ENCS4380, INTERFACING TECHNIQUES

Arduino Temperature and Light Sensor Project Documentation

Students' names

Leen Aldeek	**id:**1212391	Section: 1
Mohammad Badha	**id:**1201862	Section: 2
Lina AbuFarha	**id:**1211968	Section: 2
Hajar Al Souqi	**id:** 1203257	Section: 2

Instructor’s name: Dr. Wasel Ghanem

Date :22/5/2025

Introduction

This document provides comprehensive documentation for an Arduino-based project that reads temperature and light level data from sensors and displays them on an LCD screen with custom icons. The project alternates between displaying temperature and light level readings every 2 seconds, using custom-created icons for visual representation. Notably, the implementation uses no external libraries, relying instead on direct hardware control.

Project Requirements

The project implements the following requirements: - Reads temperature from an LM35 sensor connected to analog pin A0 - Reads light level from an LDR (Light Dependent Resistor) connected to analog pin A1 - Displays custom icons for thermometer and light bulb - Alternates between temperature and light level display every 2 seconds - Uses no external libraries for implementation

Hardware Components and Connections

Components Used

The project utilizes several hardware components to achieve its functionality:

The Arduino Uno serves as the brain of the project, processing sensor readings and controlling the LCD display. Its versatile microcontroller architecture provides the necessary analog inputs for sensor readings and digital outputs for LCD control. The board's 5V power supply is sufficient for powering all the components in this project.

A 16x2 LCD display with blue backlight provides visual output for the sensor readings. This display features 16 columns and 2 rows of characters, allowing for clear presentation of both temperature and light level data along with their respective custom icons. The blue backlight ensures good visibility in various lighting conditions and provides an aesthetically pleasing appearance.

The DHT11 temperature sensor is a precision integrated-circuit temperature device that provides an analog output voltage proportional to the ambient temperature. Its linear output characteristic makes it ideal for this application, as it simplifies the conversion from voltage to temperature. The sensor's accuracy of ±0.5°C at room temperature ensures reliable temperature readings.

A Light Dependent Resistor (LDR) serves as the light sensor, changing its resistance based on the amount of light falling on it. When connected in a voltage divider configuration, it provides an analog voltage that varies with light intensity, allowing the Arduino to measure ambient light levels. The LDR's spectral response is similar to the human eye, making it suitable for measuring visible light.

A potentiometer is used to adjust the contrast of the LCD display, allowing for optimal visibility in different viewing conditions. This component provides a variable resistance that controls the voltage to the LCD's contrast pin.

The breadboard provides a platform for prototyping the circuit without soldering, allowing for easy component placement and connection modifications. Various colored jumper wires complete the setup, connecting the components according to the circuit design.

Connections

The LCD display is connected to the Arduino in 4-bit mode to save digital pins: - VSS (GND) → Arduino GND - VDD (Power) → Arduino 5V - V0 (Contrast) → Potentiometer middle pin - RS (Register Select) → Arduino digital pin (likely pin 12) - RW (Read/Write) → Arduino GND (for write-only operation) - E (Enable) → Arduino digital pin (likely pin 11) - D4, D5, D6, D7 → Arduino digital pins (likely pins 5, 4, 3, 2) - A (Backlight Anode) → Arduino 5V through a resistor - K (Backlight Cathode) → Arduino GND

The DHT11 temperature sensor has three pins connected as follows: - Left pin (VCC) → Arduino 5V - Middle pin (Output) → Arduino Analog pin A0 - Right pin (GND) → Arduino GND

The LDR is connected in a voltage divider configuration: - One terminal → Arduino 5V - Other terminal → Arduino Analog pin A1 and to GND through a pull-down resistor

The potentiometer for LCD contrast adjustment is connected: - Left pin → Arduino 5V - Middle pin → LCD V0 (contrast) - Right pin → Arduino GND

