I-ITWK

Belegarbeit

Energieoptimierung eines batteriebetriebenen Wohnklima-Warngerätes

Lukas Brüggemann, Markus Reinhold, Sebastian Pötter

Jahrgang: 20INM

im Studiengang
Informatik Master

Embedded Systems (7320)
Prof. Dr.-Ing. Matthias Sturm

7. Januar 2022

Inhaltsverzeichnis

Abkürzungsverzeichnis

CPU Control Processor Unit

DHT Digital temperature and humidity sensor

GPIO General Purpose Input/Output

LED Light emitter diode

LPM Low Power Mode

PWM Pulsweitenmodulation

USB Universal Serial Bus

Glossar

Lookup-Table Speicherung von Ergebnissen aus komplexen Berechnungen, welche zur Laufzeit nur noch aufgerufen werden müssen.

1 Aufgabenstellung

Es soll mit dem Mikrocontroller MSP430G2553 die Lufttemperatur und die Luftfeuchtigkeit gemessen und die Werte über die On-Board LED visualisiert werden. Die Software ist mit 'Code Composer Studio' in C zu entwickeln. Einzelne Assembleranweisungen dürfen in den C–Code eingebettet werden, jedoch keine kompletten Module. Der Quellcode ist vollständig und nachvollziehbar zu kommentieren. Hierfür liegt die Implementierung für das Auslesen des Sensors bereits vor. Diese Implementation ist jedoch absichtlich ineffizient. Somit sollte die Implementierung so angepasst werden, dass der Stromverbrauch minimal ist. Dazu soll eine Schätzung der Laufzeit im Bezug auf den Stromverbrauch und der Langlebigkeit des Gerätes aufgestellt und berechnet werden. Hierbei soll die Schätzung und eine Analyse auf Basis von 2 herkömmlichen AA-Batterien sein (in Reihe, je 2,5 Ah).

Darüber hinaus soll eine Dokumentation im Umfang eines 8-10 Seiten Dokumentes verfasst werden. Darin wird der Aufbau des Programms sowie einzelne Teilfunktionen vorgestellt und erklärt. Auch soll die Dokumentation eine Bedienungsanleitung beinhalten, welche die Inbetriebnahme und die LED Anzeige erklärt.

Folgende Mindestanforderungen werden gestellt:

- Start und Stop der Messfunktion mittels Taster
- Signalisierung von behaglichem, noch behaglichem und unbehaglichem Raumklima per LED Anzeige
- Anzeige Luftfeuchte zu hoch oder zu niedrig

2 Hardware und Konzept

Für diese Aufgabe werden folgende Bauteile benötigt.

- MSP430G2553 LaunchPad Development Kit
- DHT22
- LED (Onboard)
- USB Kabel für Debug, Datenaustausch und Stromversorgung

Der 16-Bit-CPU MSP430 ist für kostengünstige und insbesondere stromsparende Embedded-Anwendungen konzipiert [3]. Hierbei kann auch noch nach Leistung und Speichergröße ausgewählt werden. Der hier verwendete Mikrocontroller ist der MSP430G2553.

Dieser Mikrocontroller soll mit dem DHT22 Sensor kommunizieren und erhaltene Daten weiterverarbeiten. Dabei soll aus der Lufttemperatur und der Luftfeuchtigkeit eine Gefühlte-Temperatur berechnet werden, welche über eine On-Board LED visualisiert wird. Die Abbildung 2 zeigt ein solches Board mit allen möglichen Pins und weiteren Informationen für die Entwickler.

