

# **Green IT**

## **Projektaufgabe**

**Prof. Dr. V. Iossifov**

# Inhaltsverzeichnis

<b>Einleitung</b>	<b>3</b>
Motivation	3
Aufgabe	4
<b>Werkzeuge</b>	<b>5</b>
MSP430g2553	5
Code Composer Studio	6
TI EnergyTrace Technology	7
Ultra-Low Power Advisor	7
<b>Ausgangssituation</b>	<b>9</b>
Präsentation Fallbeispiel	9
Analyse Fallbeispiel	10
<b>Optimierung</b>	<b>14</b>
Vorgehen und Umsetzung	14
Analyse	21
<b>Vergleich</b>	<b>23</b>
<b>Zusammenfassung und Ausblick</b>	<b>25</b>
<b>Anhang</b>	<b>26</b>
Anhang A: Code Fallbeispiel-Inefficient	26
Anhang B: Code: Code Fallbeispiel-Efficient	27

# Einleitung

Der Bereich "Green IT" oder auch "Green computing" ist ein breit gefächertes Gebiet. So kann es als Teil der Informatik gesehen werden. Wobei auch andere Wissenschaften und Fachgebiete von besagter Green IT tangiert werden. Einleitend ist dabei die Frage zu beantworten, was nun unter besagtem Bereich zu verstehen ist. Eingängig wird dazu eine Orientierung auf Wikipedia gegeben. Danach ist Green It "the study and practice of environmentally sustainable computing or IT. The goals of green computing: reduce the use of hazardous materials, maximize energy efficiency during the product's lifetime, the recyclability or biodegradability of defunct products and factory waste."<sup>1</sup>. Innerhalb dieser Arbeit wird sich auf gegebene Definition bezogen. Zuzufügen ist, dass die Green IT als ein aufstrebender Bereich angesehen werden kann. Dies wird besonders deutlich bei aktuellen Bestrebungen hinsichtlich der sogenannten "Internet of things" - Thematik.

## Motivation

Mit den bisherigen einleitenden Bemerkungen wurde das aktuelle Anwachsen der Bemühungen im zugrunde liegenden Bereiche, eben der Green IT, angedeutet. Motiviert werden diese Bemühungen aus verschiedenen Hinsichten.

Allgemein ist festzuhalten, dass lediglich begrenzte Ressourcen zur Energieerzeugung vorhanden sind. Können diese Ressourcen geschont werden ist dies unbestreitbar vorteilhaft. Darüber hinaus sind Überlegungen und auch schon Umsetzungen für den Umstieg auch die sogenannten erneuerbaren Energien zu bemerken. Innerhalb dieser Phase des Wechsels, also in der Anfangsphase erneuerbare Energien, ist die Energiegewinnung innerhalb dieses Bereiches vergleichsweise unter Umständen unbeständig oder zu gering. Werkzeuge und Geräte, die bei vergleichbarer Funktionsweise weniger Energie benötigen sind somit präferiert.

Ein weiterer Grund ist eher eingängig. So bedeutet der verbrauch weniger Energie, also weniger Ressourcen, weniger Betriebskosten des Gerätes. Auch aus diesem einfachen Grund ist eine Motivation hinzu Green IT begründbar.

---

<sup>1</sup> [https://en.wikipedia.org/wiki/Green\\_computing#Approaches](https://en.wikipedia.org/wiki/Green_computing#Approaches)

Bei aktuell steigender Bevölkerungszahl ist zudem vorstellbar, dass im vergleichbaren Verhältnis die Nutzung der technischen Werkzeuge und Hilfsmittel ebenso ansteigt. Ein gesellschaftliche Reduzierung oder Verzicht dieser Produkte ist dabei als eher unwahrscheinlich einzustufen. Wenn nun der Gebrauch weder eingeschränkt noch eingestellt wird und dennoch auf eine Reduktion der benötigten Ressourcen gesetzt wird, verbleibt lediglich noch die Option den Energiebedarf gegebener Geräte zu minimieren. Beschriebene Motivation der Green IT, gerade mit dem zuletzt beschriebenen Argument ist somit gegeben, begründet und notwendig.

## Aufgabe

Innerhalb eines Projektes des Moduls Green IT wurde sich mit dieser Arbeit entschlossen, sich mit der Energieoptimierung eines Mikrocontrollers auseinander zu setzen. Festgelegt wurde sich auf die Aufgabe: "A 1 Ampel\_Mealy\_4Z als Mealy-Automat". Gegeben ist somit ein Fallbeispiel eines Mealy-Automaten. Dieser, aktuell bereits voll funktionsfähige, Automat wechselt zwischen seinen Zuständen durch Druck auf einen externen Knopf. Die Zustände beschreiben aktuell grob den Ablauf einer Ampelschaltung. So existieren die Zustände

1. Blinken der Grünen LED
2. Blinken der roten LED
3. Blinken der grünen und roten LED zeitgleich
4. Blinken der grünen und roten LED zeitversetzt

Die semantischen Hintergründe sind dabei nicht oder nur teilweise auf die tatsächliche LED-Farbe oder sonstiges bezogen. Da diese semantischen Hintergründe nicht vordergründig für das Thema sind, wird auf diese innerhalb dieser Arbeit auch nicht weiter eingegangen. Aufgabe ist es diesen beschriebenen Automat in seiner Ressourcennutzung zu optimieren. Somit soll die Funktionsweise nach Optimierung wenig oder überhaupt nicht von der bisherigen abweichen.

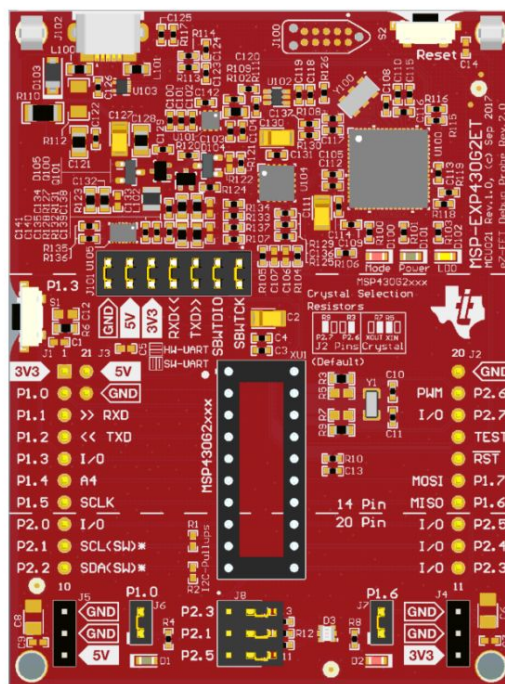
# Werkzeuge

Die Umsetzung beschriebener Aufgabe erfolgt unter Benutzung mehrerer Werkzeuge. Diese Werkzeuge sind hilfreich für das praktische Betreiben des Automaten, die Erkennung zu optimierender Code-Abschnitte und die Bestimmung der Ressourcennutzung des Automaten.

## MSP430g2553

Die Hardware-Grundlage für das Projekt wird gebildet als zwei Modulen. Das erste Modul ist das “MSP-EXP430G2ET LaunchPad development kit” von Texas Instruments (TI).

Dieses Launchpad wird als Board benutzt.