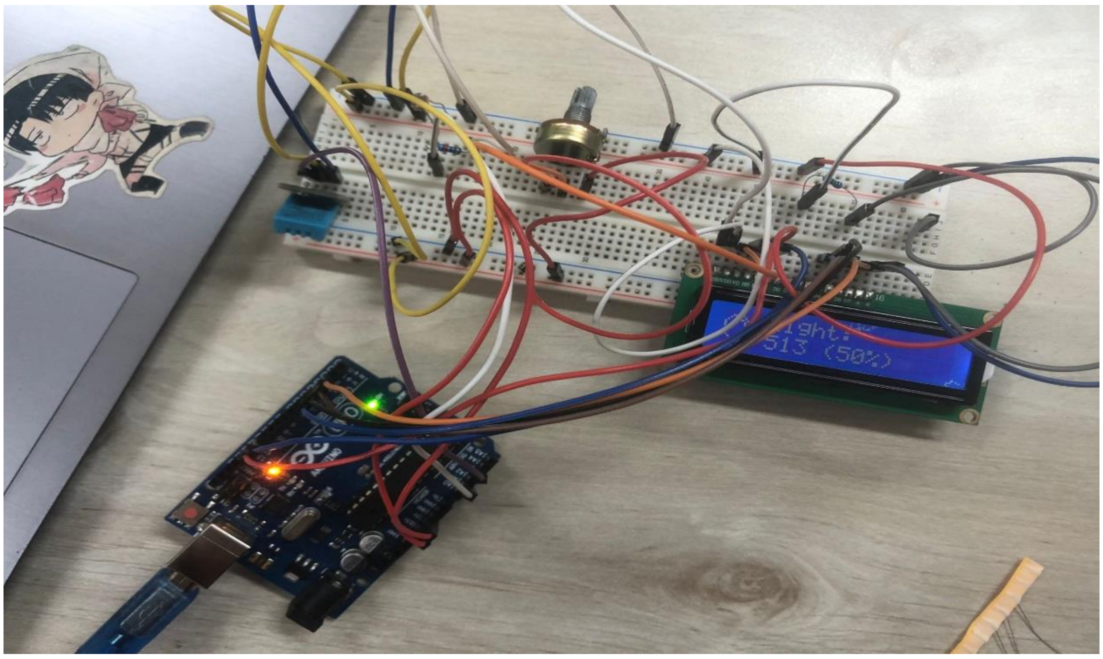


Figure (1):circuit

DHT11Temperature Sensor

The DHT11 is a precision integrated-circuit temperature sensor that provides an analog output voltage linearly proportional to the Celsius temperature. Unlike thermistors which have a non-linear response, the DHT11 has a linear output making it easier to process the readings without complex calibration or conversion formulas.

The DHT11operates on the principle that the voltage drop across a diode is temperature- dependent. Inside the DHT11, this principle is refined and calibrated to produce a precise, linear output. For every 1°C increase in temperature, the output voltage increases by 10mV (0.01V). For example, at 25°C, the output voltage would be 250mV (0.25V).

The sensor's technical specifications include an operating voltage range of 4V to 30V, a temperature range of -55°C to +150°C, and an accuracy of ±0.5°C at room temperature. Its low self-heating characteristic (less than 0.1°C in still air) ensures that the sensor's own operation doesn't significantly affect the temperature reading.

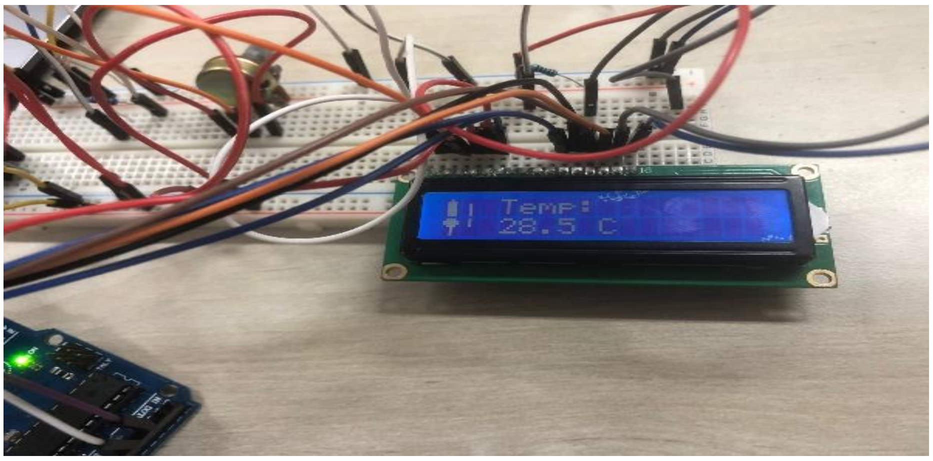
In this project, the DHT11is connected to analog pin A0 of the Arduino. The Arduino reads this analog voltage and converts it to a digital value using its built-in 10-bit Analog-to-Digital Converter (ADC). The conversion from the ADC reading to temperature in Celsius is done using the formula:

```
Temperature (°C) = (ADC Reading * 5.0 / 1023) * 100
```

Where: - ADC Reading is the digital value (0-1023) from the analog pin - 5.0 is the reference voltage of the Arduino (5V) - 1023 is the maximum value of the 10-bit ADC

(2^10 - 1) - 100 is the conversion factor (since DHT11 outputs 10mV per °C)

From the photos, we can see the DHT11successfully measuring room temperature (around 28-29°C), which is displayed on the LCD with a custom thermometer icon.



Figure(2): DHT11Temperature Sensor

LDR (Light Dependent Resistor)

A Light Dependent Resistor (LDR), also known as a photoresistor, is a passive electronic component whose resistance decreases when the intensity of light falling on it increases. This property makes it ideal for detecting ambient light levels in applications like this project.

