MEMORY-CONSTRAINED DATA LOGGER

OVERVIEW

The project implements a **data logger on Arduino** that reads analog sensor values (from a potentiometer in Proteus simulation) and stores them efficiently in a **limited EEPROM memory (512 bytes)**. The main challenge is that the Arduino's internal memory is constrained, so raw storage of each sensor reading would quickly exceed available space. To address this, the project applies **delta encoding** and **Run-Length Encoding (RLE)** for compression. Delta encoding stores only the difference between consecutive readings, reducing the size of stored values, while RLE compresses consecutive repeated deltas into a single value with a repetition count, further saving memory.

The logger starts by storing the **first sensor reading as a base value** in EEPROM. Subsequent readings are logged as **16-bit signed deltas**, which allows for capturing both positive and negative changes without overflow. The RLE algorithm tracks consecutive identical deltas and writes them to EEPROM in the format [delta high byte, delta low byte, count]. This combination allows the logger to compress sequences of readings that do not change significantly, maximizing the effective use of the limited EEPROM space.

Decompression reconstructs the original sensor values by reading the base value and sequentially applying each delta multiplied by its repetition count. The system also prints **Direct sensor values** and **Decompressed values** side by side on the Virtual Terminal in Proteus, enabling verification of the compression-decompression process. The design ensures accuracy while demonstrating efficient memory utilization, making it suitable as a **prototype for low-memory IoT data logging applications**.

How I did it

I created a memory constrained data logger in arduino using its own EEPROM and only 512 bytes of it .I used delta encoding ,circular buffers and runlength encoding here.

these are the algorithms I used iint16_t as the primary data type beacause 8 bit ones caused data overflow and started giving wrong values. using int16_t gave me a lot of headaches as I had to split the values into higher and lower bytes then recombine them again for final output This taught me about bitwise operators and their uses alot

chat gpt also helped a lot tbh I had a general idea as I had worked with files in C so that helped too

I also did it completely on proteus so thers that too

source code

```
#include <EEPROM.h>


const int sensorPin = A0;

const int EEPROM_SIZE = 512;

int writeIndex = 0;

int prevValue = 0;

int16_t prevDelta = 0;

int count = 0;

bool firstSample = true;


void writeEEPROM(int addr, byte data) {

  EEPROM.update(addr % EEPROM_SIZE, data);

}




void flushRun() {

  if (count > 0) {

    writeEEPROM(writeIndex++, (byte)(prevDelta >> 8));

    writeEEPROM(writeIndex++, (byte)(prevDelta & 0xFF));

    writeEEPROM(writeIndex++, (byte)count);

    count = 0;

  }

}
```

```
void logValue(int value) {
  if (firstSample) {
    prevValue = value;
    writeEEPROM(writeIndex++, (byte)(value >> 8));
    writeEEPROM(writeIndex++, (byte)(value & 0xFF));
    firstSample = false;
    return;
  }
  int16_t delta = value - prevValue;

  if (delta == prevDelta && count < 255) {
    count++;
  } else {
    flushRun();
    prevDelta = delta;
    count = 1;
  }
  prevValue = value;
}

int decompressLatest() {
  if (writeIndex < 2) return 0;
  int16_t tmpPrev = (EEPROM.read(0) << 8) | EEPROM.read(1);
  for (int i = 2; i < writeIndex; i += 3) {
    int16_t delta = (EEPROM.read(i) << 8) | EEPROM.read(i + 1);
    int reps = EEPROM.read(i + 2);
```