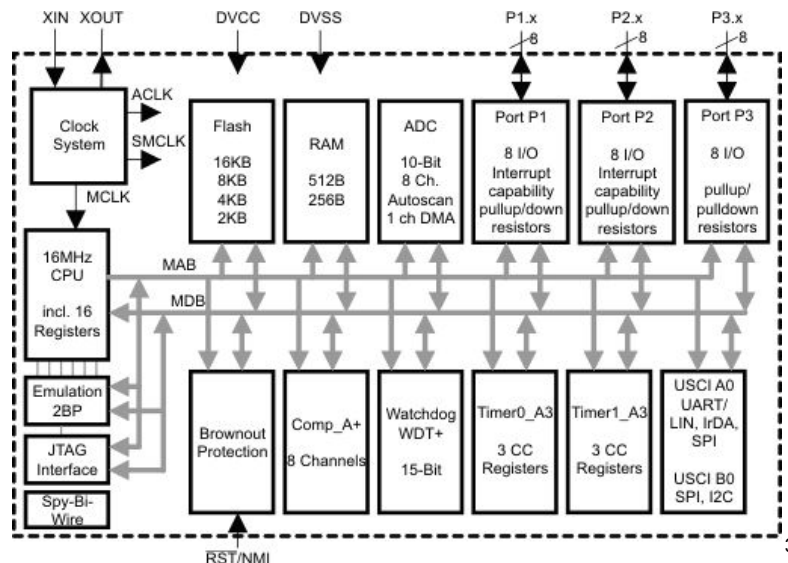


2

Das Board zeichnet sich aus unter anderem aus durch einen Socket-Platz für beispielsweise eine MSP430G2553 MCU. Das Board selbst bietet darüber hinaus 2 Knöpfe. Einer davon ist frei konfigurierbar. Als Output werden unter anderem 2 LEDs angeboten, grün und gelb. Für die sogenannte EnergyTrace-Technology-Funktion werden spezifische Komponenten benötigt. Diese sind auf diesem Boards ebenfalls installiert.

<sup>2</sup> <https://www.ti.com/lit/ug/slau772/slau772.pdf>

Als zweites Modul wird ein Mikrocontroller (MCU) der Familie “MSP430”, ebenfalls von Texas Instruments, benutzt. Konkret handelt es sich um den “MSP430g2553” - Microcontroller. Diese MCU ist auf dem Board montiert, sodass die angebotenen Schnittstellen des Boards benutzt werden können.



Diese MCU zeichnet sich unter anderem aus durch eine 16-Bit Architektur. Die Geschwindigkeit des Prozessors (CPU) liegt standardmäßig zwischen 0.8 und 1.5 MHz<sup>4</sup>. Es ist jedoch möglich die Taktfrequenz der CPU auf maximal 16 MHz zu erhöhen oder durch gewisse Feineinstellungen runter zu regulieren oder auszuschalten. Die MCU bietet zwei 16-Bit Timer, mit 3 “capture compare register” (CCR) angeboten. Darüber hinaus besteht die Möglichkeit einen sogenannten “Ultra-Low-power-modus” zu aktivieren, wodurch die Ressourcen der MCU drastisch reduziert werden.

## Code Composer Studio

Zur Entwicklung des Quelltextes wird die Entwicklungsumgebung “Code Composer Studio” (CCS) benutzt. Entwickelt wurde das CCS von Texas Instruments. Somit ist dieses Entwicklungsprogramm auf die hier zugrundeliegende Hardware spezialisiert. Das dürfte unter anderem ein Grund dafür sein, dass diese Umgebung von TI, besonders bei der

<sup>3</sup> <https://www.ti.com/product/MSP430G2553#tech-docs>

<sup>4</sup>

[https://www.ti.com/lit/ds/symlink/msp430g2553.pdf?ts=1595350876992&ref\\_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430G2553](https://www.ti.com/lit/ds/symlink/msp430g2553.pdf?ts=1595350876992&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FMSP430G2553)

Entwicklung unter der MCU MSP430-Familie, empfohlen wird. Angeboten wird umfangreiche Unterstützung im Bereich des Debuggings mit angeschlossenem Controller. Dabei wird der Code auf den Controller übertragen und gestartet. In Echtzeit ist es dann möglich, auch unter Nutzung von externer Software verschiedene Profile zu erstellen und zu analysieren. Auch das Durchiterieren der einzelnen Assembler-Anweisungen ist möglich. Als externe Software ist dabei besonders die “TI EnergyTrace Technology” sowie der “Ultra-Low-Power-Advisor” zu erwähnen. Diese Softwarepakete werden durch das CCS unterstützt.

## **TI EnergyTrace Technology**

Das Werkzeug “TI EnergyTrace Technology” (ETT) ist ein Software-Paket zum Analysieren der Ressourcennutzung eines Microcontrollers. Diese Software ist eingebettet in das Code Composer Studio und kann dort wahlweise im Prozess des Debuggings benutzt werden. Die ETT wird dabei angeboten als sogenannter “real time power-profiler”. Es ist somit unter anderem möglich Kennzahlen sowie Diagramme zu der Power- und Voltage-Nutzung zu erhalten. Dabei ist es möglich einen gegebenen Programmcode zu vermessen. Dabei werden verschiedene Parameter innerhalb einer gegebenen Zeiteinheit gemessen. Diese Messung lässt sich speichern. Beispielsweise nach Optimierung des Programmcodes kann man diesen dann erneut vermessen. Das resultierende Profil lässt sich mit dem bereits persistierten Profil vergleichen. Als Ergebnis dieses Vergleiches ist nun eine Aussage darüber möglich, ob der Code in seiner Ressourcennutzung erfolgreich optimiert wurde. Besonders zu diesem Zweck wurde dieses Werkzeug innerhalb dieser Arbeit verwendet.

## **Ultra-Low Power Advisor**

Das Werkzeug “Ultra-Low Power Advisor” (ULPA) ist ein Software-Paket zum Analysieren des zugrundeliegenden Quelltextes auf zu optimierende Code-Stellen hinsichtlich der Ressourcennutzung. Der ULPA wird von der Entwicklungsumgebung CCS unterstützt und ist in diese eingebettet. Bei konkreter Betrachtung entwickelt der ULPA für den zugrundeliegenden Quelltext Empfehlungen / Hinweise. Darüber hinaus wird zu den jeweiligen Empfehlungen auch eine passende, jedoch abstrakte, nicht spezifische, Lösung

vorge stellt. Diese ist wahlweise einsehbar und kann gegebenenfalls unter projektspezifischer Anpassung übernommen werden. Sowohl die Hinweise, als auch deren abstrakte Lösungsvorschläge, werden dem Entwickler während der Entwicklung mitgeteilt. Besonders zu diesem Zweck wurde dieses Werkzeug innerhalb dieses Projektes verwendet.



# Ausgangssituation

Das Kapitel Ausgangssituation stellt anfänglich das Fallbeispiel mittels des Quelltextes vor. Da ein grobes Verständnis im Umgang mit der MSP430er Familie innerhalb dieser Arbeit vorausgesetzt wird, wird keine detaillierte Erklärung des gesamten Codes erfolgen. Stattdessen wird eine grobe Struktur gegeben und einige wichtige Zeilen im Detail erklärt. Anschließend wird es unter Benutzung etwaiger Werkzeuge hinsichtlich der Ressourcennutzung analysiert.

