Thermometer bewegingsmelder en clock

Thermometer bewegingsmelder en clock

Het display laat de temperatuur en luchtvochtigheid zien en toont de trend met een pijltje omhoog of naar beneden.

Op de tweede regel staat de datum en tijd.

Het display gaat alleen 'aan' als er beweiging wordt geconstateerd.

Gemeten stroom is ongeveer 40 mA voor de gehele schakeling.

Screenshot 2025-03-22 110222.jpg

Aansuitschema

PIR Bewegingsmelder

shopping.webp

Van onder gezien van links naar rechts
(Let op er zijn verschillende pin configuraties. Verwijder lans/cap om de pin configuratie te zien).

- Plus 5V
- Signaal naar Pin 12 Arduino Leonarde.
- Min

Signaal naar Digitaal pin 12 Arduono Leonarde

Clock DS 3231

Untitled.jpg

Screenshot 2025-03-22 184122.jpg

http://www.rinkydinkelectronics.com/library.php?id=73

Van links naar rechts van onderkant (niet batterij kant) gezien en *laatste* 4 pootjes.

- SCL naar SCL op Arduino (op Leonarde meest rechter pin)
- SDA naar SDA op Arduino (op Leonarde op een na meest rechter pin)
- Plus 3.3V
- Min

Thermometer

image.png

Van links naar rechts (gaatjes van blauwe blokje boven).

- naar pin 4 Arduino Leonarde
- Plus 3.3V
- Min

Zie https://www.roc.ovh/books/arduino/page/temperatuur-en-luchtvochtigheid

Display

image.png

Zie https://www.roc.ovh/books/arduino/page/display

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#include <DS3231.h>

#define DHTPIN 4                // Pin connected to the DHT11 data pin
#define DHTTYPE DHT11           // Specify DHT11 sensor
#define SIGNAL_INTERVAL 600000  // 10 minutes

DHT dht(DHTPIN, DHTTYPE);
```

```
DS3231 myRTC;
bool century = false;
bool h12Flag;
bool pmFlag;

int count = 0;
int pirPin = 12;            // Pin for the HC-S501 sensor
int pirValue;

LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);
unsigned long previousMillis = 0;
const long dhtInterval = 2000; // Interval for DHT readings

// Custom characters for the LCD arrows
byte downChar[] = {
  B00000,
  B00000,
  B00100,
  B00100,
  B00100,
  B10101,
  B01110,
  B00100
};

byte upChar[] = {
  B00100,
  B01110,
  B10101,
  B00100,
  B00100,
  B00100,
  B00000,
  B00000
};

byte degreeChar[] = {
  B00110,
  B01001,
```

```
  B01001,
  B00110,
  B00000,
  B00000,
  B00000,
  B00000
};

//--------------------------------------------------------------------------------
// Generic MeasurementSensor Class Template
//--------------------------------------------------------------------------------
template <typename T>
class MeasurementSensor {
  private:
    T measurement;             // Current measurement value
    int arrow;                 // Arrow indicator: 0 = no arrow, 1 = down, 2 = up
    unsigned long arrowMillis; // Last time the arrow was updated/reset
    const long signalInterval; // Time interval to reset the arrow

  public:
    // Constructor: initializes with an initial measurement value.
    MeasurementSensor(long sigInterval, T initValue)
      : measurement(initValue), arrow(0), arrowMillis(0), signalInterval(sigInterval) {}

    // Update the measurement reading and determine the arrow indicator
    void update(T newMeasurement, unsigned long currentMillis) {
      // If the new measurement equals the previous value and the signal interval has passed,
reset arrow.
      if (newMeasurement == measurement && currentMillis - arrowMillis >= signalInterval) {
        arrow = 0;
      } else {
        if (newMeasurement > measurement) {
          arrow = 2;  // Up arrow
          arrowMillis = currentMillis;
        }
        if (newMeasurement < measurement) {
          arrow = 1;  // Down arrow
          arrowMillis = currentMillis;
        }
      }
```

```
      // If the previous value is the initial invalid value, clear the arrow indicator.
      if (measurement == static_cast<T>(-99)) {
        arrow = 0;
      }

      measurement = newMeasurement;
    }

    // Getters for measurement and arrow
    T getMeasurement() const { return measurement; }
    int getArrow() const { return arrow; }
};

//--------------------------------------------------------------------------------
// Global Instances for Temperature and Humidity
//--------------------------------------------------------------------------------
MeasurementSensor<float> tempSensor(SIGNAL_INTERVAL, -99.0);
MeasurementSensor<int> humiditySensor(SIGNAL_INTERVAL, -99);

void setup() {
  Serial.begin(9600);
  dht.begin();
  delay(2000);

  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.createChar(1, downChar);
  lcd.createChar(2, upChar);
  lcd.createChar(3, degreeChar);

  pinMode(pirPin, INPUT);

