Bedroom Clock

Both Daphne and I wanted an illuminated clock for night time that wasn't as jarring as a projection clock, that would not require sitting up to see the bedside table, and would be a little more aesthetically pleasing than commercially available clocks with lighted faces. I decided to make a clock similar to the previously made clock for the bathroom that additionally would be illuminated by a faint wall wash of light to allow seeing the hands. This project turned out to be a bear because we did not want a power cable hanging down, so it needed to run on batteries. Most LED night lights are activated by motion detectors to minimize the battery drain, but I feared that a motion detector would not be sufficient for our needs. Thus, I set out to find a way to power LEDs with as low a battery drain as possible with a goal of batteries not needing to be replaced more often than the battery for the clock quartz movement, that is about once a year.

Bath Clock

There were several approaches to the battery life problem. First, the lighting could be restricted to a dark conditions. This requires a light dependent resistor (LDR) that uses a steady current to monitor the ambient light. This current draw possibly uses as much battery life as running the lights 24 hours a day. The second approach uses a "Joule thief" circuit that allows the batteries to be used to very low voltages by jumping up the voltage using coil and mofset. This approach suffers from the LEDs growing fainter as the battery dies without the ability to compensate and also it was difficult to assess the expected battery life.

The approach I chose was to use microcomputer unit (MCU) to control the circuit. This MCU approach allows short (millisecond) sampling of the ambient light by photo-resistor to conserve battery, uses pulse wave modulation (PWM) to control brightness that is known to give brighter illumination with less current by pulsing the LEDs at a frequency not detectable by the eye as a flashing light. Brightness can then be controlled by changing the duty cycle of the power pulse to the LEDs. This approach additionally gives the ability to fine tune the response to ambient light and to control brightness (i.e. duty cycle) using variable resistors (trimmers). These trimmers typically continuously use current, but the MCU only activates them for interspersed millisecond periods to allow for adjustment. Since the MCU itself requires power, it is programed to go to power down during the day and to periodically, wake up for a few milliseconds only to sample ambient lighting. After detecting a dark room, the MCU looks at the battery voltage and calculates the PWM duty cycle needed to maintain the LED brightness, starts the LEDs and then goes to a light sleep state while maintaining PWM for the LEDs. Using this approach, the batteries can be run down to about 0.9 V before needing to be changed and the measurement of the current in the system predicts 9-12 months of battery life.

The electronic portion of the project had two aspects: designing the circuit and programming the MCU. Both were done using assistance of four LLMs: Gemini, ClaudeAI, ChatGPT, and CoPilot. All four were needed to cross-check component choices and to check for errors, hallucinations, and inaccuracies due to over-generalizations about components and hardware specifics. Gemini and ChatGPT seemed to be the most reliable and productive.

The MCU chosen was the ATtiny85 (Amtel Inc), an 8-bit processor with 8K of flash memory. The ATtiny85 was chosen because in addition to being adequate for the job, it had robust

sleep states and extensive experience documented on line. It could be easily programmed using C++ and a free compiler and an inexpensive USB interface board from Adafruit was available for use with an upload-program from Arduino (IDE). The C++ sketch is appended.

The circuit was designed to run on two AA alkaline batteries. The components were standard, with the only restrictions being that they must all be through-pin components rather that surface mounted due to the ease of hand-soldering to the board. The circuit uses a p-Mofset for reverse polarity protection, an LDR for light sensing, trimmers for LDR sensitivity and PWM duty cycle control and an n-Mofset for control of the LED lighting. Particular attention was given to choosing the n-Mofset, making certain that it would fully activate with low resistance at the low voltages used in the circuit. The circuit schematic was drawn using KiCad, but its convoluted user interface for PCB design precluded its use for online PCB manufacturers that cater to the DIY community. Consequently, the PCB was designed and hand made using double sided copper clad boards and etched with household chemicals (hydrogen peroxide and muratic acid). The circuit was first made on breadboards, tested for performance and current draw before final construction of the PCB. A component list and schematic is appended.