## Präsentation Fallbeispiel

Die Einleitung dieser Arbeit gibt bereits einen Überblick über die Funktionsweise des Automaten und somit über den groben Hintergrund des hier präsentierten Quelltextes. Einteilen lässt sich der Code in zwei Teile. Innerhalb des ersten Teils, Zeile 7-15, wird unter anderem der WatchDog-Timer ausgeschaltet und initial Register gesetzt. Durch diese Vorbereitung lassen sich später beispielsweise die grüne und rote LED als Output und der Button als Input nutzen. Wichtig innerhalb dieses Abschnittes ist die Definition von blinkMode und blinkMask. Die Variable blinkMode speichert dabei den aktuellen Zustand des Automaten. Die Variable blinkMask speichert eine sogenannte Maske. Das Ausgangspufferregister (P1OUT) wird in Phase 2 am Schleifenende mit besagter blinkMask geXORd. Dadurch kommt das blinken der LED zustande. In Teil zwei des Codes, Zeile 17-56, wird der Automat mit seinen Zuständen definiert. Wie schnell zu erkennen ist, hat der Automat vier

```
1 #include <msp430g2553.h>
2
3 #define LED BIT0
4 #define BUTTON BIT3
5
6 int main(void) {
7     WDCTL = WDTPW + WDTHOLD;
8     P1DIR = LED;
9     P1REN = BUTTON;
10    P1OUT = BUTTON;
11    P1DIR = BIT0 + BIT6;
12    int buttonPushed = 0;
13    int blinkMode = 0;
14    int blinkMask = BIT0;
15    P1OUT &= ~BIT6;
16
17    for( ; ; )
18    {
19        int j;
20        for( j = 0; j < 100; j++ )
21        {
22            volatile int i;
23            for( i = 0; i < 200; i++ );
24            if( ( P1IN & BIT3 ) == 0 )
25            {
26                if( !buttonPushed )
27                {
28                    buttonPushed = 1;
29                    blinkMode = (blinkMode + 1)%4;
30                    if( blinkMode == 0 ) {
31                        blinkMask = BIT0;
32                        P1OUT &= ~BIT6;
33                    }
34                    else if( blinkMode == 1 ) {
35                        blinkMask = BIT6;
36                        P1OUT &= ~BIT0;
37                    }
38                    else if( blinkMode == 2 ) {
39                        blinkMask = BIT0 + BIT6;
40                        P1OUT |= BIT0 + BIT6;
41                    }
42                    else
43                    {
44                        blinkMask = BIT0 + BIT6;
45                        P1OUT &= ~BIT0;
46                        P1OUT |= BIT6;
47                    }
48                }
49            }
50            else
51            {
52                buttonPushed = 0;
53            }
54        }
55        P1OUT ^= blinkMask;
56    }
57 }
```

Zustände. Dieser Code-Teil enthält drei Schleifen. Die äußerste Schleife, Zeile 17, ist eine Endlosschleife, welche den Automaten nicht enden lässt. Die LEDs werden bei Energiezufuhr stets weiter blinken. Die mittlere Schleife, Zeile 20, enthält einerseits eine weitere Schleife, die innerste Schleife, siehe Zeile 23. Diese Schleife hat keinen Inhalt und dient als eine Art “delay”-Funktion. Andererseits enthält die mittlere Schleife eine Prüfroutine, ob der Knopf gedrückt wurde oder nicht und führt den entsprechenden Code für die Funktionsweise des Automaten aus. Diese mittlere Schleife führt diese beiden Funktionen 100 mal aus. Erst nach diesen 100 Iterationen wird das Ausgaberegister mit blinkMask geXORd. Es ist somit festzuhalten das auch die mittlere Schleife auch eine zeitliche Verzögerung erzielt, also einen “delay” hervorruft.

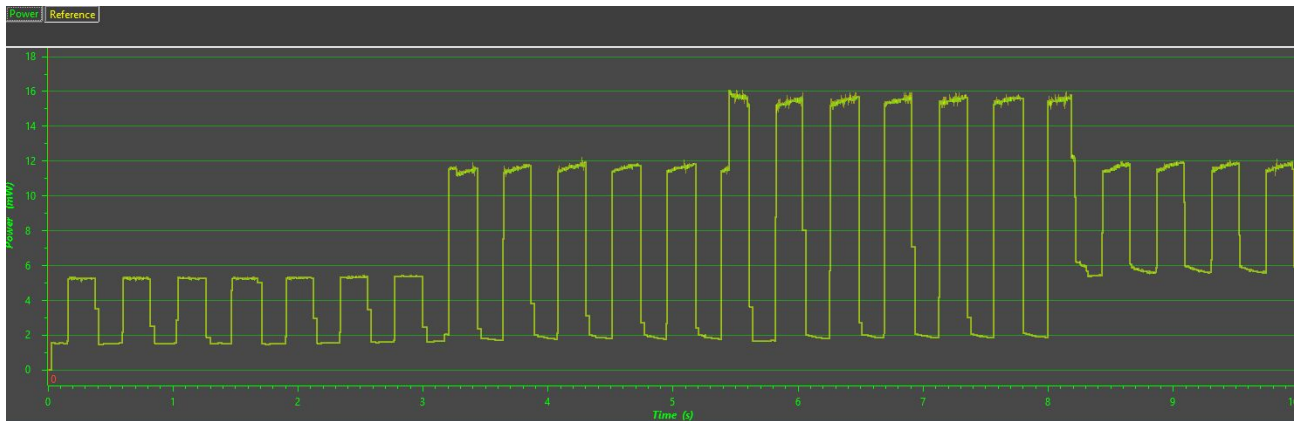
## Analyse Fallbeispiel

Die nun erfolgende Analyse des Ausgangscodes besteht aus mehreren Teilen. Diese werden nacheinander zusammengetragen.

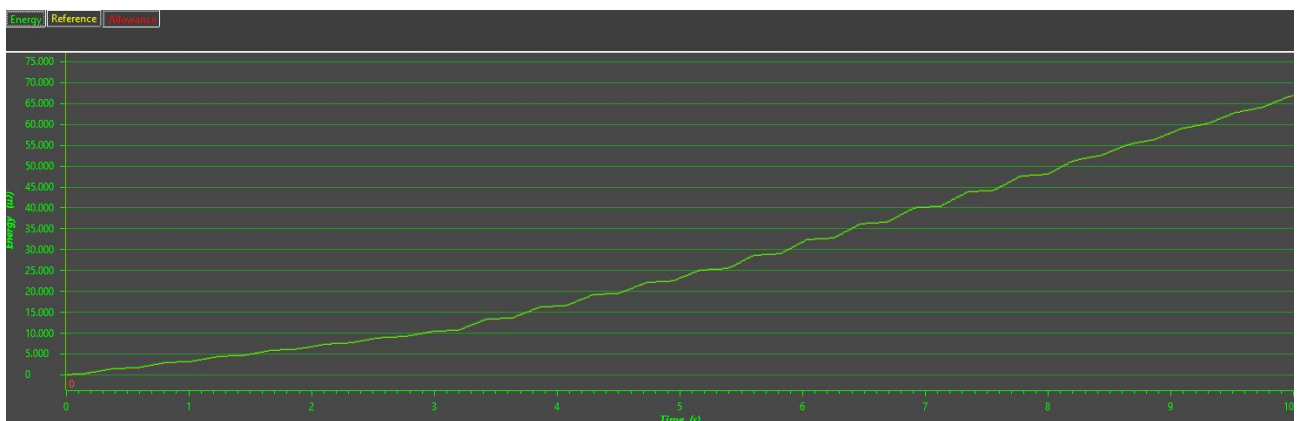
Die Analyse beginnt mit der Bestimmung des Ressourcenverbrauches unter Verwendung der ETT.