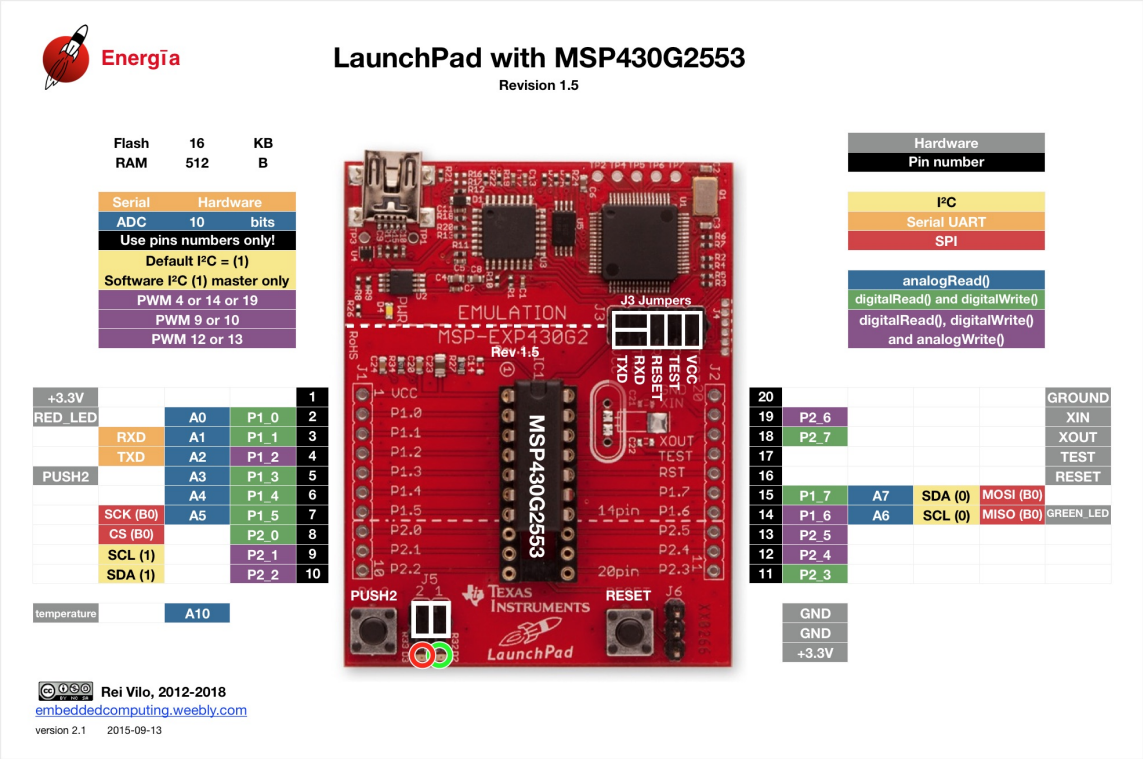

Abbildung 1: LaunchPad MSP430G2452 Board [4]

Allgemeine Daten des DHT22 Sensors [2] aus dem Datenblatt.

- 3 bis 5V Leistung und E/A
- 2,5mA maximaler Stromverbrauch während der Datenanforderung
- 0-100% Luftfeuchtigkeitsmessungen mit 2-5% Genauigkeit
- Arbeitstemperatur -40 bis 80°C ±0,5°C Genauigkeit
- Nicht mehr als 0,5 Hz Abtastrate (einmal alle 2 Sekunden)

Ob eine Lufttemperatur und Luftfeuchtigkeit behaglich ist, kann die so genannte Behaglichkeit beschreiben. Hierbei gibt es spezielle mathematische Formeln, welche aus den zuvor genannten Werten eine Behaglichkeit bestimmen. Die Behaglichkeit könnte wie in der Abbildung 2 klassifiziert werden. Wobei die X-Achse die Lufttemperatur beschreibt und die Y-Achse die Luftfeuchtigkeit. Im Rahmen dieses Projektes wird in

dem Programm für den Mikrocontroller eine abstrahierte Variante dieses Diagramms benutzt.

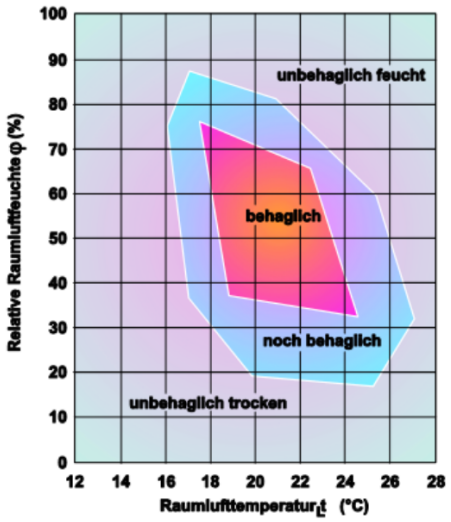

Abbildung 2: klassifizierung der gefühlten Temperatur [1]

Die Abbildung 3 beschreibt die möglichen Zustände des Programms. Dabei bedeutet 'Aus', dass das Programm möglichst energiesparend im Leerlauf arbeitet. 'An' soll den aktiven Zustand der Programms signalisieren. Hierbei wird die Messung, die Berechnung und die Visualisierung verarbeitet. Dabei wird jeweils in bestimmten Abständen gemessen und die LED's angesprochen.