The LDR is made of a semiconductor material with high resistance. When light falls on the semiconductor, photons from the light source are absorbed by the semiconductor material. These photons provide enough energy for electrons to break free from their atoms and contribute to electrical conduction. More light means more free electrons, which decreases the resistance of the LDR. The relationship between light intensity and resistance is non-linear, typically following an inverse exponential curve.

The technical specifications of a typical LDR include resistance in darkness of around 1MΩ or higher, resistance in bright light of a few kΩ or lower, and a spectral response similar to the human eye with peak sensitivity in the visible spectrum. The response time is typically in the range of milliseconds to seconds, depending on the type.

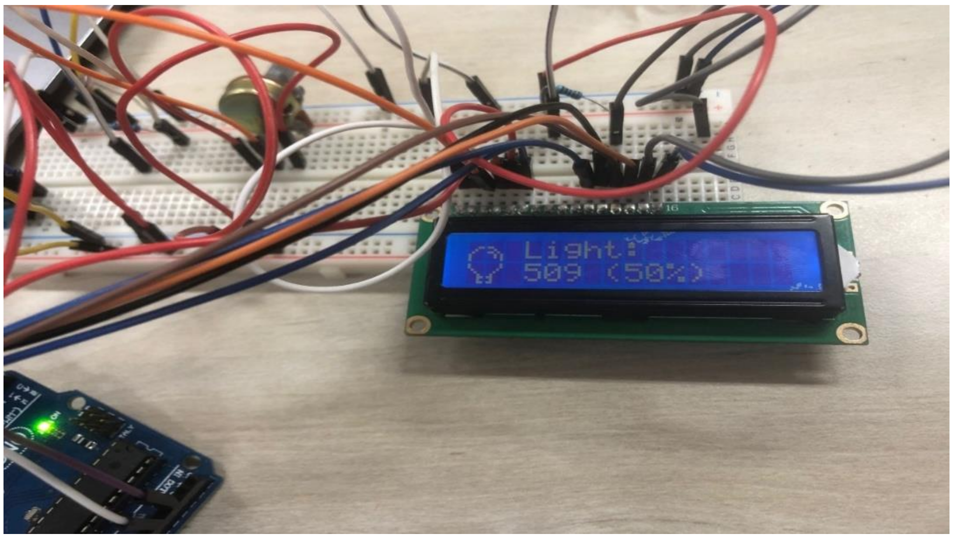
In this project, the LDR is connected to analog pin A1 of the Arduino in a voltage divider configuration. One end of the LDR is connected to 5V, while the other end is connected to both the analog pin A1 and to ground through a fixed resistor. As light levels change, the resistance of the LDR changes, altering the voltage at the analog pin. The Arduino reads this voltage and converts it to a digital value using its ADC.

The voltage at the analog pin can be calculated using the voltage divider formula:

```
Voltage at A1 = 5V * R_fixed / (R_LDR + R_fixed)
```

Where: - R_fixed is the value of the fixed resistor (typically 10kΩ) - R_LDR is the resistance of the LDR, which varies with light intensity

The Arduino code then converts this raw ADC reading (0-1023) to a percentage (0-100%) to represent the light level. From the photos, we can see readings around 50-52%, indicating moderate ambient light conditions.



Figure(3): LDR (Light Dependent Resistor)

Code Logic and Functionality

LCD Control Without External Libraries

Controlling an LCD without external libraries requires direct manipulation of the hardware pins according to the HD44780 LCD controller's timing specifications. The implementation follows a specific protocol for initialization and operation.

The initialization sequence begins by waiting for more than 15ms after power-on to ensure the LCD controller is ready. Then, the code sends the 0x30 command three times with specific timing delays as required by the HD44780 datasheet. After this, it sets the LCD to 4-bit mode (0x20) to save Arduino pins, configures the display for 2 lines with 5x8 font, turns the display on with cursor off, sets the entry mode to increment cursor without display shift, and finally clears the display and returns the cursor home

For each command or data transmission, the code sets the RS pin (Register Select) to command mode (LOW) or data mode (HIGH), sends the high nibble (4 bits) first, pulses the E pin (Enable) to latch the data, sends the low nibble, pulses the E pin again, and waits for the command to complete. This direct control method requires precise timing and bit manipulation but eliminates the need for external libraries.

Custom Character Creation

The custom thermometer and light bulb icons are created by defining 8-byte patterns where each byte represents a row of 5 pixels in the character. The process involves defining byte arrays for each custom character, sending the CGRAM (Character Generator RAM) address command, sending the 8 bytes of pattern data for each character, and returning to DDRAM (Display Data RAM) mode for normal display operations.