One of the greatest challenges was selecting the correct LEDs. I chose red LEDs because the forward voltage was the lowest compared to other colors or white LEDs. The LEDs needed to be efficient, i.e. bright at low current. Commercially available LED strips almost uniformly were made for 12V or incorporating AC converters. The few made to operate at 5V either were not cuttable to length or used chips to maintain 5V which was not compatible with the circuit and MCU to use low voltage batteries. I considered making LED strips. individual LEDs are now mostly very small, 0.8-1.5 mm and extremely difficult to handle and solder. I did try 5mm (5050) LEDs with the requisite specs but these were also difficult to handle and solder to copper strips. I finally found a Chinese supplier of an efficient 5V LED strip that only contained a single resistor for each LED thus was cuttable and this was chosen for use.

Design for the clock itself was derived from the previously made bathroom clock that housed a small high torque quartz movement in a small wooden frame with hands extending way beyond the clock itself. The inclusion of the two addition batteries and the circuit board necessitated a larger format. A full size mock up was made using scrap red oak left over from the stairs and flooring. A Rock Castle walnut scrap salvaged from the barn after its collapse served as the starting material for the final construction. The board was rough sawn 5/8" thick, 5-6 ' wide and about 4 ft long. It was first flat edged on the SS jointer and then flattened over 4 " with a 1/16" cut. The board was then double sided taped to a plywood strip and run through the 12" planer to flatten both sides giving a <0.5 inch thickness. A 5.5 inch square for the clock face was cut on the new-to-me SS 520 and then drilled for the quartz movement. A 2" wide strip was cut from the remainder and planed to ~3/8 inch thickness for the clock sides. The strip for the clock sides was beveled to 30 deg on the jointer for mounting the LED strips. This angle was determined by sequential planing of the mock up clock sides and viewing the spread of the wall wash lighting. Finally all the sides were cut to 1.5 inch width and miters cut for the corners. The glue-up sequentially used two sides each with an angle clamp, and then a second clamping of the four sides. After assembly of the wood, the movement, PCB and battery holder were added to the face. The LED strip was cut and soldered to give the correct angle on the sides and mounted by its self adhesive.

Wood Steps

1) ~~edge joint~~
2) ~~side joint~~
3) ~~glue up template for planing~~
4) ~~plan both sides measuring thickness to less than 0.5 in~~
5) cut 5.5 in square
6) drill center of square for movement
7) cut 2 in strip
8) plane to 3/8"
9) angle joint strip to 30deg
10) sand face and side strip
11) carefully measure and cut miters
12) check miters and recut if necessary
13) glue up