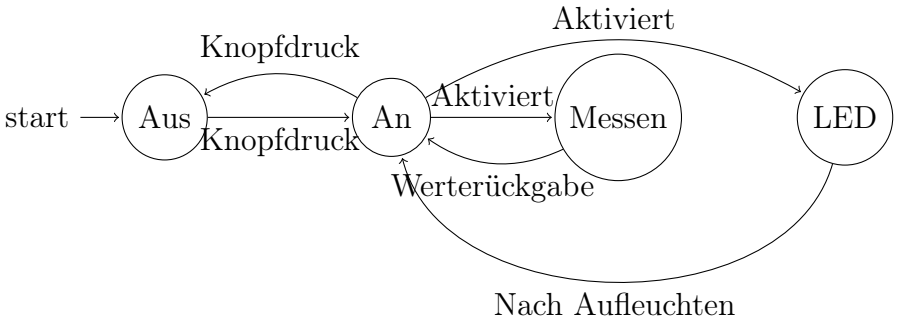

Abbildung 3: Zustandsübergang

3 Implementierung

Für die Umsetzung des vorgestellten Konzepts wurde zuerst ein Programm geschrieben, welches nicht auf den Energieverbrauch optimiert ist und später als Referenz für potentielle Energieeinsparungen dienen soll. Die einzelnen Programmteile werden im folgenden genauer vorgestellt.

3.1 Möglichkeit zum Pausieren der Messung

Damit die Messung pausiert werden kann, wird ein Interrupt für den Knopfdruck vorbereitet. Dieser überprüft zuerst, in welchem Zustand sich das Programm gerade befindet und startet bzw. stoppt dann die Messung.
Anstelle der Verwendung der vorgegebenen `__delay_cycles()`-Funktion - welche nicht vorzeitig beendet werden kann - wurde zudem ein Timer benutzt, welcher sowohl die Messung als auch die Umfärbung der Status-LED steuert. Durch Setzen des `TACCR0`-Registers kann dieser sofort gestoppt werden. Listing 1 zeigt den Programmcode der beiden Interrupts.

```
#pragma vector=PORT1_VECTOR
__interrupt void Port_1_ISR(void)
{
    dhtCheckActive ^= 1; // toggle active state
    if (dhtCheckActive) {
        dhtCheckPending = 1; // do sensor check on next program cycle
        TACCR0 = 1000*16-1; //Start Timer, Compare value for Up Mode to get 1ms delay per loop
                /*Total count = TACCR0 + 1. Hence we need to subtract 1.
                1000 ticks @ 16MHz will yield a delay of 1ms.*/

    } else {
        TACCR0 = 0;
        P1OUT &= ~(BIT0 + BIT6); //deactivate (debug) leds
    }

P1IFG &= ~0x08; // P1.3 IFG cleared
}
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A_CCR0_ISR(void)
{
    measureTimerCount++;
    // DEBUG CODE flash the green led if timer is active
    if (measureTimerCount % 100 == 0)
    {
        P1OUT ^= BIT0; // P1.0 = toggle
    }
    // END OF DEBUG
    if(measureTimerCount >= MEASURE_TIMEOUT_MS)
    {
        P1OUT ^= BIT6; // DEBUG CODE toggle red led everytime a measurement is called from the timer
        dhtCheckPending = 1; // do sensor check on next program cycle
        measureTimerCount = 0;

    }
}
```

Listing 1: Knopfdruck-Interrupt

Der Aufruf der Messfunktion erfolgt jedoch nicht direkt in einem vom Timer aufgerufenen Interrupt, sondern innerhalb der Endlosschleife in der `main`-Funktion. Listing 2 zeigt den Aufbau, bei welchem die Messfunktion nur aufgerufen wird, wenn die dazugehörige Variable `dhtCheckPending` vom Timer verändert wurde.

```
int main() {
    // [.. init ..]
    //main program loop - react to actions called by interrupts
    while(1){
        if (dhtCheckPending) {
            dhtCheckPending = 0;
            measureDHT();
        }
    }
}
```

Listing 2: Main-Funktion mit Endlosschleife

3.2 Ermittlung der Komfortzone

Da der Temperaturfühler eine Messungenauigkeit von +- 0.5°C aufweist, wurden anstatt einer genauen Messung Lookup-Tables erstellt, welche die jeweils minimalen und maximalen Feuchtigkeitswerte einer Komfortzone beinhalten. Dazu wurde zuerst die Einteilung der Zonen aus Abbildung 2 vervollständigt, sodass dem Benutzer später angezeigt werden kann, ob die Luftfeuchte erhöht oder verringert werden muss. Die resultierenden Zonen sind in Abbildung 4 dargestellt.