  // myRTC.setYear(2025);
  // myRTC.setMonth(3);
  // myRTC.setDate(20);
  // myRTC.setHour(23);
  // myRTC.setMinute(49);
  // myRTC.setSecond(0);
```

```
}

void print2digits(int number) {
  if (number < 10) {
    lcd.print("0");
  }
  lcd.print(number, DEC);
}

void loop() {
  delay(20);

  // lcd.print(":");
  // print2digits(myRTC.getSecond());

  // Check for movement
  pirValue = digitalRead(pirPin);
  if (pirValue) {
    lcd.backlight();
  }

  // Do we need to update the display?
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= dhtInterval) {
    previousMillis = currentMillis;

    // Update backlight based on PIR sensor
    pirValue = digitalRead(pirPin);
    if (!pirValue) {
      lcd.noBacklight();
    }

    count++;

    // Read new values from the DHT sensor
    float newTemperature = dht.readTemperature();
    int newHumidity = dht.readHumidity();

    if (isnan(newTemperature) || isnan(newHumidity)) {
      Serial.println("Failed to read from DHT sensor!");
```

```
  return;
}

// Update our measurement sensors
tempSensor.update(newTemperature, currentMillis);
humiditySensor.update(newHumidity, currentMillis);

// Print sensor readings to the Serial Monitor
Serial.print(count);
Serial.print(", Temperature: ");
Serial.print(tempSensor.getMeasurement(), 1);
Serial.print("°C");
Serial.print(", Humidity: ");
Serial.print(humiditySensor.getMeasurement());
Serial.print("%");
Serial.println("");
Serial.println(myRTC.getSecond(), DEC);
Serial.println("");

// Update the LCD display for temperature
lcd.setCursor(0, 0);
if (tempSensor.getArrow()) {
  lcd.write(tempSensor.getArrow());
} else {
  lcd.print(" ");
}
lcd.print(tempSensor.getMeasurement(), 1);
lcd.write(3); // degree symbol
lcd.print("C ");

// Update the LCD display for humidity (set cursor on second row)
lcd.setCursor(12, 0);
if (humiditySensor.getArrow()) {
  lcd.write(humiditySensor.getArrow());
} else {
  lcd.print(" ");
}
lcd.print(humiditySensor.getMeasurement());
lcd.print("%");
```

```
    lcd.setCursor(1, 1);
    print2digits(myRTC.getDate());
    lcd.print("-");
    print2digits(myRTC.getMonth(century));