Anfänglich gilt es dabei auf den Ablauf der Vermessen einzugehen. Wie bereits erwähnt handelt es sich hierbei um einen Mealy-Automaten. Diese Automatenart reagiert bezüglich des Zustandswechsels auf externe Inputs, hier die Knopf-Eingabe. Daraus resultiert ein Problem bei dem Prozess des Messens, da es realistisch nicht möglich ist, als Verantwortlicher genau gleich lange Zustandsphasen des Automaten innerhalb eines Zeitabschnittes zu gewährleisten. Dieses Problem wurde erkannt. Es wurde demnach nach besten Wissen versucht innerhalb eines Messzeitraumes von 10 Sekunden, jede Zustandsphase des Automaten 2.5 Sekunden andauern zu lassen. Mit den gegebenen vier Phasen sind dadurch die 10 Sekunden der Messung ausgereizt. Darüber hinaus wurde der erste Codeteil (beschrieben in der Präsentation des Codebeispiels) übersprungen und mit Zeile 15 begonnen. Codeteil eins wurde wegen des initiiierenden Zweckes übersprungen.

Die Ergebnisse der Vermessung werden nun besprochen. Mit der ersten Grafik ist zusehen, dass die Leistung (Power) sich für jeden Zustand des Automaten unterscheidet.



An dieser Stelle gilt es den Begriff der Periode innerhalb dieser Arbeit zu definieren. Unter einer Periode wird hier das Einschalten einer oder mehrerer LED(-s) und zusätzlich das daraufhin erfolgende Ausschalten dieser LEDs verstanden. Es ist zusehen, dass Zustand drei dabei am Meisten Watt verbraucht. Dies liegt vorallem daran, da dieser Zustand beide LEDs gleichzeitig pro Periode einschaltet. Der Energieverbrauch liegt dabei nach folgender Grafik bei insgesamt ca. 67 mJ.



Zusammengefasst können wichtige Charakteristiken aus dem EnergyTrace Profile entnommen werden. Demnach ist zu vermerken, dass unter Benutzung einer entsprechenden Batterie eine Laufzeit von ca 98 Stunden zu erwarten ist

EnergyTrace™ Profile	
Name	Energy 1
▼ System	
Time	10 sec
Energy	66,819 mJ
▼ Power	
Mean	6,8344 mW
Min	0,0000 mW
Max	16,0203 mW
▼ Voltage	
Mean	3,2735 V
▼ Current	
Mean	2,0874 mA
Min	0,0000 mA
Max	4,8902 mA
Battery Life	CR2032: 4 day 2 hour (est.)

Im letzten Teil der Analyse des Fallbeispiels soll die genaue Frequenz der Periode in Herz erfasst werden. Diese Frequenzzahl ist für die kommende Optimierung des Programmes von entscheidender Bedeutung.

Bei genauerer Betrachtung des Leistungs- (Power-) Diagramms sind in 10 Sekunden 23 Perioden ersichtlich. Dies lässt eine grobe Orientierung auf 2.3 Hz zu. Bei genauerer Betrachtung wird nun auf Befehlsebene die Hz-Zahl berechnet. Mittels des CCS wurde durch Benutzung der sogenannten "Profile Clock" die Anzahl der CPU-Zyklen ermittelt. Danach benötigt eine einzige Iteration der mittleren Schleife 2223 CPU-Zyklen. Da lediglich wenige Zeilen Code vorliegen, könnte eine gewisse Verwunderung bei dieser Zahl eintreten. Bei genauerer Betrachtung ist jedoch festzustellen, dass eben nicht jede C-Instruktion und auch nicht jede Assembler-Instruktion gleich einem CPU-Zyklus ist. Zur Veranschaulichung wird folgend die Befehlsreihe in Assembler (ASM) eines Schleifendurchlaufes der innersten Schleife präsentiert.

```

c03a: 5391 0000      INC.W 0x0000(SP)
c03e: 90B1 00C8 0000  CMP.W #0x00c8,0x0000(SP)
c044: 3BFA          JL   ($C$L3)

```

Ein Durchlauf dieser innersten Schleife besteht aus drei Assembler-Befehlen. Bei genauerer Untersuchung hat sich dabei aber herausgestellt, dass darauf nicht 3 CPU-Zyklen resultieren. Tatsächlich benötigt die erste ASM-Instruktion 4 CPU-Zyklen, die zweite 5- und die letzte 2-Zyklen. Demnach benötigt eine Iteration der innersten Schleife

11 Zyklen. Diese Schleife wird 200x ausgeführt, wodurch insgesamt alleine 2200, zuzüglich initialer Befehle, von den 2223 beansprucht werden. Für eine halbe Periode (also die LED einmal einschalten) werden somit 100 (Iterationen der mittleren Schleife) \* 2223 + 1 (Jump-Befehl der äußeren Endlosschleife; wird nicht weiter berücksichtigt), also ca 222300 CPU-Zyklen benötigt. Wie bereits formuliert taktet die CPU mit einer Frequenz zwischen 0.8 MHz und 1.5Mhz (hängt beispielsweise von der Temperatur ab). Innerhalb dieser Arbeit wird sich zur Berechnung der Einfachheit halber auf 1 MHz festgelegt. Danach schafft die CPU  $1000000/222300$ , also ca 4.49 halbe Perioden in einer Sekunde. Das entspricht  $4.49/2$ , also ca 2.25, aufgerundet 2.3Hz. Dieser Wert deckt sich mit der Beobachtung des Diagramms.

# Optimierung

Das Kapitel Optimierung stellt Vorgehen und Umsetzung der Quelltextoptimierung hinsichtlich der Ressourcennutzung vor. Dabei werden nacheinander die Empfehlungen des ULPA besprochen. Anschließend findet erneut eine Auswertung statt. Diese untersucht dann erneut, welche Energieressourcen der Code beansprucht.

## Vorgehen und Umsetzung

Bei der Optimierung wurde sich besonders an den Empfehlungen des ULPA orientiert. Dieser zeigt bei Untersuchung des Ausgangscodes initial neun Hinweise an. Diese Hinweise werden nun nacheinander dargestellt, diskutiert und die eigene Lösung präsentiert.

1. (ULP 13.1) Detected loop counting up. Recommend loops count down as detecting zeros is easier, line 20

Dieser erste Hinweis zielt ab auf die mittlere Schleife, die momentan hoch- statt herunterzählt. Der abstrakte Lösungsvorschlag des ULPA ist somit:

```
for (i = 5000; i>0; i--) // In instead of: (i = 0; i <5000; i++)
```

Schon hier ist jedoch zusehen, dass diese Schleife zur Optimierung der Ressourcennutzung durch einen Interrupt-Mechanismus ersetzt wird. Dieser Interrupt-Mechanismus wird mit Punkt 9 im Detail erklärt. Im Code wird dieser Initialisiert mit Zeile 26. Die sogenannte Interrupt-Service-Routine, die eben die hier gegebene Schleife ersetzt, ist in Zeile 29-Ende zu sehen.