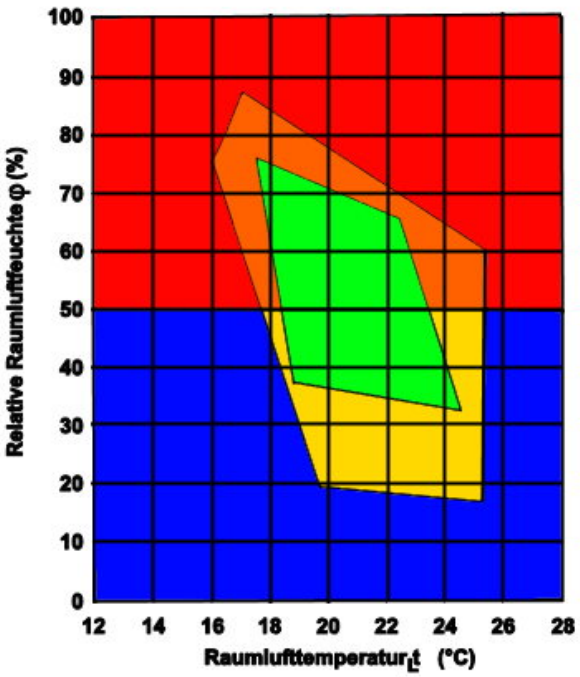

Abbildung 4: klassifizierung der gefühlten Temperatur in der lookup table

Für die genaue Umsetzung wurde je ein eindimensionales Array für die Zonen 'behaglich' und 'noch behaglich' erstellt. Die Arrays sind so formatiert, dass eine Zeile jeweils die niedrigste Luftfeuchte und die höchste Luftfeuchte der entsprechenden Temperatur darstellt. Jeder Wert außerhalb dieser Arrays ist als unbehaglich anzusehen und wird dabei noch in 'unbehaglich trocken' oder 'unbehaglich feucht' unterschieden. Listing 4 zeigt die entstandenen Arrays.

```
const int HUM_RANGE_COMFORTABLE[] =  {
    63,75, // 18 Grad
    37,73,
    36,71,
    35,68,
    35,66,
    34,55,
    33,41 // 24 Grad
};

const int HUM_RANGE_STILL_COMFORTABLE[] = {
    75,75, // 16 Grad
    36,87,
    30,85,
    25,83,
    19,82,
    19,81,
    18,76,
    18,71,
    17,66,
    17,63,
    22,50,
    32,32 // 27 Grad
};
```

Listing 3: Lookup-Tabellen der ursprünglichen Komfortzonen

Für die bessere Verständlichkeit des Programms wurden zudem Konstanten festgelegt, welche zur Unterscheidung der verschiedenen Komfort-Levels verwendet werden. Jede Konstante repräsentiert dabei eine farbige Fläche in den neu erstellten Komfortzonen.

```
// constants for predefined comfort levels
#define COMFORT_LEVEL_UNCOMFORTABLE_DRY 0
#define COMFORT_LEVEL_UNCOMFORTABLE_WET 1
#define COMFORT_LEVEL_COMFORTABLE 2
#define COMFORT_LEVEL_STILL_COMFORTABLE_WET 3
#define COMFORT_LEVEL_STILL_COMFORTABLE_DRY 4
```

Listing 4: Konstanten für die Komfortzonen

Letztendlich entscheidet die in Listing 5 gezeigte Funktion `getDHTComfortState`, ob die gemessenen Werte (Lufttemperatur und Luftfeuchtigkeit) noch in dem Bereich 'behaglich' und danach ob es noch im Bereich 'noch behaglich' ist. Ist der Wert nicht in diesen Bereichen, wird dann nur noch geprüft ob es zu trocken oder zu feucht ist. Dazu wird zuerst untersucht, ob sich die Temperatur überhaupt in einer der Lookup-Tabellen befindet, sodass es nicht zu unerwarteten Ergebnissen führt.

```
int getDHTComfortState(int temperature, int humidity) {
    //first, check if our values are inside the approximation of the
    two comfort zones
    int check_comfortable = ((temperature >= 18 && temperature <= 24)
            && (humidity >= 33 && humidity <= 75));
    int check_still_comfortable = ((temperature >= 16 && temperature <=
    27)
            && (humidity >= 17 && humidity <= 87));
    // check inner polygon (comfortable) first
    if (check_comfortable) {
        ...
    }
    //after that, check for second polygon (still comfortable)
    if (check_still_comfortable) {
        int low = HUM_RANGE_STILL_COMFORTABLE[2*(temperature-16)];
        int high = HUM_RANGE_STILL_COMFORTABLE[2*(temperature-16) + 1];

        if (humidity >= low && humidity <= high) {
            ...
        }
    }
    //last, decide if conditions are too dry or wet
    if (humidity < 50) {
        return COMFORT_LEVEL_UNCOMFORTABLE_DRY;
    }
    return COMFORT_LEVEL_UNCOMFORTABLE_WET;
}
```

