Klimagerecht programmieren

Prof. Dr. em. Vesselin Iossifov
Fachbereich Energie und Information
Hochschule für Technik und Wirtschaft Berlin
Vesselin.Iossifov@htw-berlin.de | http://www.f1.HTW-Berlin.de

Mitglied des Vorstandes
VDE/ETV e.V. Berlin-Brandenburg
www.vde-etv-berlin.de

In Memoriam

Dr. -Ing. habil. Horst Schwetlick
Vorstandsvorsitzender des Elektrotechnischen Vereins
Berlin-Brandenburg,
Bezirksverein des VDE

Kurzfassung

Die Energiekosten der Rechenzentren von Google, Amazon & Co. übersteigern schon im zweiten Nutzungsjahr die Gerätekosten, Laptops und Smartphones wären niemals wirklich “tragbar”, kleine elektrisch getriebene Drohnen würden niemals nutzbar fliegen können und die Batterien unserer Funkmäuse müsste man mindestens ein Mal pro Woche austauschen, wenn Prozessoren und Chipsätze nicht energieeffizient programmierbar wären. Welche Richtung die Entwicklung von Prozessoren seit 2000 und ihre energieeffiziente Programmierung nahm, so dass die o. g. Dienstleistungen und Geräte seit geraumer Zeit vernünftig nutzbar sind? Der Vortrag schildert Techniken einer energieeffizienten Anwender-Programmierung in der Prozessmesstechnik und zeigt, wie diese am Beispiel von Mikrocontrollern gestaltet werden kann. **Dem Vortrag wird eine Vorführung der o.g. Techniken angeschlossen**.

Schlüsselworte: Programmiertechniken zur Optimierung der Energieaufnahme von Rechenanlagen, **Embedded Systems und Anwendungen (Industrie 4.0, IoT)**, x86 basierte Rechner.

Motivation

Neben in den letzten 120 Jahren in logarithmisch-linearer **xexp2 * log x** [08] Progression wachsende Menschheit mit ihren Bedürfnissen nach Nahrung, Wasser, Wohnen, Energie, Transport, Medizin etc.(**1**), der globalisierten Energie-, Rohstoff-, Industrie- und Landwirtschaft, die ihre Erzeugnisse über zehntausenden von Kilometern per Schiff, Luftfracht und 40-Tonner liefern (**2**) und dem interkontinentalen Flug- und Kreuzfahrtschiff-Tourismus (**3**) bilden das **Internet-IT** und die darauf basierenden **digitalen Medien** die **vierte Säule** der größten CO2 Emittenten menschlichen Wirtschaftens. Das logarithmisch-lineare **xexp1.1*log x** [09] Wachstum der Säulen **2** bis **4** in den letzten 30 bis 60 Jahren bedingt das Wachstum der Säule **1** – ein System mit positiver Rückkopplung.

Das kann nicht gut gehen.

Die digitalen Medien (**4% „greenhouse gas“ (GHG) 2020**) lassen **doppelt so viel CO2** produzieren als die zivile Luftfahrt (**2% GHG 2018**) und die **Hälfte** der GHG verursacht durch **ALLE PKW** und **KRAD** dieser Welt (**8% GHG**) [05, S.17,]. **"Streamen ist das neue Fliegen"!** Junge Menschen in den westlichen Industrieländer können sich das Leben heute gut ohne Fleisch, Autos und Kohlestrom vorstellen. Aber ohne digitale Medien und Fliegen [04]?

Ein Beispiel, basierend auf eine aktuelle medienpolitische Aktivität in den digitalen Medien und durchgerechnet mit den Erkenntnissen aus [05] soll die o.g. Aussagen quantifizieren und neben den Empfehlungen aus [05] meinem Vortrag die notwendige Aktualität geben:

- YouTube-Video „Die Zerstörung der CDU.“, https://youtu.be/4Y1lZQsyuSQ, **Länge 54:57 Minuten**, bis zum 07.07.2019 **15.491.330** Aufrufe, bis zum 07.07.2019 **973750** Abonnenten des YouTube-Kanals.

Aufgabenstellung:

Wie viel **Kilogramm CO2** hat das Abspielen dieses Videos als Internet-basiertes digitales Medium mit den o.g. Parametern erzeugen lassen? Für einen User, für alle **15.491.330** User? Wie Lange könnte ein User mit dem Diesel-Äquivalent dieser Energiemenge Auto fahren?

Lösungsweg:

In [05] werden die Mengen des „**greenhouse gas“** (**GHG) CO_2** über alle Instanzen der Lebens und der Nutzungskette der Komponenten der internetbasierten Digitalen Medien, beginnend bei der Rohstoffgewinnung über die Nutzungszeit bis zum Recyceln der Endgeräte, verfolgt.

1. Aus [05, S. 72,] entnehmen wir, dass ***The action "Watch a video film online"*** für ***10 Minuten*** folgende Datentransferrate aufweist: „***high quality video 1,080p: 170 MByte", ca. 2E+08 MByte***.

2. Aus [05, S. 72, Fußnote [57]] entnehmen wir die „***Calculation methodology: "1 byte" model***".
Für Streamen eines Videos von 10 Min in der **EU** werden **9E-02 kWh** Energie am Laptop (ohne Router) verbraucht und **3E-02 $KgCO_2$** (ohne Router) emittiert.

3. Verbrauchte Energie und erzeugten CO2 Emission pro Nutzer:
$E_{\text{YouTube Rezo single user}}$ = 5,5* 9E-2 KWh = 49,5E-02 kWh oder ca. **500Wh**
Das ist die Energie, um **2 Stacks** bei 2000W 15 Min zu braten.
$GHG_{\text{YouTube Rezo single user}}$ = 5,5*3E-02 $KgCO_2$ = 16,5E-02 $KgCO_2$ oder ca. **165 gCO_2**
Das ist die GHG Emission für **1 Km** Vollgasfahrt mit meinem E4-Diesel PKW auf der Autobahn.

4. **15,491330 * 10 exp6** Rezo-User haben folgende Energie verbraucht und GHG emittiert :

$E_{\text{YouTube Rezo community}}$ = 49,5E-02 kWh * 15,5E+06 = 7,6725E+06 kWh = 7,6725 *GWh

$GHG_{\text{YouTube Rezo community}}$ = 16,5E-02 $KgCO_2$ * 15,5E+06 = 2,5575E+06 $KgCO_2$ = 2558 Tonnen CO_2

Nach H.-W. Sinn in [10] sind **5,85 GW** die *jahresdurchschnittliche* elektrische Windstromleistung in jeder Stunde für 24256 WKA in 2014 hochgerechnet aus den Daten der Fa. 50Hertz [10] . Ich habe die elektrische Energie, die ca. **30000 Windkrafträder** in Deutschland 2019 im *Jahresdurchschnitt* für **eine Stunde** liefern auf ca. **7,00 GWh** hochgerechnet. Die *YouTube Rezo community* überbietet mit ihren Energiebedarf diesen Wert um ca.**10%**, wenn der gesamte Energiebedarf in Deutschland abgerufen wäre.

5. **V**(Diesel) $_{\text{YouTube Rezo community}}$ = **7,6725E+06 kWh /10 kWh** [6] = **767250** Liter Diesel

Mein PKW verbraucht für 12T Km/6.5 L/100 Km pro Jahr = 780 L Diesel

Videoenergieverbrauch der REZO Community als Fahräquivalent in Km = **ca. 12 000 000 Km**

oder **273 Mal um den Äquator mit meinem Auto** oder **1000 Diesel-PKW mit 12T Km/Jahr** Laufleistung.