illustrations

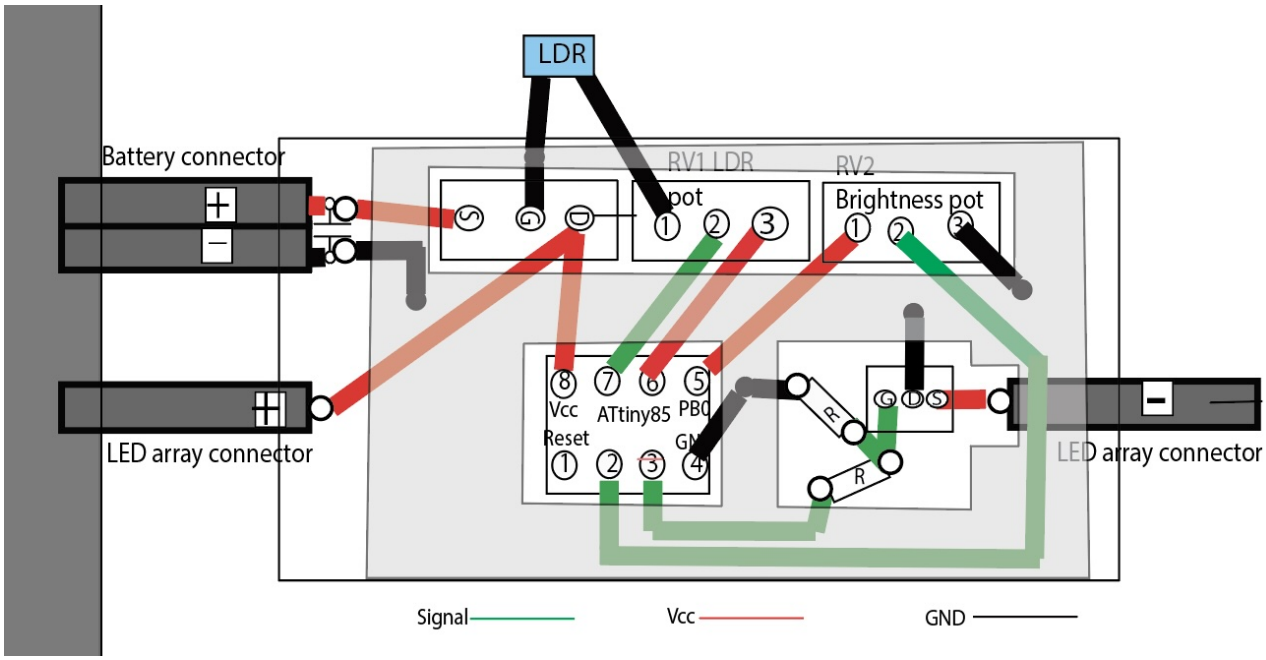

Figure 1: Drawing of the board layout

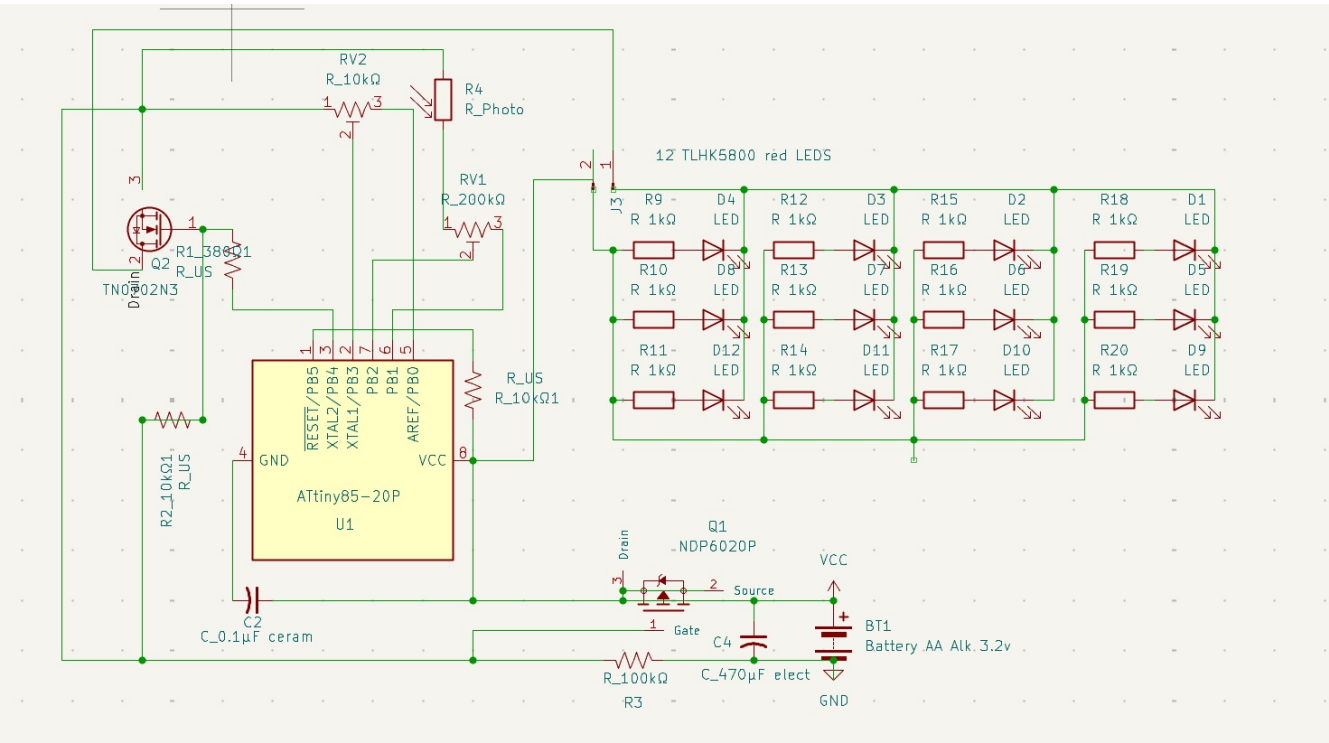

Figure 2: Circuit Diagram

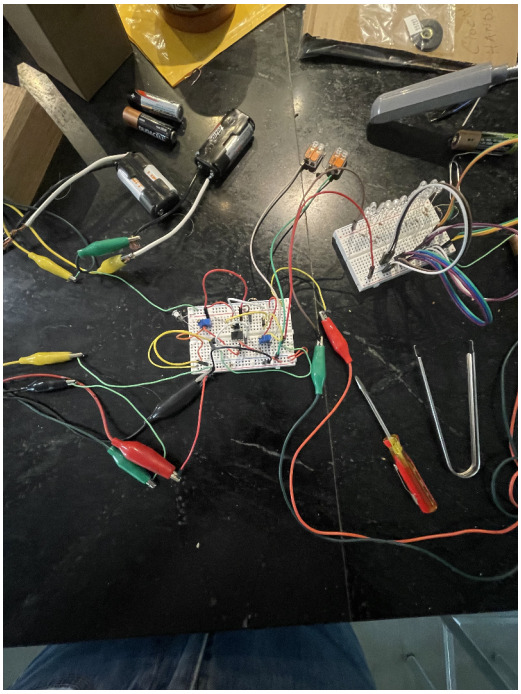

Figure 3: Bread Board Testing

PCB construction

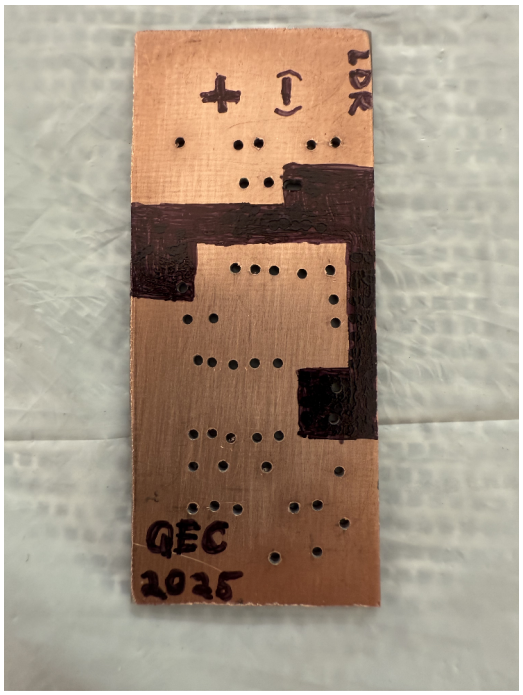

Figure 4: Masked Plate with Sharpie

Figure 5: Etching

Figure 6: Etched board

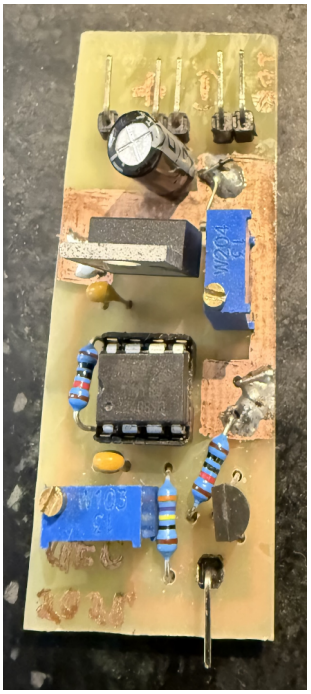

Figure 7: Assembly

starting walnut
cut sanded pieces
glue up
assembled piece
assembled with PCB etc

Parts List
MCU: ATtiny85 20PU-ND (Atmel)
DIP socket: A 08-LC-TT (Assmann WSW Components)
p-MOFSET: NDP6020P (ON Semiconductor) - reverse current control
n-MOFSET: TN0702N3-G-ND, TO92-3 (Supertex inc.) - lighting LEDs
LDR: PDV-P8103 (Advanced Photonics)
Trimpots: w103 (10kΩ), W204 (200kΩ); (Taiss)
Electrolytic Cap: 470µF; (Allecin)
Ceramic Cap: 0.1µF radial; (KEMET)
Resistors: 100Ω 1/4 W; (Yageo)
Connectors: Header Pins 2.54mm (WWZMDiB)
Battery holders: 2xAA Battery Holder with Female Dupont Connector Cable (DIYables Store)
PCB: Copper Clad Board, Double Sided 1/16" Thick, FR4 (MG Chemicals)
Clock movement: High Torque Mini Quartz Movement - 7/8" Hand Shaft (Timesavers.com)
Clock hands: Black Stick I-Shaft Hands for Quartz with 9-1/16" Minute (Timesavers.com)