Listing 5: Ermittlung der Komfortzone aus Temperatur und Luftfeuchte

4 Optimierungen am Quellcode

Obwohl der bisherige Code die benötige Funktionalität bereitstellt, ist dieser nicht sonderlich energieeffizient. Der folgende Abschnitt stellt daher Möglichkeiten zur Einsparung von Energie vor, etwa durch eine veränderte Nutzung der Status-LED oder der Nutzung verschiedener Hardwarebestandteile.

4.1 Variable Taktfrequenz

Durch die Zuweisung der Konstanten BCSCTL1 und DCOCTL kann die Taktrate des Mikrocontrollers verändert werden. Im vorgegebenen Programm ist diese auf 16 MHz festgelegt worden, weitere Möglichkeiten sind 12, 8 und 1 MHz. Die zu erwartende Lösung wäre, die Taktfrequenz standardmäßig auf 1 MHz zu konfigurieren, da

eine geringere Taktrate auch weniger Verbrauch bedeutet. Da der Temperatur- und Feuchtigkeitssensor jedoch bei dieser Taktrate nicht mehr zuverlässig angesprochen werden kann, muss während der Messung der Takt kurzzeitig von 1 MHz auf 8 MHz angehoben werden. Die verbleibende Zeit kann der Mikrocontroller im 1 MHz Takt betrieben werden. Zu diesem Zweck wurde die folgende Methode implementiert:

```
void setCPUThrottled(int throttled){
    if (throttled){
        BCSCTL1 = CALBC1_1MHZ;
        DCOCTL = CALDCO_1MHZ;
    } else {
        BCSCTL1 = CALBC1_8MHZ;
        DCOCTL = CALDCO_8MHZ;
    }
}
```

Listing 6: Variable Taktfrequenz

4.2 Interruptgesteuerter Timer und Low Power Modes

Anstatt in einer endlosen while-Schleife auf den nächsten Messzyklus zu warten, kann die Zeit zwischen zwei Zyklen alternativ über einen hardwareseitigen Timer realisiert werden. Der Mikrocontroller bietet dafür verschiedene Varianten an. Um eine möglichst lange Zeit zwischen einer Timer-Periode zu realisieren wurde dafür der Timer 'ACLK' verwendet, welcher auf einer Frequenz von lediglich etwa 32 kHz agiert. Dies hat den Vorteil, dass Periodendauern von bis zu rund 16 Sekunden realisiert werden können, in denen die kein Interrupt bearbeitet werden muss. Durch eine softwareseitige Zählvariable können noch größere Zeitintervalle realisiert werden, wobei jedoch weiterhin nach 16 Sekunden ein Interrupt ausgelöst wird.

```
#define TIMEOUT_MULTIPLIER 4; // nur aller X Perioden soll die
    eigentliche Logik aufgerufen werden
#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A_CCR0_ISR(void)
{
    currentTimeoutCount++;
    if (currentTimeoutCount < TIMEOUT_MULTIPLIER) {
        return;
    }
    currentTimeoutCount = 0;
    // restlicher Code, eigentliche Programmlogik
```

Listing 7: Interruptgesteuerter Timer

Die aus der Verwendung eines Timers mit geringer Taktfrequenz auftretenden Ungenauigkeiten können für das vorliegende Projekt vernachlässigt werden, da sich der Nutzer (im Gegensatz zu beispielsweise einer Uhr) nicht auf den absolut genauen Takt verlässt. Ein weiterer Vorteil von hardwareseitigen Timern entsteht durch die Kombination mit den sogenannten 'Low Power Modes' des Mikrocontrollers. Je nach Modus werden verschiedene Peripherien deaktiviert, welches eine Energieeinsparung zur Folge hat. Ebenso kann die durchgängige Ausführung des Quellcodes außerhalb von Interruptaufrufen pausiert werden. Low Power Mode 3 ist der restriktivste Modus, welcher noch die Benutzung von Timern erlaubt. Da das aktive Warten durch Timer ersetzt wurde kann sich der Controller größtenteils im Low Power Mode 3 befinden. Lediglich für die Messung von Temperatur und Luftfeuchte muss dieser kurzzeitig deaktiviert werden.

