:

a) Medición de la Temperatura a Intervalos Configurables: Al encender el dispositivo, éste debe iniciar la medición de la temperatura ambiente a intervalos regulares. El intervalo de tiempo entre cada medición, denotado como 'n', es configurable y se ajustará mediante un potenciómetro. Los valores posibles de 'n' variarán entre 500 milisegundos y 5 segundos.

b) Visualización de las Mediciones: Cada muestra tomada por los sensores de temperatura será mostrada en un par de displays de 7 segmentos. Los

displays deben ser capaces de representar temperaturas en un rango de 0°C a 99°C.

c) Funcionamiento Continuo: Una vez configurado el intervalo de medición mediante el potenciómetro, el dispositivo se mantendrá midiendo la temperatura y actualizando los displays a la cadencia indicada por 'n'.

Desarrollo

El sistema calcula el promedio de las temperaturas medidas por ambos sensores y muestra el valor en un par display de siete segmentos. Adicionalmente, el intervalo de tiempo entre mediciones puede ser ajustado mediante un potenciómetro, permitiendo un rango configurable de 500 ms a 5000 ms. El propósito del proyecto es proporcionar una forma precisa y personalizable de monitorear la temperatura en diferentes entornos.

1. Componentes y Conexiones

Hardware

- Arduino UNO
- Sensor de Temperatura NTC módulo KY-003
- Sensor de Temperatura DS18B20
- Potenciómetro
- 2 displays de Siete Segmentos de cátodo común
- Expansor I²C HW-171
- Resistencias y cables de conexión

Conexiones

- Sensor NTC: Conectado a un pin analógico del Arduino (A0)
- Sensor DS18B20: Conectado a un pin digital del Arduino (D2)
- Potenciómetro: Conectado a un pin analógico del Arduino (A1)
- Displays de Siete Segmentos: Conectado a pines digitales del Arduino (D3 - D9) para las unidades y al expansor I²C HW-171 (P0 - P6) para las decenas.

- Expansor I2C HW-171: Conectado a los pines I²C del Arduino (SDA y SCL) que corresponden a los pines analógicos A4 y A5 respectivamente.

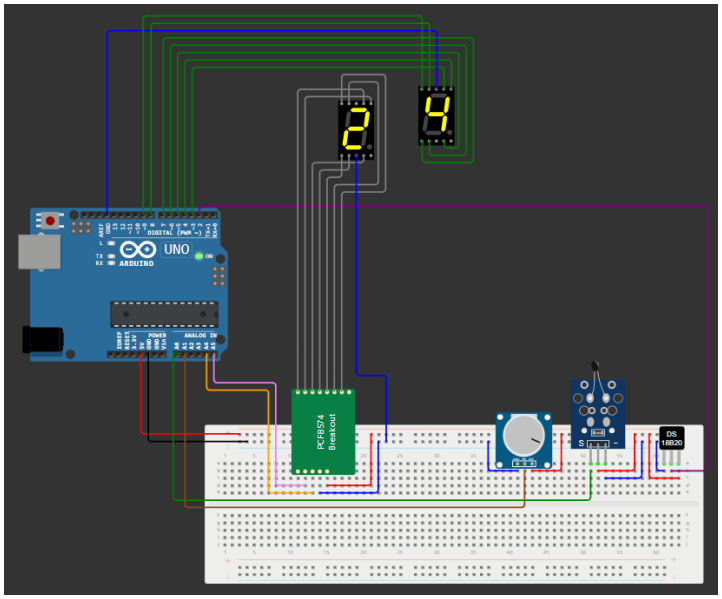

Imagen 1. Conexiones para el sistema medidor de temperatura.

2. Código fuente

El código fuente del proyecto se estructura en varias funciones que permiten la lectura de los sensores, el cálculo del promedio de las temperaturas, la lectura del valor del potenciómetro, y la visualización de la temperatura en el display de siete segmentos.

La función `loop()` es el núcleo del programa. En cada iteración, lee el intervalo de tiempo del potenciómetro, calcula las temperaturas de los sensores, determina el promedio, y actualiza el display de siete segmentos con las unidades y decenas de la temperatura promedio.

```
void loop() {
    interval = potentiometer_interval();
    Serial.print("Intervalo de tiempo entre mediciones: ");
    Serial.print(interval);
    Serial.println("ms");

    float ntc_temp = ntc_sensor();
    float ds18b20_temp = ds18b20_sensor();
    average_temp = average_temperature(ntc_temp, ds18b20_temp);
    Serial.print("Temperatura: ");
    Serial.print(average_temp);
    Serial.println(" °C");

    unitDisplayNumber(getUnits(average_temp));
    tensDisplayNumber(getTens(average_temp));

    delay(interval);
}
```

Imagen 2. Función loop del código fuente.

3. Integración de Ensamblador en el Código

Para optimizar ciertas partes del código, se integraron instrucciones en ensamblador en línea. A continuación, se presenta un ejemplo de cómo se implementó en ensamblador la función de para obtener la unidad de la temperatura para que se despliegue en el display de 7 segmentos.

```
int getUnits(float numero) {
    int entero = (int)numero;
    int unidades;

    asm volatile(
        "ldi r18, 10\n\t"      // Load immediate value 10 into register r18
        "movw r24, %1\n\t"     // Move the value of entero into r24:r25
        "1: subi r24, 10\n\t"  // Subtract 10 from r24
        "brcs 2f\n\t"          // If carry set, branch to label 2
        "rjmp 1b\n\t"          // Jump back to label 1
        "2: subi r24, -10\n\t" // Add 10 to r24 (undo last subtraction)
        "mov %0, r24\n\t"      // Move the final result (units) to the output variable
        : "=r"(unidades)       // Output operands
        : "r"(entero)          // Input operands
        : "r18", "r24", "r25"  // Clobbers
    );

    return unidades;
}
```

Imagen 3. Implementación de la función `getUnits()` en lenguaje ensamblador

Conclusión

El desarrollo del proyecto de medición de temperatura con Arduino UNO ha permitido una comprensión más profunda de los sistemas embebidos, destacando la importancia de los lenguajes de alto nivel como C++, que facilitan la abstracción y el desarrollo eficiente de aplicaciones. Sin embargo, al abordar este proyecto, también se ha puesto en valor el conocimiento de bajo nivel, específicamente mediante el uso del ensamblador en el microcontrolador ATmega328p.

Entender las entrañas de los sistemas, como las computadoras, microcontroladores y microprocesadores, es crucial para resolver problemas de mayor complejidad. Conocer detalles sobre el uso de la memoria, particiones, direcciones y tiempos de ejecución nos brinda la capacidad de optimizar el rendimiento en futuros desarrollos.

El reto principal fue comprender el grado de abstracción que implica el uso del ensamblador. Aunque no alcanzó un nivel de dificultad extremo, el entendimiento de las instrucciones y operaciones internas del ATmega328p requirió una dedicación considerable. Esta experiencia ha sido enriquecedora, proporcionando una perspectiva clara de cómo las operaciones de bajo nivel impactan el rendimiento y eficiencia de un sistema.

Implementación: jorge-jrzz/temperature_sensors | Simulación WOKWI