Code for ATTiny85

```
//change of calculatePwmValue to set the minimal level of brightness to 0
//bright_threshold to 850;
//Spence Konde Core, use for bootloader:
//ATtiny25/45/85 (no bootloader)
//B.O.D. disabled
//Chip ATtiny85
//Clock Source "1mHz internal"; EEPROM "retained"; LTO "enabled",
//milli(s)/micro(s) "disabled"; Timer1 clock "CPU"; Programmer USBtinyISP (ATtinyCore) SLOW
// Implements a perceptual LUT for smooth dimming and optimized WDT timers
// Final version as of July 5, 2025
// changes clock scaler to 250kHz and give final PWM = 1kHZ
// changes the slope and intercept for PWM compensation calculation

#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/power.h>
#include <avr/wdt.h>
#include <math.h>
#include <avr/pgmspace.h>

// === Pin Assignments ===
const uint8_t potPowerPin = 0;       // PB0: Powers brightness pot
const uint8_t pwmPin = 4;            // PB4: PWM to MOSFET
const uint8_t ldrPowerPin = 1;       // PB1: Sensor circuit power
const uint8_t ldrAdcChannel = 1;     // PB2
const uint8_t brightnessAdcChannel = 3; // PB3

const int DARK_THRESHOLD = 950;
const int BRIGHT_THRESHOLD = 900;
const float CUTOFF_VCC_VOLTS = 1.90f;

// === Vcc Compensation Constants ===
const float REG_SLOPE = 1.873f;
const float REG_INTERCEPT = -2.923f;
const float VCC_NOMINAL_VOLTS = 3.20f;
const float I_MAX_AT_VNOMINAL = REG_SLOPE * VCC_NOMINAL_VOLTS + REG_INTERCEPT;
const float MIN_MULTIPLIER = 0.5f;
const float MAX_MULTIPLIER = 7.0f;

volatile boolean wakeUpForLDRCheck = false;

// === NEW PERCEPTUAL BRIGHTNESS LUT ===
// This table provides perceptually even steps in brightness.
// It is designed to be used with a LINEAR potentiometer.
const uint8_t brightnessLut[] PROGMEM = {
  0, 1, 2, 3, 4, 5, 6, 8, 10, 13, 17, 22, 28, 36, 46, 59, 75, 96, 123, 158, 202, 255
};
const int lutSize = sizeof(brightnessLut);

void setup() {
  CLKPR = (1 << CLKPCE);
  CLKPR = (1 << CLKPS1) | (1 << CLKPS0);  // 1 MHz

  pinMode(pwmPin, OUTPUT);
  pinMode(potPowerPin, OUTPUT);
  pinMode(ldrPowerPin, OUTPUT);
  digitalWrite(potPowerPin, LOW);
  digitalWrite(ldrPowerPin, LOW);
  digitalWrite(pwmPin, LOW); // Initialize the PWM pin to LOW

  power_all_disable();
```

```
  ACSR |= (1 << ACD);  // Disable analog comparator
  DIDR0 |= (1 << ADC1D) | (1 << ADC2D) | (1 << ADC3D);  // Disable digital input buffers

  goToPowerDown();
}

void loop() {
  if (wakeUpForLDRCheck) {
    wakeUpForLDRCheck = false;

    // === Battery Cutoff Check ===
    power_adc_enable();
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS0);
    float vcc = measureVcc();
    ADCSRA &= ~(1 << ADEN);
    power_adc_disable();

    if (vcc < CUTOFF_VCC_VOLTS) {
      power_timer1_disable();
      pinMode(pwmPin, OUTPUT);
      digitalWrite(pwmPin, LOW);
      goToPowerDown();
    }

    // === Light Level Check ===
    digitalWrite(ldrPowerPin, HIGH);
    _delay_us(1000);

    power_adc_enable();
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS0);
    int ldrVal = readAdc(ldrAdcChannel);
    ADCSRA &= ~(1 << ADEN);
    power_adc_disable();

    digitalWrite(ldrPowerPin, LOW);

    if (ldrVal > DARK_THRESHOLD) {
      wakeUpAndRunLeds(vcc);
    }
  }
  goToPowerDown();
}

void wakeUpAndRunLeds(float vcc) {
  // Change the clock prescaler to slow down the system clock for power saving
  CLKPR = (1 << CLKPCE);  // Enable clock prescaler change
  CLKPR = (1 << CLKPS2) | (1 << CLKPS0);  // Set prescaler to 32. This divides the BASE 8MHz clock.
  // 8 MHz / 32 = 250 kHz system clock