4.3 Anpassungen der LED

Die Status-LED ist in der unoptimierten Version der größte Energieverbraucher. Umso wichtiger ist es, dass diese so effizient wie möglich verwendet wird, ohne die Benutzbarkeit des Programms maßgeblich zu verschlechtern. Anstatt einer kontinuierlichen Anzeige der Zustände 'behaglich' und 'noch behaglich' wird ein Blinkrhythmus implementiert, welcher die LED nur etwa eine halbe Sekunde aktiviert und danach für eine Minute deaktiviert. Vergleichbar ist dieses Verhalten mit einem Rauchmelder für Privathaushalte, welcher durch eine blinkende Lampe einzig und allein signalisiert, dass die Batterien noch nicht leer sind. Für den Zustand 'behaglich' wird dazu die LED grün gefärbt, im 'noch behaglichen' Zustand blinkt die LED entweder rot oder blau, je nachdem, ob die Luftfeuchte zu hoch und zu niedrig ist. Somit kann der Benutzer schon vor dem Erreichen des 'unbehaglichen' Zustands entsprechend handeln. Wenn der Zustand dennoch unbehaglich wird, leuchtet die LED durchgängig und je nach Luftfeuchte rot oder blau. Der daraus resultierende erhöhte Energieverbrauch kann dadurch begründet werden, dass der Benutzer sofort und durchgängig über den Ausnahmezustand informiert werden soll, vergleichbar mit dem akustischen Signal eines Rauchmelders im Falle eines Feuers.

Weiterhin wurde die hell leuchtende LED mit Hilfe von Pulsweitenmodulation (PWM) auf ihre halbe Helligkeit gedimmt, sodass zusätzliche Energie gespart werden kann. Trotz der reduzierten Helligkeit kann die LED weiterhin auch bei normalem Tageslicht abgelesen werden. Lediglich bei direkter Sonneneinstrahlung kann selbst die ungedimmte LED nicht mehr erkannt werden. Die PWM kann hardwareseitig durch Benutzung des zweiten Timers realisiert werden (der erste wird bereits für die Pulse der Messung verwendet) und kann somit auch im Low Power Mode erfolgen. Da

es sich bei der Ansteuerung der LEDs nur um wenige Zeilen handelt, wird dies direkt in Interrupts durchgeführt. Aus diesem Grund wird der Puls-Timer bei blinkender LED nicht auf die komplette Zeit bis zur nächsten Messung konfiguriert, sondern vielmehr auf die nächste Änderung einer LED. Aus diesem Grund existiert eine Hilfsvariable, welche den derzeitigen Zustand des Boards speichert und durch welche im Puls-Interrupt die jeweiligen Aktionen ausgeführt werden.

4.4 Weitere Anpassungen im Quellcode

Da die Grenzen zwischen den Komfort-Levels in der Aufgabenstellung mit Ganzzahlen festgelegt wurden, können diese ebenso im Programm so umgesetzt werden. Zu diesem Zweck werden die Rückgaben der Temperatur und der Luftfeuchte des DHT-Sensors in Ganzzahlen konvertiert. Zusätzlich wurden unbenutzte GPIO pins auf Low geschalten, damit diese weniger Strom verbrauchen. [5]

5 Energieverbrauch und Laufzeitanalyse

Die unoptimierte Version des Codes ergab die folgenden Werte im Bezug auf die Laufzeit. Dieser Code ist nicht optimiert und es gibt nur den Modus in dem das Gerät misst und den Modus in dem das Gerät im Standby ist. Für eine vergleichbare Schätzung ist das Gerät für 30 Minuten am Tag im aktivem Modus und für die restliche Zeit im Standby, diese Aufteilung ist in Tabelle 1 dargestellt.

Betriebsarten	
Mode	Dauer
LED an und Messung	1800s/d (0,5h)
Standby	84600s/d (23,5h)

Tabelle 1: Betriebsarten

Der Verbrauch in den einzelnen Betriebsmodi ist in Tabelle 2 dargestellt. Diese Werte wurden durch EnergyTrace gemessen.