([06] Energie für 1 Liter Diesel: ca. 10 kWh)

6. Aus [04, S. 2 bis 6] entnehmen wir, dass die Altersgruppe der **14 bis 29-jährigen** in Deutschland mit einem Konsum von digitalen Medien von **ca. 6 Stunden täglich** mit einer Stärke von **ca. 14 Millionen** Personen im Jahr 2017 in [11] angegeben wird. Wir wollen nun mit den aus [05] entnommenen Daten für pro Byte/Kg und ggf. Jahr den Energieverbrauch und GHG- Emission der o.g. Altersgruppe durch Nutzung von digitalen Medien berechnen (**Laptop, LAN-Anschluß, ohne Router**) :

6.1 **$E_{AG14\text{-}29\ single\ User\ pro\ Stunde}$ = 6* 9E-02 kWh = 54E-2 kWh** oder ca. **540 Wh pro Stunde**

$GHG_{AG14\text{-}29\ single\ User\ pro\ Stunde}$ = 6* 3E-02 $KgCO_2$ = 18E-02 $KgCO_2$ oder ca. 180 gCO_2 pro Stunde

6.2 **$E_{AG14\text{-}29\ single\ User\ pro\ Tag}$ = 6*6* 9E-02 kWh = 324E-2 kWh** oder ca. **3,24 kWh pro Tag**

$GHG_{AG14\text{-}29\ single\ User\ pro\ Tag}$ = 6*6* 3E-02 $KgCO_2$ = 108E-02 $KgCO_2$ oder ca. 1,080 $KgCO_2$ pro Tag

6.3 **$E_{AG14\text{-}29\ single\ User\ pro\ Jahr}$ = 365*6*6* 9E-02 kWh = 1,1826 MWh pro Jahr**

$GHG_{AG14\text{-}29\ single\ User\ pro\ Jahr}$ = 365*6*6* 3E-02 $KgCO_2$ = 394,2 $KgCO_2$ pro Jahr

Stromverbrauch meiner 3-köpfigen Familie (ohne AG 14-29) in einem EFH für 2018: **2,745 MWh/Jahr**

6.4 **$E_{AG14\text{-}29\ Community\ pro\ Tag}$ = 14E+06*6*6* 9E-02 kWh = 45,36E+06 kWh**

45,36 GWh pro Tag entsprechen 3,15% des E-Energie Verbrauchs in D 2018 pro Tag (1,44 TWh) [12].

6.5 **$E_{AG14\text{-}29\ Community\ pro\ Jahr}$ = 365 * 45,36E+06 kWh = 16,556E+09 kWh**

16,56 TWh pro Jahr, 38% mehr als die E-Energie für Verkehr in D 2017 [13].

V(Diesel) $_{AG14\text{-}29\ Community\ pro\ Jahr}$ **= 16,56E+09 kWh/10 kWh** [06] = **1,6556E+9 Liter Diesel**

oder die Energie zum Betreiben von 2,12 Millionen Diesel-Autos mit einer Laufleistung von 12T Km/Jahr.

6.6 **GHG (**Betrieb mit Router**)**

$GHG_{AG14\text{-}29\ Community\ pro\ Jahr}$ = 14E+06*365*6*6* 3E-02 $KgCO_2$ + 14E+06*28 $KgCO_2$ /Jahr

5,912 Megatonnen CO_2 pro Jahr, 7,5% des in D emittierten CO_2 pro Jahr [14].

Quo vadis, CO_2 Emittent digitale Medien?

Könnte es uns wie den Menschen im Alten Testament ergehen ?

1. Mose 6,5-9,17 „Aber die Erde war verderbt vor Gott und voller Frevel. Da sah Gott auf die Erde, und siehe, sie war verderbt; denn alles Fleisch hatte seinen Weg verderbt auf Erden."

2. Mose 6,5-9,17 „Mache dir einen Kasten von Tannenholz und mache Kammern darin und verpiche ihn mit Pech innen und außen. Und mache ihn so: Dreihundert Ellen sei die Länge, fünfzig Ellen die Breite und dreißig Ellen die Höhe. Ein Fenster sollst du für den Kasten machen obenan, eine Elle groß. Die Tür sollst du mitten in seine Seite setzen. Und er soll drei Stockwerke haben, eines unten, das zweite in der Mitte, das dritte oben."

Als Informatiker verstehe ich Gottes Botschaft so:

Gott zeigte sich Noah und sprach zu ihm: **„Mach' back-up, denn ich werde formatieren!“**

Sollten wir aus Angst vor einer möglichen Sintflut infolge des Klimawandels einen zivilisatorischen Selbstmord begehen?

© Ivan Kutuzov-Kuti, 31.07.19, Zeitung Trud, Bulgarien.

Wir sollen Wege zur Minimierung des Energieverbrauchs der IT suchen und finden.

1. Programmiertechniken für eine energieeffiziente Softwareentwicklung

Anwendungen von Computer- und Kommunikationssystemen in zivilen Feldern können heute wie folgt aufgeteilt werden: **industrielle Anwendungen** (IoT, Industrie 4.0), **Geschäftsanwendungen** (Banken, Handel, Transportlogistik, Bürokommunikation) und **digitale Medien** (digitale Post, Telefonie, Kurz-nachrichtendienste, Bilder, Audio- und Videostreaming). Der Energieverbrauch ist ein Schlüssel-faktor beim Entwurf der o.g. Systemen. Als die Computerarchitekturen in den letzten 20 Jahren die Möglichkeit in verschiedenen Energie-Modi betrieben zu werden bekamen, stellten diese eine neue Herausforderung an den Anwender-Algorithmen und deren Codierung zur Minimierung der Energieaufnahme dar. Im Vortrag möchte ich Methoden zur Minimierung der Energieaufnahme in **industriellen Anwendungen**, die bei der Codierung von Basisalgorithmen der Automatisierungstechnik angewandt werden, vorstellen. Dieser Vortrag **verifiziert** und **diskutiert** die Ergebnisse der Texas Instruments Fallstudie zu Low-Power Programmierung ***„MSP430™ Advanced Power Optimizations: ULP Advisor™ Software and EnergyTrace™ Technology“*** [01].

2. Die Fallstudie von Texas Instruments "MSP430™ Advanced Power Optimizations"

Die Fallstudie zu Low-Power Programming Techniken von Texas Instruments Inc. mit den Werkzeugen **C**ode **C**omposer **S**tudio **Software Environment (CCS), U**ltra**L**ow**P**ower **(ULP) Advisor™ software** und **EnergyTrace technology™** nutzt das Hardware Kit **MSP-EXP430FR5969 LaunchPad™**. Die erste Fallstudie beginnt mit einer energie-ineffizienten Version, der üblichen Codierung in C eines prozess-messtechnischen Algorithmus von Temperatur über dem ADC des Mikrocontrollers. Die Werkzeuge **UltraLowPower Advisor™** und **EnergyTrace Technology™** werden eingesetzt, um die Energieeffizienz dieser Lösung mit den Parametern **Strom, Spannung** und **Energie** numerisch und grafisch zu quantifizieren. Basierend auf den Code-Analysen und -Empfehlungen der beiden Werkzeugen werden neue Programmiertechniken eingesetzt, um eine **zweite** Version des Algorithmus zu entwickeln. Der Analyseprozess wird wiederholt und mit den vom **ULP Advisor™** empfohlenen Techniken wird eine **dritte**, sehr energieeffiziente Code-Version, entwickelt mit den ersten beiden verglichen und die Energieausbeute wird festgestellt.

Das Software-Werkzeug **UltraLowPower Advisor™** nutzt die Ergebnisse des C-Compilers, um auf Energieineffizienzen des entwickelten Codes bezogen auf die Low-Power Fähigkeiten des Mikrocontrollers hinzuweisen und Lösungen aus einer Bibliothek von ULP-Regeln für diese anzubieten. Sie helfen dem Entwickler, seinen Code entsprechend der Low-Power Fähigkeiten des Mikrocontrollers TI MSP430 weiterzuentwickeln. Der Entwickler erkennt eigene energieineffiziente Codesequenzen und lernt energieeffiziente Programmiertechniken. Voraussetzung dafür sind detaillierte Kenntnisse der Struktur und Funktion des Mikrocontrollers mit ULP-Fähigkeiten. Das ist die **erste neue Herausforderung** an den Entwicklern von MSR-Algorithmen in der Automatisierungstechnik. Die **zweite neue Herausforderung** stellt das Einsetzen der vom **ULP Advisor™** empfohlen neuen ULP-Programmiertechniken dar.

EnergyTrace technology™ ist das Werkzeug mit dem die Echtzeit-Messung der physikalischen Größen Strom, Spannung, Energie und Zeit und die Visualisierung mehrerer interner Zustände der CPU im Debug-Prozess des Codes durchgeführt wird.

3. Das Werkzeug TI EnergyTrace technology™

3.1 Methode der Energiemessung

Elektrische Leistungsmessung wird üblicherweise mit Hilfe des Stromdurchflusses durch ein Widerstand bei anliegender Spannung zu einem diskreten Zeitpunkt bestimmt. Die **EnergyTrace technology™** wendet eine neue Methode dafür an. Die Debugger-Schaltung des Kits **MSP-EXP430FR5969 LaunchPad™** mit einem softwaregesteuerten DC-DC Wandler generiert die Stromversorgung des Kits (1.2 V-3.6 V). Das Frequenzspektrum der mit dem DC-DC Wandler generierten Energieimpulse entspricht dem Energieverbrauch des Mikrocontrollers. Der eingebaute Kalibrator definiert den Energieäquivalent für einen einzelnen Ladeimpuls. Die Breite eines jeden Ladeimpulses bleibt dabei konstant. Der Debugger zählt jeden Ladeimpuls. Die Summe aller Ladeimpulse zusammen mit der verstrichenen Zeit werden zur Berechnung des durchschnittlichen Stromdurchflusses herangezogen. So wird auch die kürzeste Aktivität des Mikrocontrollers zum gesamten Energieverbrauch herangezogen.