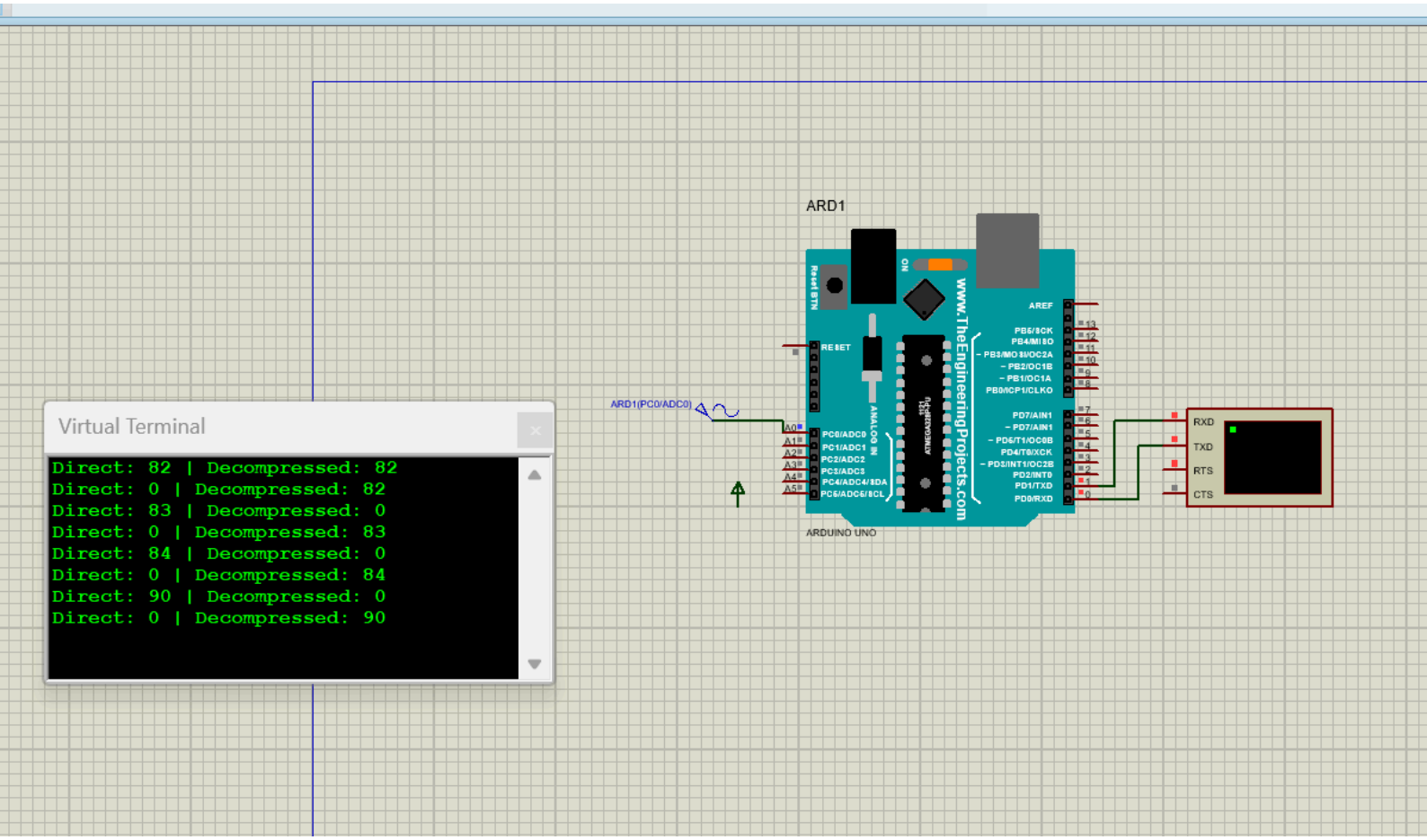
```
    tmpPrev += delta * reps;
  }
  return tmpPrev;
}

void setup() {
  Serial.begin(9600);
}

void loop() {
  int current = analogRead(sensorPin);
  logValue(current);
  int decompressed = decompressLatest();

  Serial.print("Direct: ");
  Serial.print(current);
  Serial.print(" | Decompressed: ");
  Serial.println(decompressed);
  delay(500);
}
```

output image



here direct values show the raw sensor values and decompressed values show the values retained and decompressed from the EEPROM of the arduino