  power_adc_enable();
  ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS0);
  digitalWrite(potPowerPin, HIGH);
  _delay_us(500);

  // --- MODIFIED: Use averaging for a more stable reading ---
  long brightnessTotal = 0;
  for (int i = 0; i < 4; i++) {
    brightnessTotal += readAdc(brightnessAdcChannel);
  }
  int brightness = brightnessTotal / 4;

  digitalWrite(potPowerPin, LOW);
  ADCSRA &= ~(1 << ADEN);
  power_adc_disable();

  int pwm = calculatePwmValue(brightness, vcc);
```

```
  power_timer1_enable();
  analogWrite(pwmPin, pwm);

  while (true) {
    ADCSRA &= ~(1 << ADEN);
    power_adc_disable();
    goToIdle();

    power_adc_enable();
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS0);
    digitalWrite(potPowerPin, HIGH);
    digitalWrite(ldrPowerPin, HIGH);
    _delay_us(500);

    // --- MODIFIED: Use averaging for a more stable reading ---
    long newBrightnessTotal = 0;
    for (int i = 0; i < 4; i++) {
      newBrightnessTotal += readAdc(brightnessAdcChannel);
    }
    int newBrightness = newBrightnessTotal / 4;

    int ldr = readAdc(ldrAdcChannel);

    digitalWrite(potPowerPin, LOW);
    digitalWrite(ldrPowerPin, LOW);
    ADCSRA &= ~(1 << ADEN);
    power_adc_disable();

    if (ldr < BRIGHT_THRESHOLD) break;

    int newPwm = calculatePwmValue(newBrightness, vcc);
    if (abs(newPwm - pwm) > 2) {
      analogWrite(pwmPin, newPwm);
      pwm = newPwm;
    }
  }

  power_timer1_disable();
  pinMode(pwmPin, OUTPUT);
  digitalWrite(pwmPin, LOW);
}

// --- REPLACED: This is the final, corrected function ---
int calculatePwmValue(int brightnessSetting, float vcc) {
  // Dead zone for a clean "off" state. Tune this value as needed.
  if (brightnessSetting < 15) {
    return 0;
  }

  // 1. Map the linear ADC input to an index for our perceptual LUT.
  int lutIndex = map(brightnessSetting, 0, 1023, 0, lutSize - 1);

  // 2. Look up the corresponding non-linear PWM value from the table.
  uint8_t basePwm = pgm_read_byte(&brightnessLut[lutIndex]);

  // 3. The VCC compensation logic remains unchanged.
  float i_actual = REG_SLOPE * vcc + REG_INTERCEPT;
  float multiplier = (i_actual <= 1.0f) ? MAX_MULTIPLIER : I_MAX_AT_VNOMINAL / i_actual;
  multiplier = fmaxf(MIN_MULTIPLIER, fminf(MAX_MULTIPLIER, multiplier));

  float compensated = basePwm * multiplier;
  return constrain((int)compensated, 0, 255);
}

float measureVcc() {
```

```
  ADMUX = (1 << MUX3) | (1 << MUX2);
  long total = 0;
  for (int i = 0; i < 10; i++) {
    ADCSRA |= (1 << ADSC);
    while (bit_is_set(ADCSRA, ADSC));
    if (i > 1) total += ADC;
  }
  return 1126.4f / ((float)total / 8.0f);
}

int readAdc(uint8_t channel) {
  ADMUX = (channel & 0x0F);
  ADCSRA |= (1 << ADSC);
  while (bit_is_set(ADCSRA, ADSC));
  return ADC;
}

// --- MODIFIED: WDT timers optimized for power and responsiveness ---
void goToPowerDown() {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  cli();
  wdt_reset();
  WDTCR = (1 << WDCE) | (1 << WDE);
  // Set to 8 seconds for maximum power saving when LEDs are off
  WDTCR = (1 << WDIE) | (1 << WDP3) | (1 << WDP0);
  sei();
  sleep_cpu();
  sleep_disable();
  wdt_disable();
}

void goToIdle() {
  set_sleep_mode(SLEEP_MODE_IDLE);
  sleep_enable();
  cli();
  wdt_reset();
  WDTCR = (1 << WDCE) | (1 << WDE);
  // Set to 125ms for responsive dimming control when LEDs are on
  WDTCR = (1 << WDIE) | (1 << WDP2);
  sei();
  sleep_cpu();
  sleep_disable();
  wdt_disable();
}

ISR(WDT_vect) {
  wakeUpForLDRCheck = true;
}
```