3.2 EnergyTrace Modi

Der **EnergyTrace Modus** erlaubt die Energiemessung am Mikrocontroller. Die Stromversorgung des Mikrocontrollers wird permanent vom Debugger analysiert, um Daten zum Energie- und Leistungsaufnahme zu gewinnen. In diesem Modus kann der Energieverbrauch in Folge des Programmablaufs ermittelt werden.

Der **EnergyTrace++ Modus** liefert Informationen über den Energieverbrauch und der Low-Power Zustände der CPU (LPM0 to LMP4.5, Active Mode). Diese Zustände beschreiben den Status der Peripheriefunktionsgruppen und alle Systemtakte sowie den gerade aktiven Low-Power Mode. Mit diesem Werkzeug kann überprüft werden, ob der Code in jeder Zeile das erwartete Verhalten von CPU und Peripherie bewirkt und dass jede Peripheriegruppe nach durchgeführter Aktivität wieder abgeschaltet wird.

3.3 Grenzen der Energie- und Leistungsmessung

Es ist keine direkte Korrelation zwischen der Daten, die **EnergyTrace++** liefert und den Energieverbrauch an einer einzelnen Codeziele oder Maschinenbefehl möglich.

4. Die Fallstudie – ein Element aus IoT

Die Verifikation der Möglichkeiten der Werkzeuge **ULP Advisor™, EnergyTrace technology™** und des **MSP-EXP430FR5969** Kits durch Code bewirkte Energie- und Leistungsaufnahme zu analysieren und diese für die Minimierung dieser Werte zu optimieren ist Ziel der Studie.

4.1 Die Fallstudie

Drei Quellcode Dateien werden inspiziert. In jeder Datei wird die selbe **IoT-Funktion** ausgeführt – jede Sekunde wird eine ADC Wandlung eines Temperaturwertes durchgeführt. Dieser wird in Celsius und Fahrenheit berechnet und das Ergebnis über die UART Schnittstelle übertragen.

Folgende Analyseschritte werden durchgeführt:

- Einsatz des ULP Advisor™
- Generierung von EnergyTrace Technology™ Profilen
- Messung der absoluten Leistungsaufnahme

4.2 Feedback vom ULP Advisor™

Die **Advice window** in **CCS** zeigt Empfehlungen zu Low-Power Optimierung für `Inefficient.c`.

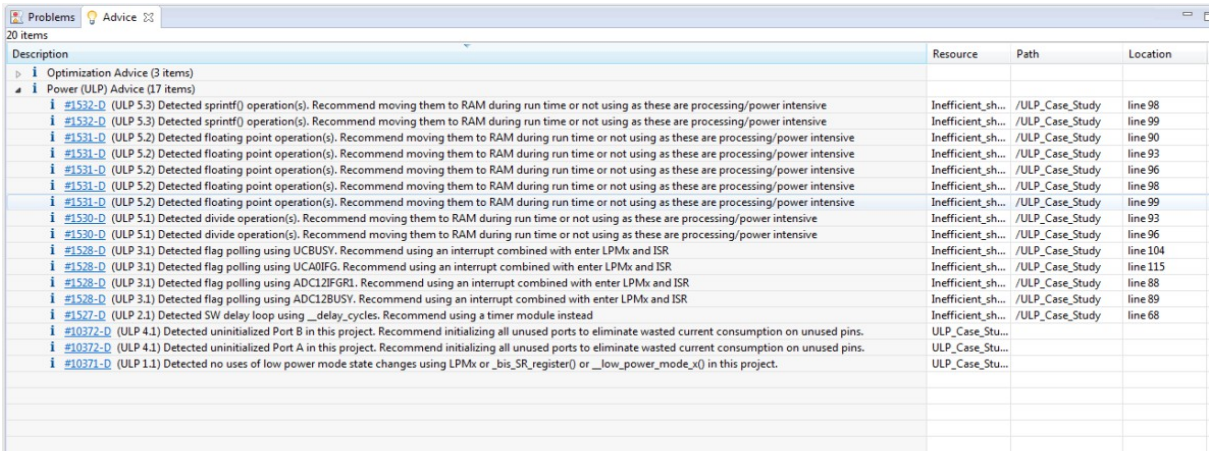


Bild 1: ULP Advisor™ Feedback für `Inefficient.c`

Die Liste gibt einige Empfehlungen vom ULP Advisor™ zu Low-Power Regelverletzungen:

- (ULP 5.2) Detected **floating point** operations.
- (ULP 5.1) Detected **divide operations.**
- (ULP 2.1) Detected **software delay loop using** _delay_cycles.
- (ULP 4.1) Detected **uninitialized port** in this project.
- (ULP 1.1) Detected **no uses of low-power mode state**

4.3 Referenz Profil mit EnergyTrace technology™ erzeugen

Um Codelösungen energetisch mit EnergyTrace technology™ zu bewerten, zeigen Bild 2 und 3

Laufzeitinformationen zu Energieverbrauch, Leistungs- und Stromfluss, die während der Low-Power und Active Modi für CPU und Peripheriebaugruppen innerhalb von 10 Sekunden Laufzeit gemessen wurden.

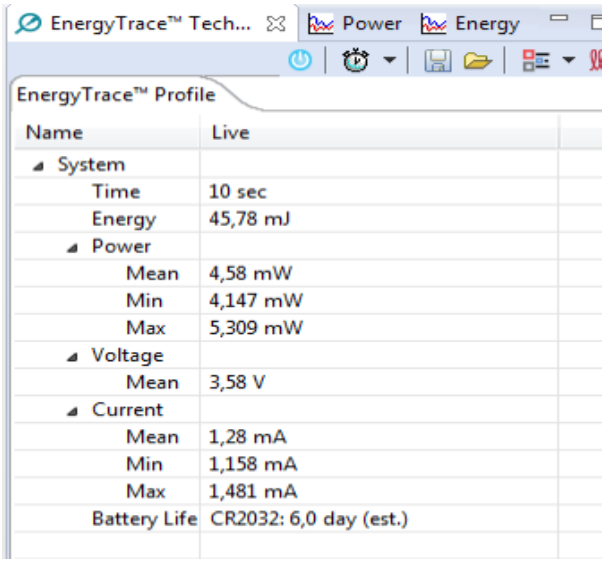


Bild 2: `Inefficient.c` Energie, ~Leistung, ~Strom

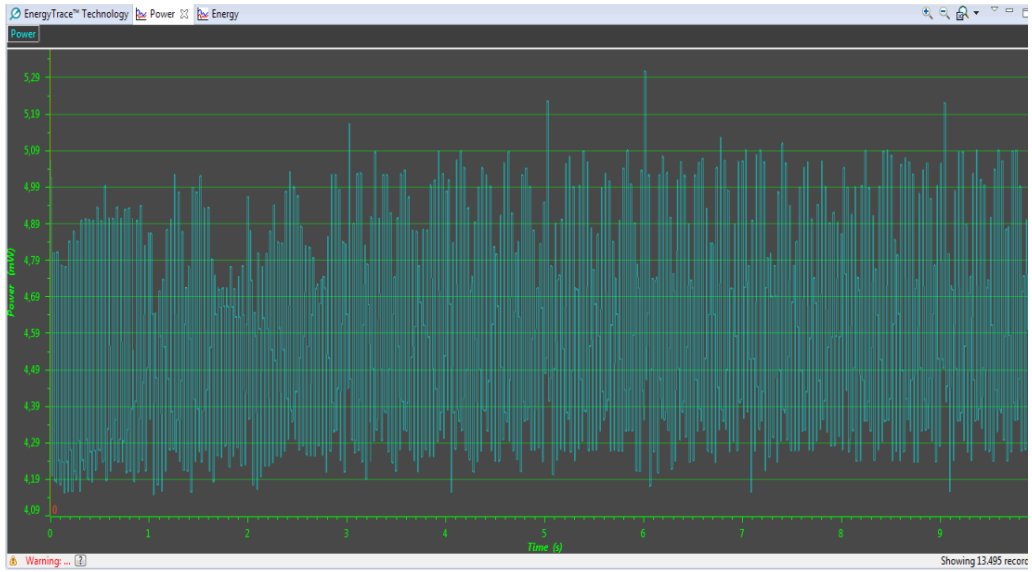


Bild 3: `Inefficient.c` Leistungsprofil

In der Leistungsgrafik (Bild 3) korrespondieren die Amplituden der Leistungsaufnahme mit den Zeitpunkten der ADC Temperaturmessung. Die Energiegrafik zeigt den Energieverbrauch innerhalb von 10 Sekunden der Code Ausführung.