For example, the thermometer icon uses a pattern that represents the bulb and stem of a thermometer, while the light bulb icon depicts the characteristic shape of an incandescent bulb. These custom characters are stored in the LCD's CGRAM at positions 0 and 1, allowing them to be displayed by sending the corresponding character codes (0 and 1) to the display.

Sensor Reading Implementation

The temperature sensor reading is implemented by using `analogRead(A0)` to get a raw ADC value (0-1023), converting this value to voltage using the formula `voltage = adcValue * (5.0 / 1023.0)`, and then converting voltage to temperature with `temperature = voltage * 100.0` (since DHT11 outputs 10mV per °C). The temperature value is then formatted for display with one decimal place.

The light sensor reading follows a similar process, using `analogRead(A1)` to get a raw ADC value, converting this to a percentage with `lightPercent = (adcValue / 1023.0) * 100.0`, and formatting the light level for display with the percentage symbol.

Display Alternation Logic

The alternating display is implemented using a timing mechanism based on the `millis()` function, which returns the number of milliseconds since the Arduino began running the current program. The code stores the last time the display was updated and checks if the difference between current time and last update time exceeds 2000ms (2 seconds). If it does, it toggles a state variable between temperature and light display modes, updates the display based on the current state, and updates the last display time.

This approach avoids using `delay()`, which would block other code execution, allowing for potential future enhancements like button inputs or additional sensors. The main loop continuously reads both sensors, checks if it's time to update the display, and toggles between showing temperature and light level when appropriate.

Memory and Performance Optimization

Without external libraries, the code must efficiently manage memory. This is achieved through using 4-bit mode for the LCD to save I/O pins, minimizing variable usage (particularly avoiding floating-point variables where possible), using direct port manipulation for faster execution and smaller code size, and reusing buffers for formatting different display content.

Basic error handling is implemented through bounds checking for sensor readings, filtering out implausible values (e.g., negative temperatures when using the DHT11 in normal conditions), and ensuring proper initialization of the LCD before attempting to write to it. This approach ensures the system remains stable even if sensor readings are temporarily unreliable.

Project Implementation Evidence

Temperature Display

The photos show the LCD displaying "TEMP:" followed by temperature readings of 28.5°C and 28.2°C, with a custom thermometer icon displayed on the left side. The temperature readings are shown with one decimal place precision, and the blue backlight of the LCD provides good contrast for the displayed text and icons.

These images demonstrate the successful implementation of the temperature reading functionality using the DHT11sensor. The custom thermometer icon is clearly visible and appropriately represents the temperature measurement function. The readings appear stable and within expected room temperature range, indicating proper calibration and implementation of the DHT11 sensor interface.

Light Level Display

Several photos show the LCD in light level display mode, with "Light:" followed by numeric values ranging from 503-534 (50-52%) and a custom light bulb icon on the left side. The same blue backlight provides clear visibility for these readings as well.

These images demonstrate the successful implementation of the light level reading functionality using the LDR sensor. The custom light bulb icon effectively represents the light measurement function. The readings show moderate ambient light conditions, with the percentage representation making the values more intuitive to understand.

System Operation

The videos provide comprehensive evidence of the system's operation:

The first video shows the LCD alternating between temperature and light level displays at approximately 2-second intervals as specified in the requirements. Both custom icons (thermometer and light bulb) are clearly visible during their respective display modes, and the readings remain stable throughout, indicating reliable sensor operation.

The second video demonstrates the LDR's response to changing light conditions. When someone covers the LDR with their hand, reducing the ambient light reaching the sensor, the light level reading on the LCD decreases significantly. When the hand is removed, the reading returns to its previous value. The system continues to alternate between temperature and light displays during this demonstration, showing that the timing mechanism works reliably regardless of sensor input changes.

The third video shows the system operating over a longer period, with the LCD consistently alternating between temperature and light displays through multiple transitions. The readings remain stable, with only minor fluctuations that are normal for these types of sensors. Both custom icons continue to display correctly throughout the extended operation, and the circuit maintains proper operation without any visible issues.

Conclusion

The Arduino Temperature and Light Sensor project successfully implements all the specified requirements. It reads temperature from an DHT11 sensor on A0 and light level from an LDR on A1, displays custom icons for thermometer and light bulb, alternates between temperature and light level every 2 seconds, and uses no external libraries.

The implementation demonstrates good practices in hardware interfacing, sensor reading, and display control. The direct manipulation of the LCD without external libraries showcases a deeper understanding of hardware communication protocols. The custom character creation for the thermometer and light bulb icons adds a professional touch to the user interface.

The project could be extended in several ways, such as adding data logging capabilities, implementing a calibration routine for the sensors, or adding user input buttons to switch between different display modes or adjust settings. The current implementation provides a solid foundation for such enhancements.

Overall, this project serves as an excellent example of combining analog sensors with digital display output in an Arduino environment, demonstrating fundamental concepts in embedded systems programming and hardware interfacing.