Stromverbrauch (mA)	
Mode	Total
LED an und Messung	13,0466
Standby	4,923

Tabelle 2: Stromverbrauch

Durch Berechnung des täglichen Energieverbrauchs anhand der Zeit in einem Betriebsmodi, sowie dem Verbrauch in diesem Modi, ergibt sich ein täglicher Verbrauch von 122,2138 mA/h. Diese Zusammensetzung sowie die Aufteilung des Energieverbrauchs im Verhältnis zum gesamten Verbrauch ist in der Tabelle 3 verbildlicht.

Täglicher Energieverbrauch				
Mode	Dauer	Stromverbrauch mA	Energie mA/h	Prozent total
LED an und Messung	1800s/d (0,5h)	13,0466	6,5233	5%
Standby	84600s/d (23,5h)	4,923	115,6905	95%
Totaler Energieverbrauch pro Tag			122,2138	100%

Tabelle 3: Täglicher Energieverbrauch

Durch Berechnung des täglichen Verbrauches im Verhältnis zu der verfügbaren Kapazität der Batterien ergibt sich eine Laufzeit von 20 Tagen. Diese finale Auswertung ist in Tabelle 4 zu sehen.

Batterielebensdauer	
Energiequelle	2xAA
Kapazität (mAh)	2500 mA/h
Minimale Umgebungstemperatur	20°C
Täglicher Durchschnittsenergieverbrauch	122,2138 mA/h
Erwartete Batterielebensdauer	20 Tage

Tabelle 4: Batterielebensdauer

6 Energieverbrauch und Laufzeitanalyse der optimierten Version

Annahmen:

- Nachts wird auf Standby geschaltet
- Am Tag wird für 30 Minuten ein kritischer Wert erreicht
- Für die restliche Zeit wird durch Blinken der Wert mitgeteilt

Diese Aufteilung sowie zusätzliche Informationen zu den Betriebsmodi sind in der Tabelle 5 dargestellt. Im Vergleich zu dem Test der unoptimierten Version gibt es einen 3. Betriebsmodus in dem eine LED blinkt.

Betriebsarten			
Mode	Beschreibung	Dauer	Bemerkung
LED an	LED dauerhaft an	1800s/d (0,5h)	extrem Werte
LED blinkt		55800s/d (15,5h)	normaler Bereich
Standby		28800s/d (8h)	nachts während man schläft

Tabelle 5: Betriebsarten (Optimiert)

In Tabelle 6 werden die im EnergyTrace gemessenen Werte des optimierten Quellcodes angezeigt. Hierbei ist der gemessene Stromverbrauch wesentlich niedriger als der aus Tabelle 2 des unoptimierten Quellcodes.

Stromverbrauch (mA)	
Mode	Total
LED an	7,3414
LED blinkt	0,1739
Standby	0,0826

Tabelle 6: Stromverbrauch (Optimiert)

Durch das Berechnen des täglichen Energieverbrauchs anhand der Zeit, in dem sich das Gerät in einem Modus befindet, mit dem Verbrauch in diesem Modus, ergibt sich ein totaler Verbrauch von 7,02695 mA/h pro Tag. Diese Zusammensetzung sowie die Aufteilung des Energieverbrauchs im Verhältnis zum gesamten Verbrauch ist in der Tabelle 7 verbildlicht.

Täglicher Energieverbrauch				
Mode	Dauer	Stromverbrauch mA	Energie mA/h	Prozent total
LED an	1800s/d (0,5h)	7,3414	3,6707	52%
LED blinkt	55800s/d (15,5h)	0,1739	2,69545	38%
Standby	28800s/d (8h)	0,0826	0,6608	9%
Totaler Energieverbrauch pro Tag			7,02695	100%

Tabelle 7: Täglicher Energieverbrauch (Optimiert)

Mit der Batteriekapazität der AA-Batterien von 2500 mA/h und dem täglichen Durchschnittsenergieverbrauch von 7,02695 mA/h erhalten wir eine Laufzeit von 356 Tagen. Diese finale Auswertung ist in Tabelle 8 zu sehen.

Batterielebensdauer	
Energiequelle	2xAA
Kapazität (mAh)	2500 mA/h
Minimale Umgebungstemperatur	20°C
Täglicher Durchschnittsenergieverbrauch	7,02695 mA/h
Erwartete Batterielebensdauer	356 Tage

Tabelle 8: Batterielebensdauer (Optimiert)

7 Bedienungsanleitung

Sobald das Thermostat mit Strom versorgt wurde, kann es für die Behaglichkeitsmessung benutzt werden. Um das Thermostat in den aktiven Modus zu versetzen, muss dazu noch der Knopf gedrückt werden. Im aktiven Modus werden die gemessenen Werte über die LED's visualisiert.