Die **Energiegrafik** zeigt den Energieverbrauch	**Zustandsdiagramm** - Zustände der MPU- und Peripherie-Power Modi und Systemtakte

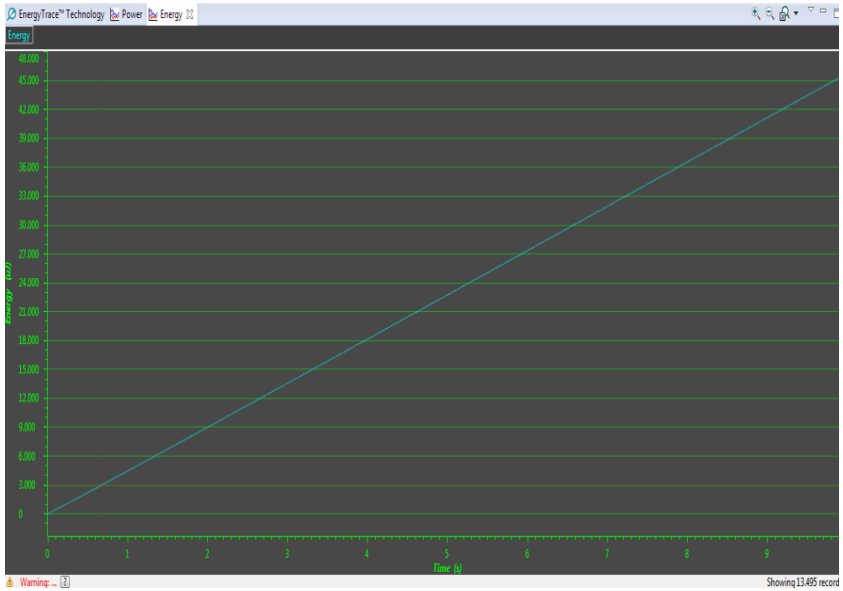
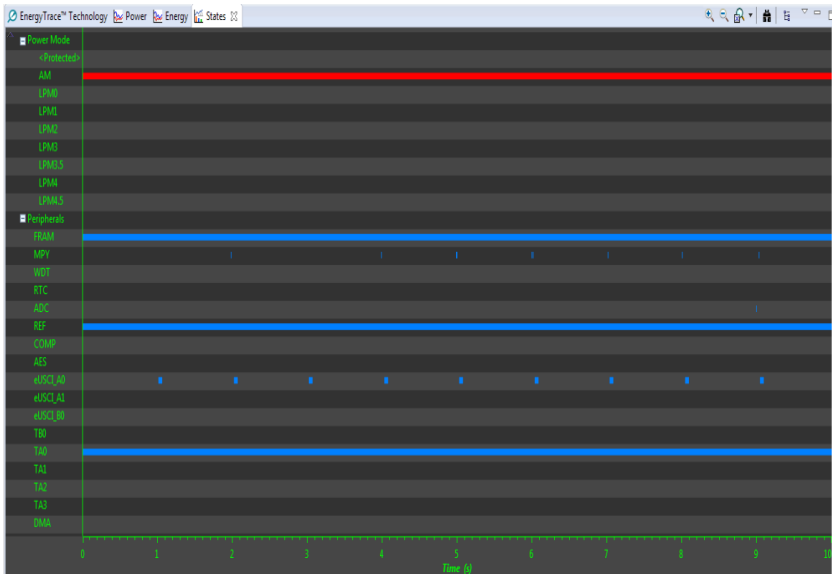

	

Bild 4: `Inefficient.c` Energie	Bild 5: `Inefficient.c` MCU/Interface Zustände

4.4 Ziele der energetischen Codeoptimierung

`Inefficient.c`	`Efficient.c`
sprintf()	Standardfuntion wird mit einer optimierten Zeichenkettenverarbeitung ersetzt.
Floating point operations	Ersetzen durch Integer-Calculus
Divide operations	Keine Änderung
Flag polling	Einsatz von Interrupts
Software delays	Einsatz von Timer
Floating port pins	Ungenutzte Output-Pins auf “NULL” setzen
No use of low-power modes	Einsatz von **L**ow**P**ower**M**odes 1 bis 3
Counting up in loops	Keine Änderung

Tabelle 1: Low-Power Optimierungsempfehlungen für `Inefficient.c` vom **ULP Advisor™**

Wege zur energetischen Codeoptimierung : **Setzen von ungenutzten Output Pins auf "NULL"**

Port pins	Set unused pins to output low
//Configure used port pin P2SEL1 |= BIT1 | BIT0; // Configure UART pins P2SEL0 &= ~(BIT1 | BIT0); // Configure UART pins PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode //Programming techniques **//Unused port pins with random values**	//**Configure all port pins output low** PAOUT = 0; PBOUT = 0; PJOUT = 0; PADIR=0xFFFF; PBDIR = 0xFFFF; PJDIR = 0xFF; PAIFG = 0; PBIFG = 0; //Programming techniques **//Unused port pins set to output low** // Configure used port pins P2SEL1 |= BIT1 | BIT0; // Configure UART pins P2SEL0 &= ~(BIT1 | BIT0); // Configure UART pins PM5CTL0 &= ~LOCKLPM5; // Disable GPIO power-on default high-impedance mode
Inefficient.c	Efficient.c

Tabelle 2: Setzen von ungenutzten Output Pins auf "NULL"

4.4.1 Besondere Beachtung der Nutzung von Low-Power Modi

Eine besondere Herausforderung der Programmierung in Low-Power Modi ist die Nutzung der dazu passenden Systemtakten wie unten angegeben. Programmtechnisch wird die MPU im LPM 3 versetzt. Das Zustandsdiagramm in **EnergyTrace** zeigt aber LPM1 an. So kann die Unvollkommenheit im Codes schnell gefunden werden: die MPU hat doch nicht das erwartete LPM 3 angenommen. Um das zu erreichen, muss ACLK-Takt programmiert werden. MCLK und SMCLK werden im LPM 3 deaktiviert. Ein Blick im Code zeigt, dass ADC, Timer und UART alle SMCLK nutzen. Diese Taktart bedingt LPM 1. Um LPM3 zu nutzen muss anders programmiert werden (Tabelle 3). Drei System-Taktarten stehen zur Verfügung und sind über Software wählbar. Der Nutzer kann so die beste Balance zwischen Rechenleistung und Energieverbrauch wählen:

- Auxiliary clock: ACLK is software selectable for individual peripheral modules
- Master clock: MCLK is divided by 1, 2, 4, or 8. MCLK is used by the CPU and system
- Sub-main clock: SMCLK is software selectable for individual peripheral modules.

Software Zeitkonstanten via Timer: Keine Nutzung von Low-Power Modi	Konfigurieren und Nutzen des Timers: Via Interrupt und LPM3
// Configure 1sec Timer TA0CTL = TASSEL__SMCLK | ID__8 | MC__UP | TACLR | TAIE; // SMCLK / 8, up mode, clear timer TA0EX0 = TAIDEX_7; //(SMCLK / 8) / 8 ~ 15.625 kHz. Default SMCLK: 1MHz TA0CCR0 = 0x3D09; // ~1 sec //Programming techniques **//No use of low-power modes**	**// Configure Timer using LPM3** TA0CCR0 = 6; // ~0.4ms TA0CCTL0 |= CCIE; // Capture/compare interrupt enable. __bis_SR_register(LPM3_bits | GIE); // Enter LPM3. Delay for Ref to settle. TA0CCR0 = 0x3D09; // Change timer delay to ~1 sec. //Programming techniques **//Uses LPM 1 to 4**
Inefficient.c	Efficient.c

Tabelle 3: Zeitkonstanten im Timer setzen

Zustandsdiagramm ohne Nutzung von Low-Power Modi (Bild 7) und mit Low-Power Modi (LPM1, Bild 6)