Claude AI summary of circuit and control.

ATtiny85 Light-Activated LED Controller — Full Circuit Description

Purpose: A 2×AA battery-powered nightlight that automatically switches an LED array on at dusk, dims/brightens intelligently as the battery discharges, and sleeps at microamp-level current between light checks to maximize battery life.

Power Path

- **J2 (battery input)** → **NDP6020P (P-channel reverse-polarity protection)** → rest of circuit.
- The P-MOSFET's body diode and low VGS(th) (~-0.7V) let it conduct fully even as the battery sags toward 2.0V; if the battery is inserted backward, the MOSFET stays off and protects everything downstream.
- **C1/C3** decouple the supply against the switching transients from PWM.

Sensing

- **LDR + R1 voltage divider** feeds an ATtiny85 ADC pin — this is the ambient-light sensor.
- **RV1** sets/trims the light threshold (day/dark crossover point read by the firmware).
- **RV2** sets manual brightness/dimming level, read on a separate ADC pin.
- A pull-down resistor on the LDR ADC pin (added after debugging) prevents the floating-input fault that originally kept LEDs dark until the pin was probed.

Control — ATtiny85 (the brain)

The firmware ties sensing and power switching together with several coordinated behaviors:

1. **Light decision with hysteresis** — instead of a single threshold, the code uses separate `DARK_THRESHOLD_ON/OFF` and `DAYLIGHT_THRESHOLD_ON/OFF` values, and requires several consecutive ADC readings past the threshold before flipping state. This debouncing stops flicker at twilight when ambient light hovers right at the crossover.
2. **PWM brightness output** — once "dark" is confirmed, the ATtiny85 drives `analogWrite` on the gate of the N-channel switching MOSFET (TN0702N3-G), setting LED brightness per RV2's manual setting.
3. **VCC compensation** — periodically (~every 5 minutes, to limit the cost of doing it), the firmware measures the actual supply voltage using the ATtiny85's internal 1.1V bandgap reference, then recalculates the PWM duty cycle upward as VCC drops. This keeps LED brightness visually consistent through the whole battery discharge curve rather than dimming as the batteries wear down.
4. **Sleep management** — two distinct low-power states:
    - **Power-down sleep** while LEDs are off, woken only by a pin-change interrupt from the LDR circuit — this is where nearly all the battery life comes from.
    - **Idle sleep** while LEDs are active, which keeps Timer/PWM hardware running so the LED output doesn't glitch, while still sleeping the CPU core between interrupts.
5. **Watchdog timer** — an 8-second watchdog resets the system if firmware ever hangs while LEDs are on, adding field reliability without user intervention.

Switching

- **TN0702N3-G (N-channel, low-side switch)** sits between the LED array and ground. Its 1.0V max gate threshold and datasheet-guaranteed RDS(ON) at VGS=2V make it reliably drivable directly from the ATtiny85's PWM pin even as battery voltage approaches 2V — no gate driver needed.
- **12 LEDs (TLHK5800), each with its own 1kΩ series resistor**, arranged in parallel — individual resistors mean one LED failing doesn't take down the array, and current stays low (~0.2mA–2mA per LED, ~24mA total at full brightness).

How it all works together

Ambient light drops → LDR divider crosses the dark threshold (confirmed over several readings) → ATtiny85 wakes from power-down via pin-change interrupt → switches to idle sleep, begins PWM'ing the TN0702N3-G gate at a brightness set by RV2 and corrected for current battery voltage → watchdog keeps it running reliably → when ambient light later crosses back above the (separate) daylight threshold, PWM is cut to zero and the ATtiny85 drops back into power-down sleep, drawing next to nothing until the next pin-change interrupt.