2. (ULP 13.1) Detected loop counting up. Recommend loops count down as detecting zeros is easier, line 23

Auch dieser Hinweis zielt auf das Herunterzählen statt Hochzählen ab. Diesmal wird jedoch die Schleife in Zeile 23 angesprochen. Durch den ULPA wird empfohlen:

```
for (i = 5000; i>0; i--) // In instead of: (i = 0; i <5000; i++)
```

Die eigene Lösung sieht jedoch wie folgt aus. Zweck dieser innersten Schleife ist die Einschaltbreite mit einer Verzögerung zu verlängern. Diesen "delay" müsste man theoretisch ebenfalls durch einen Interrupt-Mechanismus behandeln. Da bereits die mittlere Schleife durch einen Interrupt-Mechanismus ersetzt worden ist, wird dieses

Vorgehen an dieser Stelle schwieriger. Auf Empfehlung wird dazu Abstand genommen. Als eigene Lösung wird stattdessen diese Schleife gelöscht und die Verzögerung mittels des einen Interrupts realisiert. Dieser muss dementsprechend länger sein.

```

1 #include <msp430g2553.h>
2
3 const int ccr0_init = 6947; // 6947; //7812;
4 int buttonPushed = 0;
5 int blinkMode = 0;
6 int blinkMask = BIT0;
7
8 int main(void) {
9     WDTCTL = WDTPW + WDTHOLD;
10    P1DIR = 0xFF;
11    P1OUT = 0x00;
12    P2DIR = 0xFF;
13    P2OUT = 0x00;
14    P3DIR = 0xFF;
15    P3OUT = 0x00;
16    P1DIR = BIT0;
17    P1DIR = BIT6;
18    P1REN = BIT3;
19    P1OUT = BIT3;
20    P1DIR = BIT0 + BIT6;
21    BCSCCTL1 |= 0x87;
22    BCSCCTL2 |= DIVS_2;
23    TACTL = TASSEL_2 + MC_1 + ID_3;
24    CCTL0 = CCIE;
25    CCR0 = ccr0_init;
26    __bis_SR_register(LPM0_bits + GIE);
27 }
28
29 #pragma vector=TIMER0_A0_VECTOR
30 __interrupt void Timer_A (void)
31 {
32     if(( P1IN & BIT3 ) == 0 )
33     {
34         if( !buttonPushed )
35         {
36             buttonPushed = 1;
37             if( blinkMode == 0 ) {
38                 blinkMask = BIT0;
39                 P1OUT &= ~BIT6;
40                 blinkMode = 1;
41             }
42             else if( blinkMode == 1 ) {
43                 blinkMask = BIT6;
44                 P1OUT &= ~BIT0;
45                 blinkMode = 2;
46             }
47             else if( blinkMode == 2 ) {
48                 blinkMask = BIT0 + BIT6;
49                 P1OUT |= BIT0 + BIT6;
50                 blinkMode = 3;
51             }
52             else
53             {
54                 blinkMask = BIT0 + BIT6;
55                 P1OUT &= ~BIT0;
56                 P1OUT |= BIT6;
57                 blinkMode = 0;
58             }
59         }
60     }
61     else
62     {
63         buttonPushed = 0;
64     }
65     P1OUT ^= blinkMask;
66 }

```

### 3. (ULP 11.2) Assignment of higher bits (constants) to "P1OUT" within a loop.

Recommend using lower 4 bits to allow use of constant generators, line 40

Diese Empfehlung macht darauf aufmerksam, dass die höheren Bits des Ausgabenpufferregisters P1OUT nicht unbedingt innerhalb einer Schleife angesprochen werden sollten. Es wird von dem ULPA folgender abstrakter Lösungsvorschlag gebracht:

```

void main(void) {
    unsigned int i, variable=FLAG_1;
    P1DIR |= BIT2;
    P1DIR &= ~BIT2;
    for (i=0; i < 1000; i++) {
        P1DIR ^= BIT2;
    }
}

```

Wie bereits angesprochen werden alle Schleifen ersetzt durch eine Interrupt-Service-Routine (ISR). Mit diesem Vorgehen wird dieser Hinweis indirekt gelöst, da eben keine Schleifen mehr vorhanden sind.

4. (ULP 11.2) Assignment of higher bits (constants) to "P1OUT" within a loop.

Recommend using lower 4 bits to allow use of constant generators, line 46

Diese Empfehlung ist jener aus Punkt 3 sehr ähnlich. Demnach ist es auch hier so, dass die Schleifen ersetzt werden und somit dieser Hinweis indirekt gelöst wird.

5. (ULP 5.1) Detected modulo operation(s). Recommend moving them to RAM during run time or not using as these are processing/power intensive, line 29

Modulo-Operationen sind gerade für Microcontroller sehr anspruchsvoll. Bezogen wird sich dabei auf die Zeile

```
blinkMode = (blinkMode + 1)%4;
```

In diesem Sinne wird durch den ULPA empfohlen, diese Operation während der Laufzeit in den Hauptspeicher zu verlagern oder komplett zu ersetzen. Die Empfehlung des ULPA ist demnach:

1. Create a load = FLASH, run = RAM memory section in the device linker command file.

- Before RAM: origin = 0x2400, length = 0x2000, After: RAM\_EXECUTE: [...]

2. Relocate user-generated functions:

```
- #pragma CODE_SECTION(FunctionName, ".run_from_ram")
void FunctionName(void)
{ [...]
```

Ziel dieser Anweisung ist es, den aktuellen Zustand des Automaten zu speichern. Die Lösung dieser Empfehlung besteht nun darin, die komplette Modulo-Operation zu ersetzen. Ersetzt wurde diese Funktionalität, indem nun nicht mehr per Modulo-Operation der aktuelle Zustand gespeichert wird, sondern bei Eintritt in einen Zustand, der Folgezustand innerhalb von blinkMode gespeichert wird. Zusehen ist dies im Code beispielsweise in den Zeilen 40 und 45. Man sieht, dass bei Eintritt in den Zustand 0, direkt der Folgezustand, eben Zustand 1, mit Zeile 40, in der Variable blinkMode gespeichert wird. Dadurch wird die Modulo-Operation ersetzt und der Hinweis gelöst.



6. (ULP 2.1) Detected SW delay loop using empty loop. Recommend using a timer module instead, line 23

Hinweis sechs macht darauf aufmerksam, dass statt einer Verzögerungsschleife mit keinem Inhalt, besser ein timer-module eingesetzt werden sollte. Hintergrund ist, dass bei besagter Verzögerungsschleife stets die ressourcenbedürftige CPU rechnet. Mittels des angesprochenen Timer-Modules würde die CPU in einen Ruhemodus versetzt werden können, wodurch Ressourcen gespart werden. Der abstrakte Lösungsvorschlag des ULPA ist demnach:

```
void main(void) {
    WDTCTL = WDTPW + WDTHOLD;
    P1DIR |= 0x01;
    CCTLO = CCIE;
    CCR0 = 50000;
    TACTL = TASSEL_2 + MC_2;
    _BIS_SR(LPM0_bits + GIE);
}
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void) {
    P1OUT ^= 0x01;
    CCR0 += 50000;
}
```

Mit den vorhergehenden Beschreibungen wurde bereits erklärt, dass die angesprochene Schleife, in Zeile 23, gelöscht wird. Somit wird indirekt dieser Hinweis gelöst. Wie bereits erwähnt wird jedoch tatsächlich ein Timer-Modul für die mittlere Schleife eingesetzt. Dies wird unter Punkt 9 genauer erläutert.

7. (ULP 4.1) Detected uninitialized Port 3 in this project. Recommend initializing all unused ports to eliminate wasted current consumption on unused pins., line -

Hinweis sieben erklärt, dass Port 9 noch nicht initialisiert wurde, also trotzdem Energie verbraucht. Dabei wird durch den ULPA empfohlen diesen Port zu initialisieren und gegebenenfalls auszuschalten. Der abstrakte Lösungsvorschlag ist dabei:

```
P1DIR = 0xFF;
P1OUT = 0x00;
```

Nach diesem Vorschlag werden die Leitungen einerseits als Ausgangsleitungen initialisiert. Mit der zweiten Anweisung wird die zugehörige Ausgangsleitung auf LOW geschaltet. Es wird also wenig/keine Energie durch diesen ehemals nicht initialisierten, offenen Pin verbraucht. Die implementierte Lösung ist dabei in den Zeilen 14 und 15 mit

```
P3DIR = 0xFF;  
P3OUT = 0x00;
```

zu sehen.