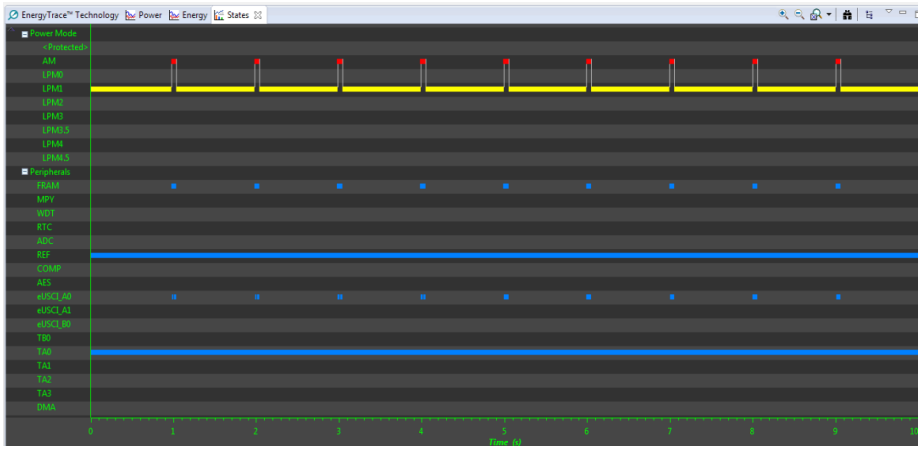


Bild 6: `Efficient.c` MPU/Interface Zustände

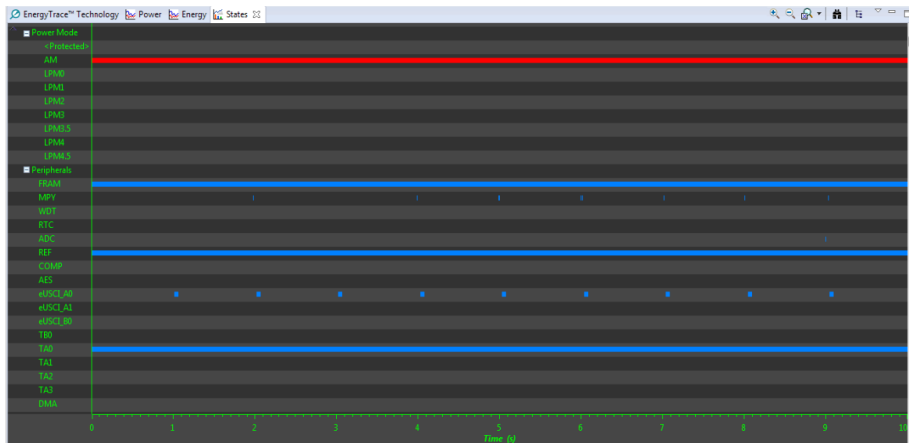


Bild 7: `Infficient.c` MPU/Interface Zustände

Wege zur energetischen Codeoptimierung: **ADC via Interrupt und LPM3 nutzen**

Configure/use ADC: via while()-Loop	**Configure/use ADC: via interrupt and LPM3 (1)**
`while(1)` `{` `  if(TA0IV == TA0IV_TAIFG)` `// Poll the timer overflow interrupt status` `  {` `  ADC12CTL0 |= ADC12ENC | ADC12SC;` `// Sampling and conversion start` `  while(ADC12IFGR1 & ~ADC12IFG30);` `// Wait for the conversion to complete` `  while(ADC12CTL1 & ADC12BUSY);` `  temp = ADC12MEM0;` `// Temperature in Celsius` `// Temperature in Fahrenheit` `  }` `//more code` `}` `//Programming techniques` **`//Flag polling`**	`while(1)` `{` `  __bis_SR_register(LPM3_bits | GIE);` `// Timer activation vie interrupt` `// Enter LPM3, wait for ~1sec timer` `  ADC12CTL0 |= ADC12ENC | ADC12SC;` `// Sampling and conversion start` `  __bis_SR_register(LPM3_bits | GIE);` `// Wait for conversion to complete` `  temp = ADC12MEM0;` `// 'temp' = the raw ADC temperature conv. result` `  __bic_SR_register(GIE);` `// Timer activation vie interrupt` `// Temperature in Celsius` `// Temperature in Fahrenheit` `}` `//Programming techniques` **`//Use of interrupts wherever possible`**
Inefficient.c	Efficient.c

Tabelle 4: Konfigurieren und Nutzen des ADC

Wege zur energetischen Codeoptimierung: **ADC via Interrupt und LPM3 nutzen**

Configure/use ADC: Via while()-Loop	**Configure/use ADC: Via interrupt and LPM3 (2)**
`while(1)` `{` `  if(TA0IV == TA0IV_TAIFG)` `// Poll the timer overflow interrupt status` `  {` `  ADC12CTL0 |= ADC12ENC | ADC12SC;` `// Sampling and conversion start` `  while(ADC12IFGR1 & ~ADC12IFG30);` `// Wait for the conversion to complete` `  while(ADC12CTL1 & ADC12BUSY);` `  temp = ADC12MEM0;` `// Temperature in Celsius` `// Temperature in Fahrenheit` `  }` `//more code` `}`	`#pragma vector = ADC12_VECTOR` `__interrupt void ADC12_ISR(void)` `{` `  switch(__even_in_range(ADC12IV,76))` `  {` `  //more code` `  case ADC12IV_ADC12IFG0:` `  // Vector 12:  ADC12MEM0` `  ADC12IFGR0 &= ~ADC12IFG0;` `  // Clear interrupt flag` `  __bic_SR_register_on_exit(LPM3_bits);` `  // Exit active CPU` `  break;` `  //more code` `  }` `}`
Inefficient.c	Efficient.c

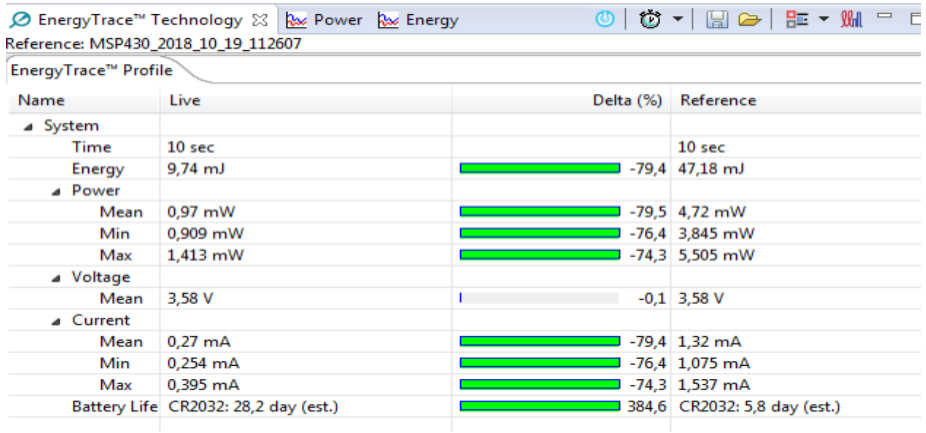
Tabelle 5: Konfigurieren und Nutzen des ADC

4.4.2 Code analysieren im ULP Advice window

Nur ein kleiner Rest von Empfehlungen bleibt im **ULP Advice window** nach Kompilieren des in `efficient.c` umbenannten weiterentwickelten Code von `inefficient.c`.

4.4.3 Vergleich der Energieprofile von `inefficient.c` und `efficient.c`

Der **EnergyTrace window** zeigt den Vergleich der Energieparameter der beiden Codeversionen (Bild 8). Die Grafik links zeigt das Ergebnis der Minimierung der Größen Energie, Leistung und Stromdurchfluß (Bild 9, 10).