Blinkt eine grüne LED des Thermostats, so ist das Klima behaglich und es muss nichts unternommen werden. Falls die LED blau blinkt, ist das Klima noch behaglich aber zu feucht. Rotes blinkendes Licht bedeutet, dass das Klima noch behaglich aber etwas trocken ist. Dauerhaft leuchtendes Rot bedeutet, das Klima ist nicht mehr behaglich und es sollte etwas dagegen unternommen werden. Auch dauerhaft blaues Licht bedeutet, dass das Klima nicht mehr behaglich ist und das es zu feucht ist.

Um das Thermostat wieder abzuschalten muss der Knopf nochmal gedrückt werden. Im Schlaf-Modus ist der Energieverbrauch am niedrigsten. Eine weitere Möglichkeit ist das Thermostat wieder vom Strom zu trennen. Das Thermostat sollte wenn möglich über Nacht entweder vom Strom getrennt oder in den 'Schlaf-Modus' versetzt werden. Hierfür kann der Energieverbrauch auf ein Jahr geschätzt werden, wenn das Gerät acht Stunden im Schlaf-Modus und die restlichen 16 Stunden im Betriebsmodus ist. Um einen noch niedrigeren Energieverbrauch zu erreichen, sollte das Thermostat möglichst lang und oft im 'Schlaf-Modus' sein oder nur angeschaltet werden, wenn er aktiv gebraucht wird.

8 Fazit

Es konnte der Aufgabenstellung entsprochen und dabei eine optimierte und energiesparende Version des Programms entwickelt werden. Durch dieses Projekt wurde ein tieferes Verständnis für ressourcenschonende Programmierung vermittelt. Vor allem auch in Hinsicht der Hardware und dessen Spezifikation für maximale Laufzeit. Das

Problem dieses Projektes lag eher darin, dass die eigentliche Arbeit am Projekt erst Ende Dezember möglich war. Dadurch konnte man nur effektiv einen Monat an diesem Projekt arbeiten, weil es Ende Januar bereits wieder abgegeben werden musste. Dieses Projekt war sehr interessant und lehrreich bezüglich 'low power' Anwendungen. Es sollte jedoch darüber nachgedacht werden, dass jedes Gruppenmitglied eigene Hardware zum experimentieren bekommt, da sich sonst nur eine Person mit dem Quellcode wirklich beschäftigen kann.

Trotz, dass die LED im Vergleich zur Originalversion länger an ist, ist die Laufzeit mit 356 Tagen im Vergleich zu 20 Tagen viel länger. Das kommt zum großen Teil durch die Optimierung im Standby, welcher fast 60 mal weniger Energie verbraucht als im Original. Die Verbesserungen des Verbrauchs im Betriebsmodus, welcher im extremsten Fall trotzdem nur halb so viel Energie verbraucht, kommen durch eine Vielzahl von Verbesserungen und Veränderungen.

Dieser Aufbau und die Art und Weise wie das Gerät benutzt wird, hat einen großen Effekt darauf, wie lange die Laufzeit des Gerätes ist. Nutzer die das Gerät nur kurz anschalten um eine kurze Information zum Zustand des Raumes zu erhalten, können eine weitaus längere Laufzeit des Gerätes erwarten.

Literaturverzeichnis

Literatur

[1] *Behagliche Temperatur*. 2020. URL: https://upload.wikimedia.org/wikipedia/commons/c/ce/Temperatur_Feuchtigkeit_behaglich.png (besucht am 20. 10. 2020).

[2] *DHT22 Sensor*. 2020. URL: https://www.adafruit.com/product/385 (besucht am 23. 11. 2020).

[3] *MSP430 Mikrocontroller*. 2020. URL: https://www.ti.com/microcontrollers/msp430-ultra-low-power-mcus/overview.html (besucht am 23. 11. 2020).

[4] *Guide to the MSP430G2 LaunchPad (MSP-EXP430G2)*. URL: https://energia.nu/pinmaps/msp-exp430g2/ (besucht am 03. 12. 2020).

[5] *MSP430 GPIO Programming Tutorial*. URL: http://www.ocfreaks.com/msp430-gpio-programming-tutorial/ (besucht am 23. 01. 2020).

Abbildungsverzeichnis

Tabellenverzeichnis

A Selbständigkeitserklärung

Hiermit erkläre ich, das wir die vorliegende Arbeit mit dem Titel "Projekt für Embedded Systemsßelbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Leipzig, 7. Januar 2022

Name (Unterschrift)