8. (ULP 4.1) Detected uninitialized Port 2 in this project. Recommend initializing all unused ports to eliminate wasted current consumption on unused pins., line -  
Der Hinweis acht ist erneut sehr ähnlich dem Hinweis sieben. Hinweis und Lösungsvorschlag des ULPA stimmen somit überein. Die eigene Lösung ist in den Zeilen 12 und 13 zusehen.

9. (ULP 1.1) Detected no uses of low power mode state changing instructions, line -  
Durch Hinweis neun wird nun ein vergleichsweise großer Schwerpunkt angesprochen. Bei Gesamtbetrachtung ist offensichtlich, dass momentan kein Low-Power-Modus benutzt wird. Stattdessen wird unter anderem im aktuellen Code auf Schleifen zurückgegriffen, die eine Verzögerung generieren. Dem MSPg2553-Controller ist es dabei möglich sich in einen sogenannten Low-Power-Modus zu versetzen, bei dem deutlich weniger Energie verbraucht wird. Als abstrakten Lösungsansatz stellt der ULPA dabei folgenden Code vor.

```
void main(void){  
    BCSCCTL1 |= DIVA_2;  
    WDTCTL = WDT_ADLY_1000;  
    IE1 |= WDTIE;  
    P1DIR = 0xFF;  
    P1OUT = 0;  
    P2DIR = 0xFF;  
    P2OUT = 0;  
    while(1) {  
        int i;  
        P1OUT |= 0x01;  
        for (i = 5000; i>0; i--);  
    }
```

```

        P1OUT &= ~0x01;
        __bis_SR_register(LPM3_bits + GIE);
    }
}

#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer (void) {
    __bic_SR_register_on_exit(LPM3_bits);
}

```

Bei Verwendung eines Low-Power-Modus wird unter anderem die CPU in einen Ruhezustand versetzt. Es ist somit logisch, dass eine Art Interrupt vorliegen muss, um die CPU aus diesem Ruhezustand zu holen, den entsprechenden Code innerhalb einer Interrupt-Service-Routine (ISR) auszuführen und danach gegebenenfalls erneut in den Ruhezustand zu versetzen. Im Lösungsvorschlag des ULPA ist zu erkennen, dass dazu ein Interrupt mit dem Low-Power-Modus 3 implementiert wurde. Dieser basiert auf dem WatchDog-Timer. Daraus abgeleitet stellt sich die umgesetzte Lösung wie folgt dar.

Innerhalb der eigenen Lösung wurde statt des WatchDog-Timers, das Timer-Modul A benutzt. Dabei wurde auf den Low-Power-Modus 0 zurückgegriffen, siehe Zeile 26. Dieser schaltet bei Umschalten in den Ruhemodus den Prozessor und dessen "clock", die MCLK oder "Master Clock" aus. In Betrieb verbleiben die Uhren SMCLK ("Sub-Main-Clock") und die ACLK ("Auxiliary-Clock"). Bezüglich der Uhr wird als letztes die Quelle für die SMCLK-Uhr in Zeile 21 mit dem BCSCTL1-Register gewählt. Die Zahl 0x87 lässt sich in Binärcode schreiben als 1000 0111. Man sieht, dass das letzte Bit, Bit 7 gesetzt ist. Mit diesem Bit wird der XT2-Oscillator ein- oder ausgeschaltet. Da dieses Bit auf 1 gesetzt wurde, wird dieser ausgeschaltet. Dadurch wird Chip-intern auf den DCO-Oscillator (DCO) zurückgegriffen. Dieser DCO hat eine Frequenz von ungefähr 1 Mhz. Die anderen Bits der Zahl 0x87 - Zahl setzten lediglich Bits, die bereits per Default gesetzt wurden.

In Zeile 23 ist unter anderem zu sehen, dass mit der TASSEL\_2 - Konstante Timer-A-Modul mit der SMCLK-Uhr konfiguriert wurde. In gleicher Zeile wird außerdem der sogenannte continuous-up-mode mit den Bits der Konstante MC\_1 gewählt. Demnach wird mit dem Timer, der mit der Uhr SMCLK arbeitet, die wiederum durch den DCO gespeist wird, bis zum Capture-Control-Register-0 (CCR0) hochgezählt. Falls der Wert des CCR0 erreicht worden ist, wird mit dem MC\_1-Mode erneut von 0 angefangen und wieder hochgezählt. Es ist also von entscheidender Bedeutung, welche Zahl dem CCR0

zugewiesen wird, diese Bestimmt die Frequenz der LED. Da nun der DCO eine Frequenz von ca 1 MHz hat, wurden 2 sogenannte Divider benutzt. Diese reduzieren die Inputfrequenz durch einen Divisor. Benutzt wird dabei als erstes ein Divisor in der SMCLK Uhr in Zeile 22. Da Timer-A zur Benutzung der SMCLK-Uhr konfiguriert wurde, wird diese resultierende Frequenz an diesen weitergeleitet. Dieser benutzt jedoch einen zweiten Divisor, wie in Zeile 23 zu erkennen ist. Dort wird im TACTL-Register das Bit ID\_3 gesetzt. Das teilt die Inputfrequenz wiederum um 8. Mit diesen Zwei Informationen ist es nun möglich den Wert für CCR0 zu berechnen.

Zur Bestimmung des CCR0-Wertes existieren zwei pseudo-unabhängige Wege. Einerseits kann ausgehend von der Anzahl der Schritte für die zu ersetzende Schleife CCR0 berechnet werden. Andererseits kann CCR0 auch ausgehend von dem Basistakt berechnet werden.

Ausgehend von der Anzahl der Instruktionen in der Schleife der Basisimplementierung ergibt sich dabei folgende Berechnung. Instruktionen gab es ungefähr 222300 bei einer halben Periode. Das heißt es müssen durch den Timer 222300 übersprungen werden, um eine ähnliche zeitliche Verzögerung zu generieren. So lässt sich Rechnen  $222300 / 8$  (timer-a-input-divider) / 4 (SMCLK Divider) = 6946.875, gerundet also 6947 und grob gerundet 6900. Demnach muss der Wert von CCR0 auf 6947 initialisiert werden um eine halbe Periode in vergleichbarer Zeit durchzuführen. Wie sich nach kurzer grafischer Analyse des Power-Diagramms herausstellt, liefert dieser Wert ebenfalls eine Periode von ca 2.3 Hz. Die Rechnung ist somit korrekt.