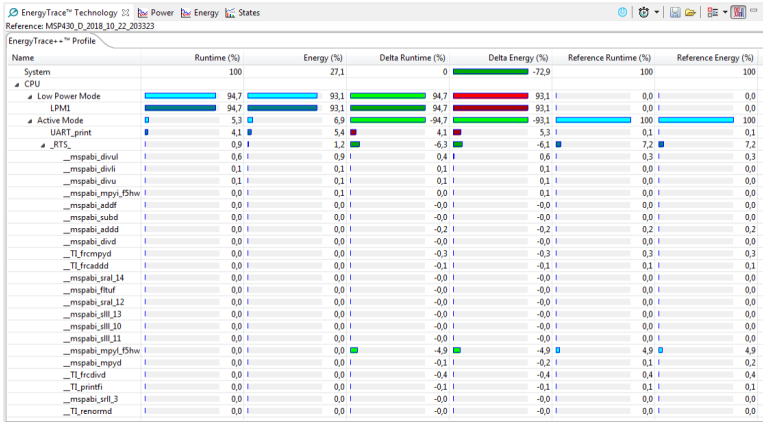


Bild 8: `Efficient.c` Vergleich mit `Inefficient.c`

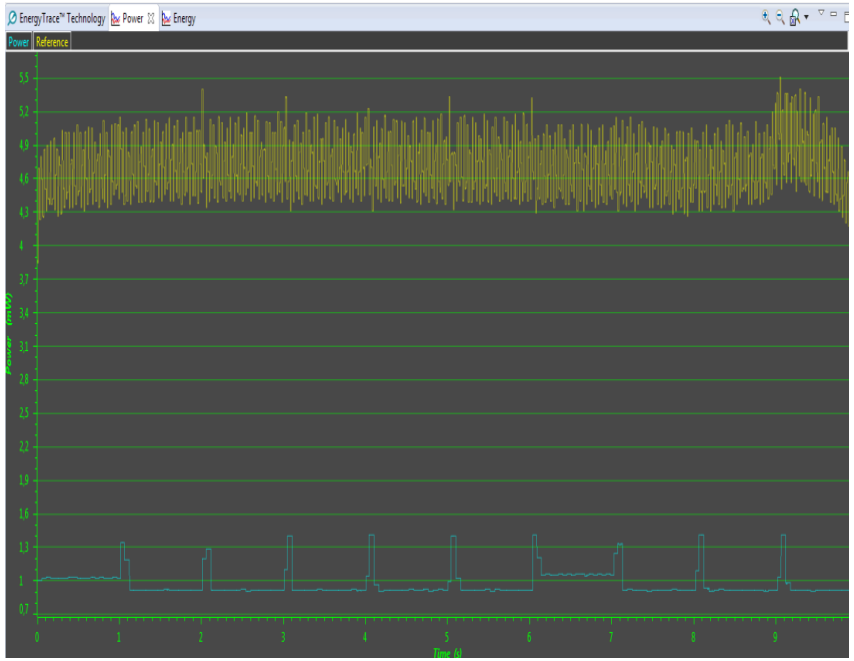


Bild 9: `Efficient.c` Leistung (Blue) Vergleich mit `Inefficient.c` Leistung (yellow)

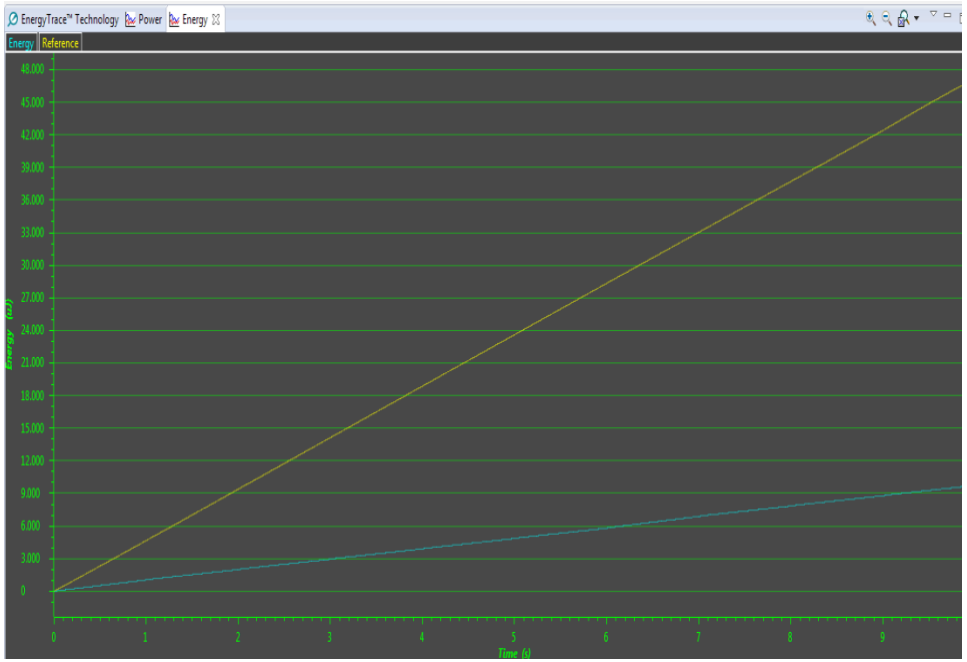


Bild 10: `Efficient.c` Energie (Blue) Vergleich mit `Inefficient.c` Energie (yellow)

4.5 Ziele der energetischen Codeoptimierung

`Efficient.c`	`MostEfficient.c`
sprintf()	Standardfuntion wird mit einer optimierten Zeichenkettenverarbeitung ersetzt.
Floating point operations	Ersetzen durch Integer-Calculus
Divide operations	Einsatz von Reziproken Werten außerhalb der Schleife. Einsatz von MUL, ADD und Shift statt intDIV.
Flag polling	Einsatz von Interrupts
Software delays	Einsatz von Timer
Floating port pins	Ungenutzte Output-Pins auf “NULL” setzen
No use of low-power modes	Einsatz von **L**ow**P**ower**M**odes 1 bis 3
Counting up in loops	Runterzählen zu NULL in Schleifen

Tabelle 6: Low-Power Optimierungsempfehlungen für `Efficient.c` zu `MostEfficient.c` vom ULP Advisor™

Wege zur energetischen Codeoptimierung: **Clock System settings für LPM**

Default Clock System settings	Configure Clock System settings for LPM
//Programming techniques **//Configure clocks** Three clock signals are available and software selectable. The user can select the best balance of performance and low power consumption: • ACLK: Auxiliary clock. ACLK is software selectable for individual peripheral modules. • MCLK: Master clock. MCLK is divided by 1, 2, 4, or 8. MCLK is used by the CPU and system. • SMCLK: Sub-main clock. SMCLK is software selectable for individual peripheral modules.	**// Configure Clock System** CSCTL0_H = 0xA5; // CS password CSCTL2 = SELA__LFXTCLK; // ACLK sourced from LFXT CSCTL3 = DIVA__1; // No division CSCTL4 |= LFXTDRIVE_3 | SMCLKOFF | VLOOFF; // Highest crystal drive setting. MAY CHANGE SMCLK turned off. CSCTL4 &= ~LFXTOFF; // Turn on LFXT do { CSCTL5 &= ~(LFXTOFFG | HFXTOFFG); // Clear XT1 fault flag SFRIFG1 &= ~OFIFG; }while (SFRIFG1 & OFIFG); // Test oscillator fault flag **//Programming techniques** **//Configure clocks for low power consumption**
Efficient.c	MostEfficient.c

Tabelle 7: Konfiguration der Clock System settings für LPM

Wege zur energetischen Codeoptimierung: **Timer nutzen via Clock System für LPM**

Configure Timer by default timer configuration	**Configure and use Timer via clock system for LPM**
// Configure Timer TA0CTL = TASSEL__SMCLK | ID__8 | MC__UP | TACLR; // SMCLK / 8, up mode, clear timer. TA0EX0 = TAIDEX_7; // (SMCLK / 8) / 8 ~ 15.625 kHz. Default SMCLK: 1MHz TA0CCR0 = 6; // ~0.4ms TA0CCTL0 |= CCIE; // Capture/compare interrupt enable. __bis_SR_register(LPM3_bits | GIE); // Enter LPM3. Delay for Ref to settle. TA0CCR0 = 0x3D09; // Change timer delay to ~1 sec. **//Programming techniques** **// Clock configuration by default**	**// Configure Timer for LPM** TA0CTL = TASSEL__ACLK | MC__UP | TACLR; // ACLK, up mode, clear timer. TA0CCR0 = 131; // ~0.4ms TA0CCTL0 |= CCIE; // Capture/compare interrupt enable. __bis_SR_register(LPM3_bits | GIE); // Enter LPM3. Delay for Ref to settle. TA0CCR0 = 0x8000; // Change timer delay to ~1 sec. **//Programming techniques** **//Configure clocks for low power consumption**
Efficient.c	MostEfficient.c

Tabelle 8: Konfiguration und Nutzung des Timers via Settings für LPM

4.5.1 Codeanalyse im ULP Advice window

Die Division in der Funktion *rawToAsciiString(int16_t input)* wurde mit intSHIFT und intADD ersetzt. Zwei Divisionen sind notwendig, um die Zeile der ADC Werte in Celsius und Fahrenheit Grad zu konvertieren. Es gibt nach den ULP Empfehlungen keinen anderen Weg diese Divisionen zu vermeiden, außer die vorberechnete Konvertierungstabelle in den RAM unterzubringen und zur Laufzeit auszulesen.