    lcd.setCursor(10, 1);
    print2digits(myRTC.getHour(h12Flag, pmFlag));
    lcd.print(":");
    print2digits(myRTC.getMinute());
  }
}
```

Code met higest/lowest

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>
#include <DS3231.h>

#define DHTPIN 4                // Pin connected to the DHT11 data pin
#define DHTTYPE DHT11           // Specify DHT11 sensor
#define SIGNAL_INTERVAL 600000  // 10 minutes

DHT dht(DHTPIN, DHTTYPE);

DS3231 myRTC;
bool century = false;
bool h12Flag;
bool pmFlag;

int count = 0;
int pirPin = 12;  // Pin for the HC-S501 sensor
int pirValue;

LiquidCrystal_I2C lcd = LiquidCrystal_I2C(0x27, 16, 2);
unsigned long previousMillis = 0;
const long dhtInterval = 2000;  // Interval for DHT readings

// Custom characters for the LCD arrows
byte downChar[] = {
```

```
  B00000,
  B00000,
  B00100,
  B00100,
  B00100,
  B10101,
  B01110,
  B00100
};

byte upChar[] = {
  B00100,
  B01110,
  B10101,
  B00100,
  B00100,
  B00100,
  B00000,
  B00000
};

byte degreeChar[] = {
  B00110,
  B01001,
  B01001,
  B00110,
  B00000,
  B00000,
  B00000,
  B00000
};

//--------------------------------------------------------------------------------
// Generic MeasurementSensor Class Template
//--------------------------------------------------------------------------------
template<typename T>
class MeasurementSensor {
private:
  T measurement;  // Current measurement value
  T lowestMeasurement;
```

```
  T highestMeasurement;
  int arrow;                  // Arrow indicator: 0 = no arrow, 1 = down, 2 = up
  unsigned long arrowMillis;  // Last time the arrow was updated/reset
  const long signalInterval;  // Time interval to reset the arrow

public:
  // Constructor: initializes with an initial measurement value.
  MeasurementSensor(long sigInterval, T initValue)
    : measurement(initValue), arrow(0), arrowMillis(0), lowestMeasurement(0),
highestMeasurement(0), signalInterval(sigInterval) {}

  // Update the measurement reading and determine the arrow indicator
  void update(T newMeasurement, unsigned long currentMillis) {

    // If this is the first valid measurement, initialize min/max
    if (lowestMeasurement == static_cast<T>(0) && highestMeasurement == static_cast<T>(0)) {
      lowestMeasurement = newMeasurement;
      highestMeasurement = newMeasurement;
    }

    // Determine highest and lowest
    if (newMeasurement < lowestMeasurement) {
      lowestMeasurement = newMeasurement;
    }
    if (newMeasurement > highestMeasurement) {
      highestMeasurement = newMeasurement;
    }

    // If the new measurement equals the previous value and the signal interval has passed,
reset arrow.
    if (newMeasurement == measurement && currentMillis - arrowMillis >= signalInterval) {
      arrow = 0;
    } else {
      if (newMeasurement > measurement) {
        arrow = 2;  // Up arrow
        arrowMillis = currentMillis;
      }
      if (newMeasurement < measurement) {
        arrow = 1;  // Down arrow
        arrowMillis = currentMillis;
```

```
      }
    }

    // If the previous value is the initial invalid value, clear the arrow indicator.
    if (measurement == static_cast<T>(-99)) {
      arrow = 0;
    }

    measurement = newMeasurement;
  }

  // Getters for measurement and arrow
  T getMeasurement() const {
    return measurement;
  }
  T getLowest() const {
    return lowestMeasurement;
  }
  T getHighest() const {
    return highestMeasurement;
  }
  int getArrow() const {
    return arrow;
  }
  void resetMinMax(T currentValue) {
    lowestMeasurement = currentValue;
    highestMeasurement = currentValue;
  }
};

//--------------------------------------------------------------------------------
// Global Instances for Temperature and Humidity
//--------------------------------------------------------------------------------
MeasurementSensor<float> tempSensor(SIGNAL_INTERVAL, -99.0);
MeasurementSensor<int> humiditySensor(SIGNAL_INTERVAL, -99);

void setup() {
  Serial.begin(9600);
  dht.begin();
  delay(2000);
```

```
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.createChar(1, downChar);
  lcd.createChar(2, upChar);
  lcd.createChar(3, degreeChar);

  pinMode(pirPin, INPUT);

  // myRTC.setYear(2025);
  // myRTC.setMonth(3);
  // myRTC.setDate(20);
  // myRTC.setHour(10);
  // myRTC.setMinute(49);
  // myRTC.setSecond(0);
}

void print2digits(int number) {
  if (number < 10) {
    lcd.print("0");
  }
  lcd.print(number, DEC);
}

int lastResetHour = -1;  // Initialize to an invalid hour

void loop() {
  delay(20);

  // lcd.print(":");
  // print2digits(myRTC.getSecond());

  // Check for movement
  pirValue = digitalRead(pirPin);
  if (pirValue) {
    lcd.backlight();
  }

  // Do we need to update the display?
```

```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= dhtInterval) {
  previousMillis = currentMillis;

  // Update backlight based on PIR sensor
  pirValue = digitalRead(pirPin);
  if (!pirValue) {
    lcd.noBacklight();
  }

  count++;

  // Read new values from the DHT sensor
  float newTemperature = dht.readTemperature();
  int newHumidity = dht.readHumidity();

  if (isnan(newTemperature) || isnan(newHumidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Get current hour from RTC
  int currentHour = myRTC.getHour(h12Flag, pmFlag);
  // Check if hour has changed
  if (currentHour != lastResetHour) {
    tempSensor.resetMinMax(newTemperature);
    humiditySensor.resetMinMax(newHumidity);
    lastResetHour = currentHour;
    Serial.println("min/max values reset");
  }

  // Update our measurement sensors
  tempSensor.update(newTemperature, currentMillis);
  humiditySensor.update(newHumidity, currentMillis);

  // Update the LCD display for temperature
  lcd.setCursor(0, 0);
  if ( count % 6 ) {
    if (tempSensor.getArrow()) {
      lcd.write(tempSensor.getArrow());
```

```
      } else {
        lcd.print(" ");
      }
      lcd.print(tempSensor.getMeasurement(), 1);
      lcd.write(3);  // degree symbol
      lcd.print("C     ");
    } else {
      lcd.print("");
      lcd.print(tempSensor.getLowest(),1);
      lcd.write(3);
      lcd.print(" ");
      lcd.print(tempSensor.getHighest(),1);
      lcd.write(3);
      lcd.print(" ");
    }

    // Update the LCD display for humidity (set cursor on second row)
    if (humiditySensor.getArrow()) {
      lcd.write(humiditySensor.getArrow());
    } else {
      lcd.print(" ");
    }
    lcd.print(humiditySensor.getMeasurement());
    lcd.print("%");

    lcd.setCursor(1, 1);
    print2digits(myRTC.getDate());
    lcd.print("-");
    print2digits(myRTC.getMonth(century));

    lcd.setCursor(10, 1);
    print2digits(myRTC.getHour(h12Flag, pmFlag));
    lcd.print(":");
    print2digits(myRTC.getMinute());
  }
}
```

--

Revision #16
Created 2025-03-22 09:58:49 UTC by Max
Updated 2025-03-30 09:17:20 UTC by Max