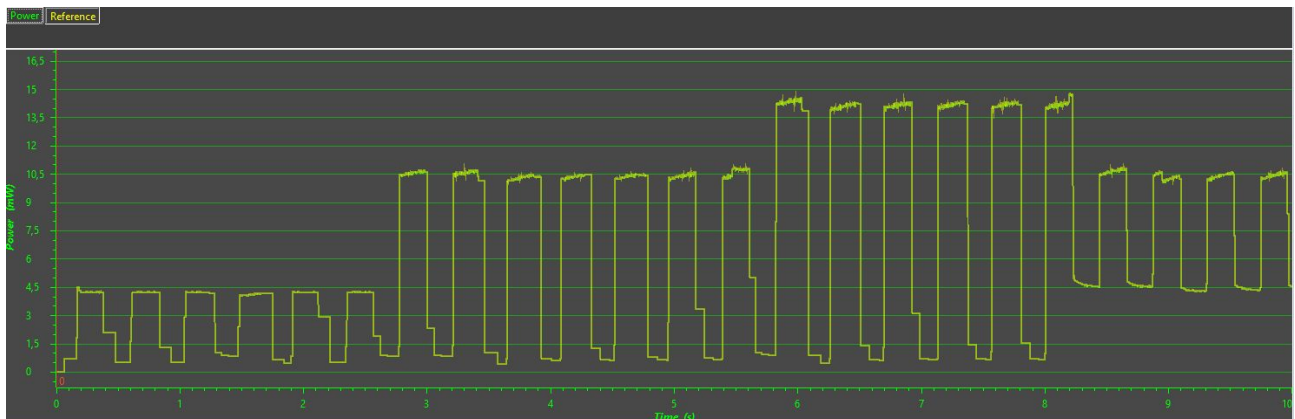
Ausgehend von der Herz-Zahl der MCU ergibt sich eine andere Rechnung. Demnach lässt die der Wert für CCR0 wie folgt bestimmen:  $1000000$  (Frequenz der CPU) / 8 (timer-a-input-divider) / 4 (smclk-divider) / 2.3 (angestrebte Frequenz) / 2 (da eine Periode 2 Durchläufe benötigt und hier nach einer halben Periode gefragt ist) = 6793.47, grob gerundet 6800. Demnach muss der Wert von CCR0 ausgehend von der Frequenz der CPU auf 6800 gesetzt werden. Dieser Wert sich ungefähr gleich, jedoch etwa geringer zu dem Wert aus Rechnung 1. Daraus lässt sich schlussfolgern, dass die Annahme von 1 MHz der CPU nicht ganz korrekt ist. Wie zu erkennen ist, schafft die CPU einen gewissen Prozentsatz an Instruktionen mehr pro Sekunde. CCR0 wurde im Code in Zeile 3 und 25 gesetzt.

Die Zeilen 29 bis Ende beschreiben die ISR mit dem bekannten Code des Automaten.

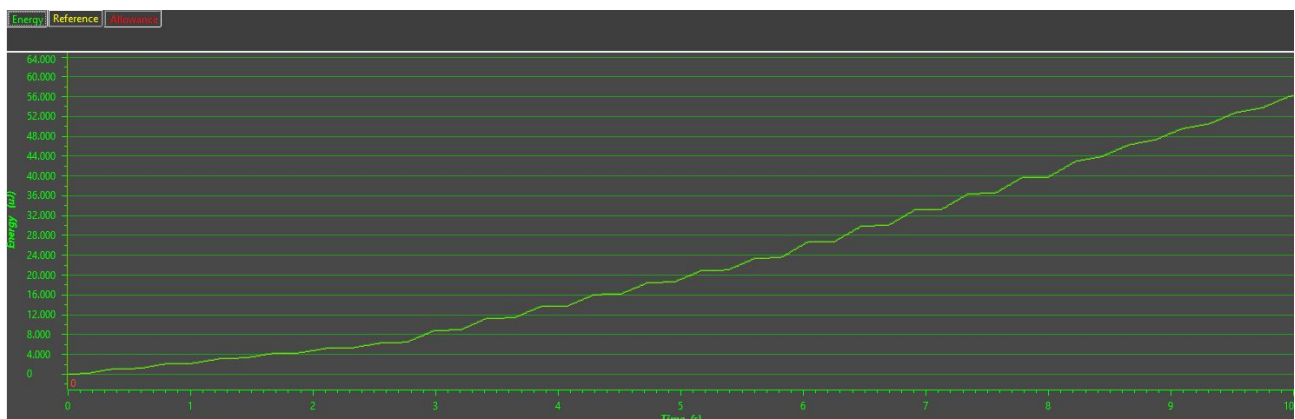
## Analyse

Bei der Auswertung des optimierten, nun effizienten Code wurden ebenfalls auf bestimmte Umstände Rücksicht genommen. Einerseits wurde hier ebenfalls der Codeteil der Initialisierung übersprungen. Es wurde also ab Zeile 26 gemessen. Andererseits wurde auch hier versucht eine jede Zustandsphase des Automaten ca. 2.5 Sekunden andauern zu lassen.

Die Ergebnisse der Analyse werden nun mit folgenden Diagrammen präsentiert. Das erste Diagramm zeigt dabei die beanspruchte Leistung (Power) des Microcontrollers. Es ist hier zusehen, dass bei ausgeschalteter LED die Watt nahe 0 sind, also kaum Ressourcen verbraucht werden.



Das Diagramm bezüglich der verbrauchten Energie in Joule zeigt darüber hinaus die Energieaufnahme über den Vermessungszeitraum. Dieser steigt beinahe linear an.

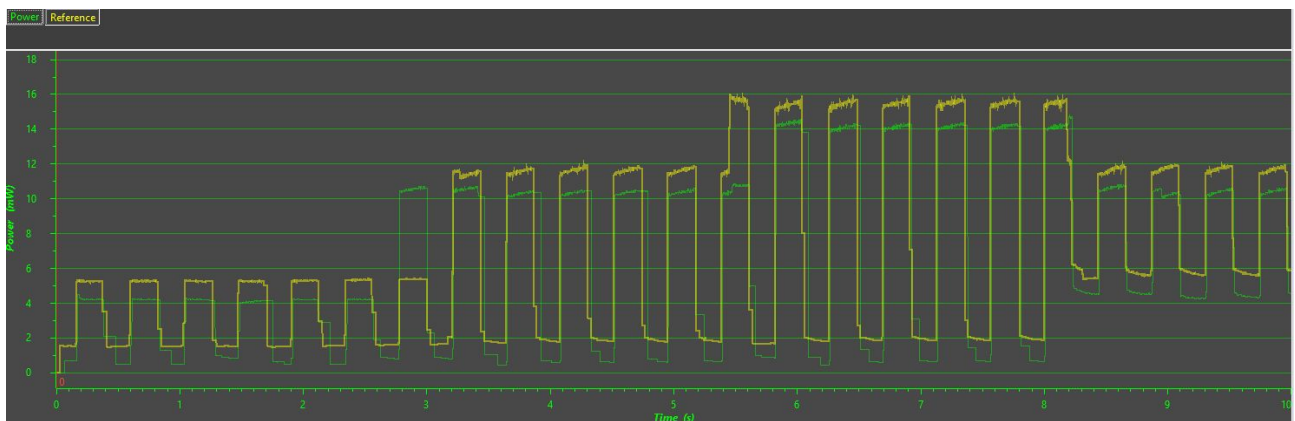


Zusammengefasst werden die gemessenen Charakteristiken in der Folgeabbildung. Zusätzlich zu den bisherigen Informationen ist hier interessant, dass bei Verwendung einer angemessener Batterie, eine Laufzeit von ungefähr 117 Stunden zu erwarten ist.

EnergyTrace™ Profile	
Name	Energy 1
▼ System	
Time	10 sec
Energy	56,175 mJ
▼ Power	
Mean	6,1339 mW
Min	0,0000 mW
Max	14,8148 mW
▼ Voltage	
Mean	3,2735 V
▼ Current	
Mean	1,8735 mA
Min	0,0000 mA
Max	4,5236 mA
Battery Life	CR2032: 4 day 21 hour (est.)

# Vergleich

Der Vergleich beider Ressourcenprofile erfolgt ebenfalls mittels der EnergyTrace Technology. Bezüglich der Leistung Leistung ergibt sich dabei folgender Vergleich.