4.5.2 Analyse der MPU/Interface/Clock window

LPM3 ist jetzt korrekt implementiert (Bild 11) und SMCLK und MODOSC sind deaktiviert.

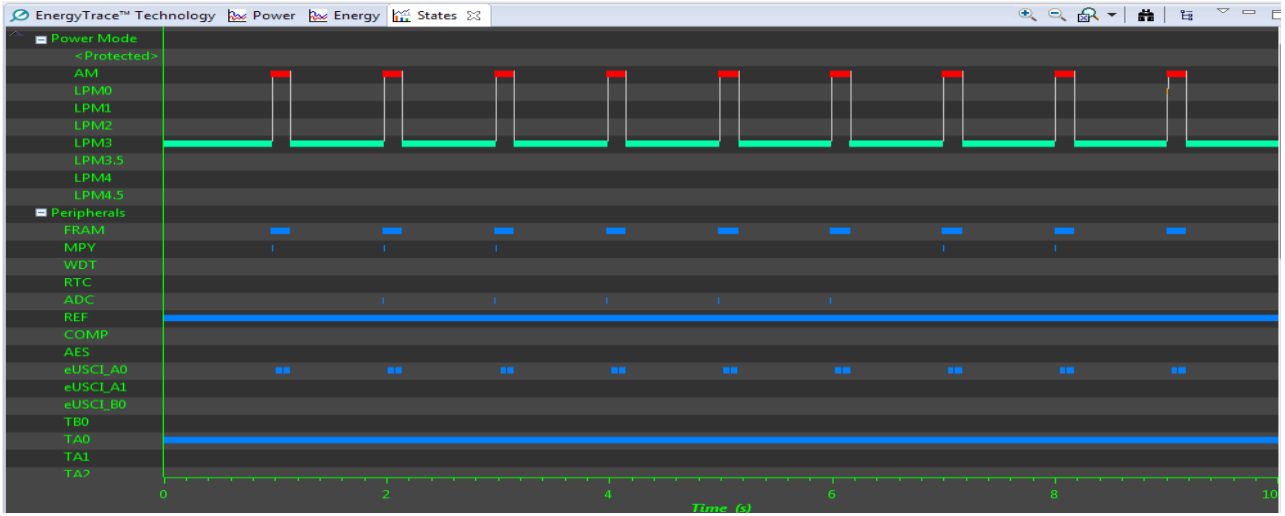
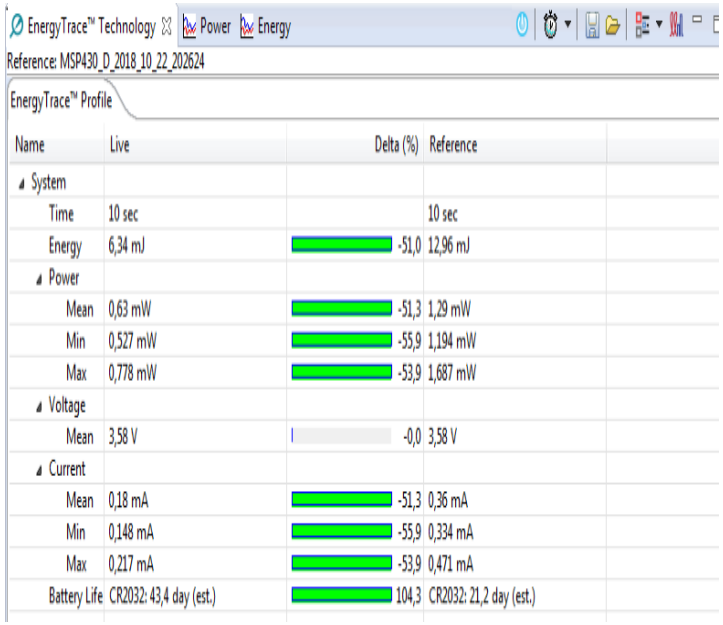


Bild 11: `MostEfficient.c` MCU/Interface Zustände

4.6 Vergleich aller Energieprofile

Der Energieverbrauch verringert sich deutlich mit dem `MostEfficient.c` Code (Bild 12 bis 14). Die besten (geringsten!) Werte der Leistung mit `Efficient.c` Code sind nur **60%** von den schlechtesten (größten!) Werte der Leistung mit `MostEfficient.c` Code zu verzeichnen.



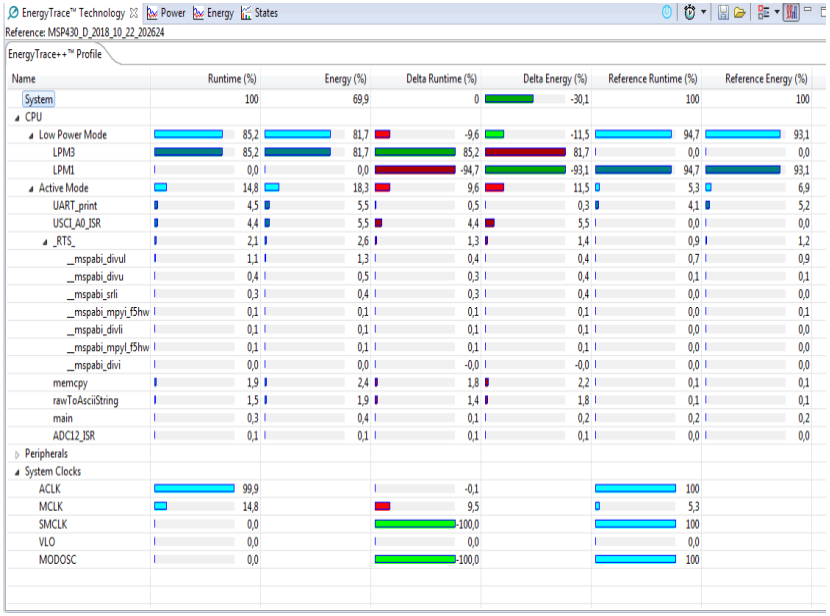


Bild 12: `MostEfficient.c` Vergleich mit `Efficient.c`

Bild 13: `MostEfficient.c` Leistung (Blue) Vergleich mit `Efficient.c` Leistung (Yellow)

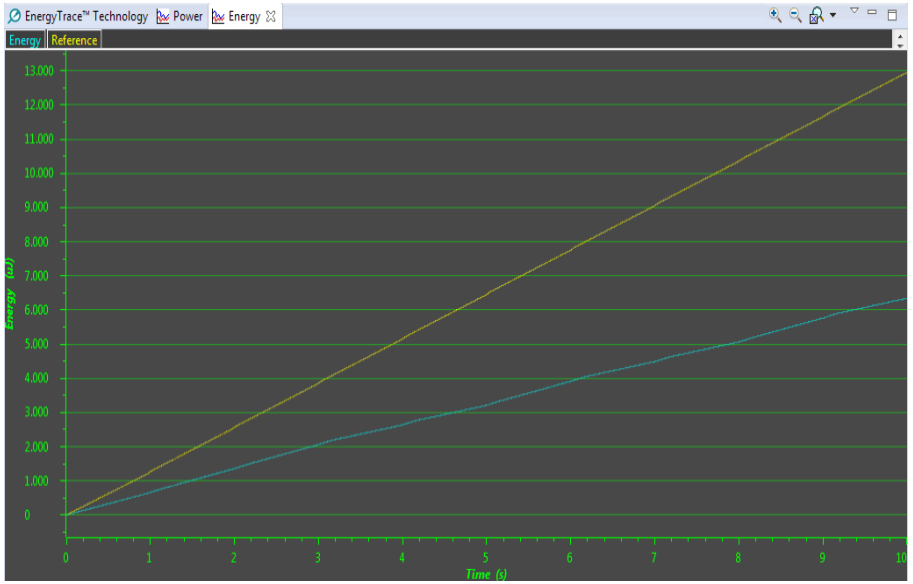
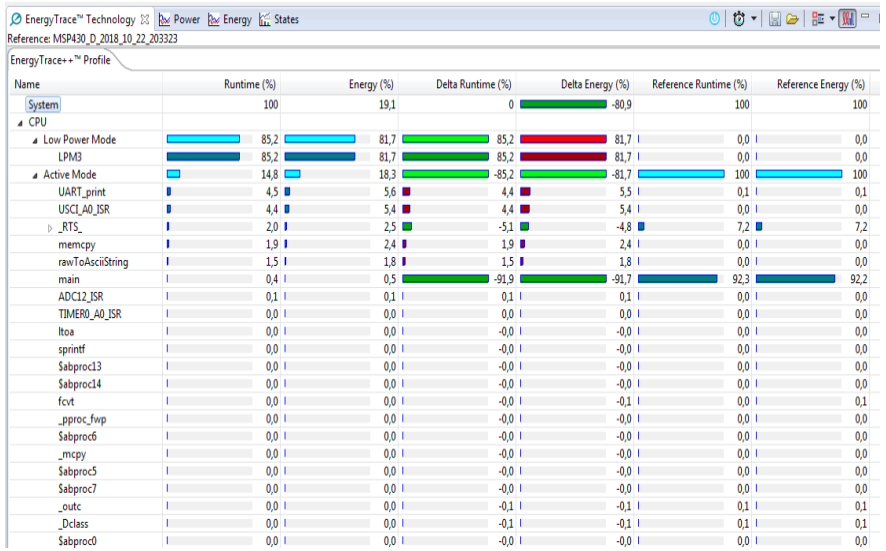