Wie bereits mehrmals erwähnt, ergibt sich bei der Vermessung immer eine Ungenauigkeit, da die Zustände per Knopfdruck gewechselt werden. Auch aus diesem Grund ist die nur ungefähre Überlappung der Zustände zu erklären. Bei der Auswertung ist jedoch schnell erkennbar, dass das Profil des optimierten Codes im Allgemeinen weniger Watt benötigt als der Basiscode. Dies ist Ursache der Optimierung und beweist deren Erfolg. Die Energieaufnahme der LEDs ließ sich jedoch nicht verringern. Es ist somit bezüglich der Leistungsaufnahme der LEDs kein Unterschied festzustellen.

Der gesamte Energieverbrauch lässt sich auch aus dem Folgediagramm ablesen. Hier ist deutlich zu erkennen, dass der Effizient-Code weniger Joul verbraucht als der Inefficient-Code.



Zusammengefasst werden diese ersten Eindrücke durch die Übersicht des Profilvergleiches. Danach wurde unter anderem ca 15.9% weniger Energie verbraucht. Dies ist auch ein Grund dafür, dass nun eine mögliche passende Batterie ca 18.9% länger den Microcontroller versorgen kann als im Vergleich zum Inefficient-Code.

EnergyTrace™ Profile				
Name	Energy 1	Delta (%)		Energy 2
▼ System				
Time	10 sec			10 sec
Energy	56,175 mJ		-15,9	66,819 mJ
▼ Power				
Mean	6,1339 mW		-10,2	6,8344 mW
Min	0,0000 mW			0,0000 mW
Max	14,8148 mW		-7,5	16,0203 mW
▼ Voltage				
Mean	3,2735 V		-0,0	3,2735 V
▼ Current				
Mean	1,8735 mA		-10,2	2,0874 mA
Min	0,0000 mA			0,0000 mA
Max	4,5236 mA		-7,5	4,8902 mA
Battery Life	CR2032: 4 day 21 hour (est.)		18,9	CR2032: 4 day 2 hour (est.)



## Zusammenfassung und Ausblick

Innerhalb dieser Arbeit wurde ein bestehender Code in seiner Ressourcennutzung erfolgreich optimiert. Dazu fand ein "Reverse Engineering" statt, wobei der Basiscode analysiert und teilweise bis auf seine CPU-Zyklen zurückgeführt wurde. Das Wissen aus dieser Analyse wurde benutzt, um einen optimierten Code, den Efficient-Code, zu erstellen. Dabei wurden unter anderem unbenutzte Ports initialisiert und LOW-geschaltet sowie Modulo-Rechnungen ersetzt. Besondere Beachtung fand bei der Optimierung die Implementierung eines Timers, geknüpft an eine ISR in Verbindung mit dem Einsatz eines Low-Power-Modes. Nach Umsetzung wurde der optimierte Code analysiert. Dabei wurde festgestellt, dass dieser ca 15.9% weniger Energie benötigt und auch aufgrund dessen ca 18.9% länger, bei angemessener Batterie, versorgt werden kann.

Bezüglich des Ausblicks wird eine mögliche weitere Verbesserung vorgestellt, die eventuell zukünftig umgesetzt werden kann. So wurde innerhalb dieser Arbeit auf den Low-Power-Modus 0 gesetzt. Es ist jedoch vorstellbar, dass der Low-Power-Modus 3 eingesetzt werden könnte. Dieser würde über die CPU und die "MCLK" hinaus noch die SMCLK und die DCO ausschalten. Dadurch könnten eventuell noch weitere Ressourcen gespart werden.

# Anhang

## Anhang A: Code Fallbeispiel-Inefficient

```
1 #include <msp430g2553.h>
2
3 #define LED BIT0
4 #define BUTTON BIT3
5
6 int main(void) {
7     WDTCTL = WDTPW + WDTHOLD;
8     P1DIR = LED;
9     P1REN = BUTTON;
10    P1OUT = BUTTON;
11    P1DIR = BIT0 + BIT6;
12    int buttonPushed = 0;
13    int blinkMode = 0;
14    int blinkMask = BIT0;
15    P1OUT &= ~BIT6;
16
17    for( ; ; )
18    {
19        int j;
20        for( j = 0; j < 100; j++ )
21        {
22            volatile int i;
23            for( i = 0; i < 200; i++ );
24            if(( P1IN & BIT3 ) == 0 )
25            {
26                if( !buttonPushed )
27                {
28                    buttonPushed = 1;
29                    blinkMode = (blinkMode + 1)%4;
30                    if( blinkMode == 0 ) {
31                        blinkMask = BIT0;
32                        P1OUT &= ~BIT6;
33                    }
34                    else if( blinkMode == 1 ) {
35                        blinkMask = BIT6;
36                        P1OUT &= ~BIT0;
37                    }
38                    else if( blinkMode == 2 ) {
39                        blinkMask = BIT0 + BIT6;
40                        P1OUT |= BIT0 + BIT6;
41                    }
42                    else
43                    {
44                        blinkMask = BIT0 + BIT6;
45                        P1OUT &= ~BIT0;
46                        P1OUT |= BIT6;
47                    }
48                }
49            }
50            else
51            {
52                buttonPushed = 0;
53            }
54        }
55        P1OUT ^= blinkMask;
56    }
57 }
```

## Anhang B: Code: Code Fallbeispiel-Efficient

```
1 #include <msp430g2553.h>
2
3 const int ccr0_init = 6947; // 6947; //7812;
4 int buttonPushed = 0;
5 int blinkMode = 0;
6 int blinkMask = BIT0;
7
8 int main(void) {
9     WDTCTL = WDTPW + WDTHOLD;
10    P1DIR = 0xFF;
11    P1OUT = 0x00;
12    P2DIR = 0xFF;
13    P2OUT = 0x00;
14    P3DIR = 0xFF;
15    P3OUT = 0x00;
16    P1DIR = BIT0;
17    P1DIR = BIT6;
18    P1REN = BIT3;
19    P1OUT = BIT3;
20    P1DIR = BIT0 + BIT6;
21    BCSCTL1 |= 0x87;
22    BCSCTL2 |= DIV5_2;
23    TACTL = TASSEL_2 + MC_1 + ID_3;
24    CCTL0 = CCIE;
25    CCR0 = ccr0_init;
26    __bis_SR_register(LPM0_bits + GIE);
27 }
28
29 #pragma vector=TIMER0_A0_VECTOR
30 __interrupt void Timer_A (void)
31 {
32     if(( P1IN & BIT3 ) == 0 )
33     {
34         if( !buttonPushed )
35         {
36             buttonPushed = 1;
37             if( blinkMode == 0 ) {
38                 blinkMask = BIT0;
39                 P1OUT &= ~BIT6;
40                 blinkMode = 1;
41             }
42             else if( blinkMode == 1 ) {
43                 blinkMask = BIT6;
44                 P1OUT &= ~BIT0;
45                 blinkMode = 2;
46             }
47             else if( blinkMode == 2 ) {
48                 blinkMask = BIT0 + BIT6;
49                 P1OUT |= BIT0 + BIT6;
50                 blinkMode = 3;
51             }
52             else
53             {
54                 blinkMask = BIT0 + BIT6;
55                 P1OUT &= ~BIT0;
56                 P1OUT |= BIT6;
57                 blinkMode = 0;
58             }
59         }
60     }
61     else
62     {
63         buttonPushed = 0;
64     }
65     P1OUT ^= blinkMask;
66 }
```