Bild 14: `MostEfficient.c` Energie (Blue) Vergleich mit `Efficient.c` Energie (Yellow)

Der Energieverbrauch **(47,55 mJ)** des Mikrocontrollers mit dem `Inefficient.c` Code ist mehr als **800%** höher als der Energieverbrauch **(5,84 mJ)** mit dem `MostEfficient.c` Code. **Das sind beachtliche Ergebnisse!**



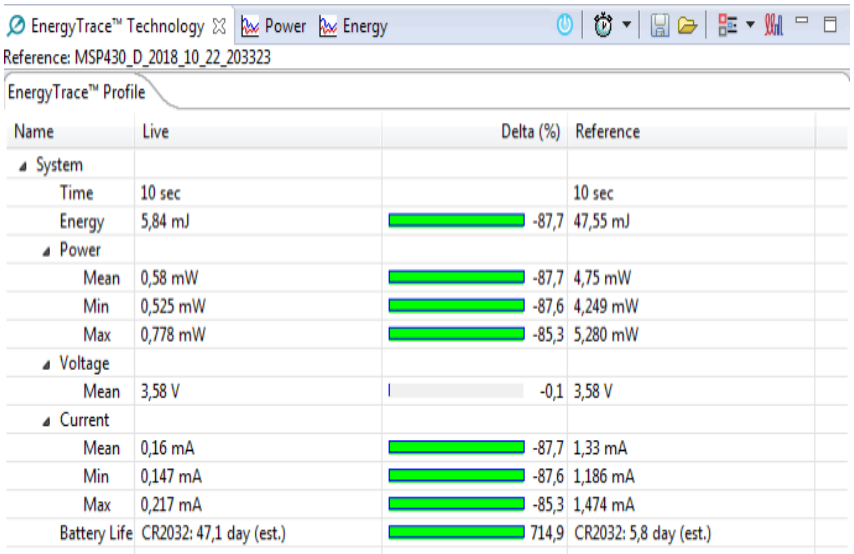


Bild 15: `MostEfficient.c` Vergleich mit `Inefficient.c`

Bild 16: `MostEfficient.c` Leistung Vergleich mit `Inefficient.c` Leistung

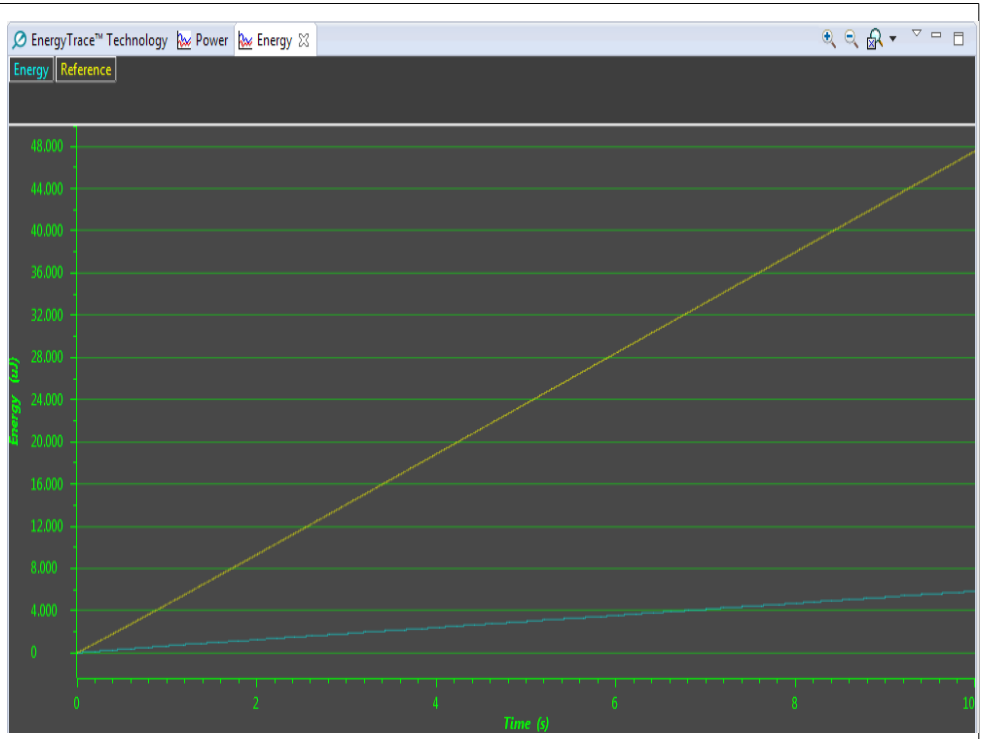


Bild 17: `MostEfficient.c` Energie Vergleich mit `Inefficient.c` Energie

4.7 Messung der absoluten Leistungsaufnahme

Im **EnergyTrace++ Mode** werden interne Zustände von MCU/Interface gemessen. Der Mikrocontroller wird ständig via 4-wire JTAG oder Spy-bi-Wire mit der Debugger-Schaltung verbunden. Diese Prozesse verbrauchen Energie und es ist nicht möglich die Energie für die Anwendung von dieser für den Debugger zu trennen. **Absolute Power Massung** ist die einzige Einstellung, die den Energieverbrauch ohne die Debugger-Schaltung des Mikrocontrollers erfasst. (Bild 18 und 19).

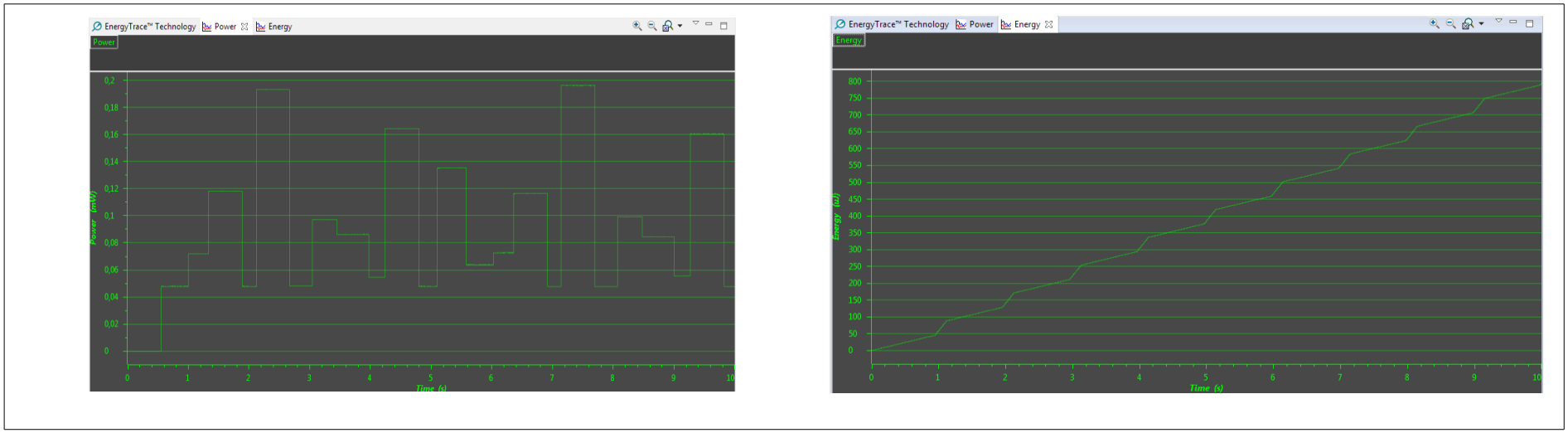
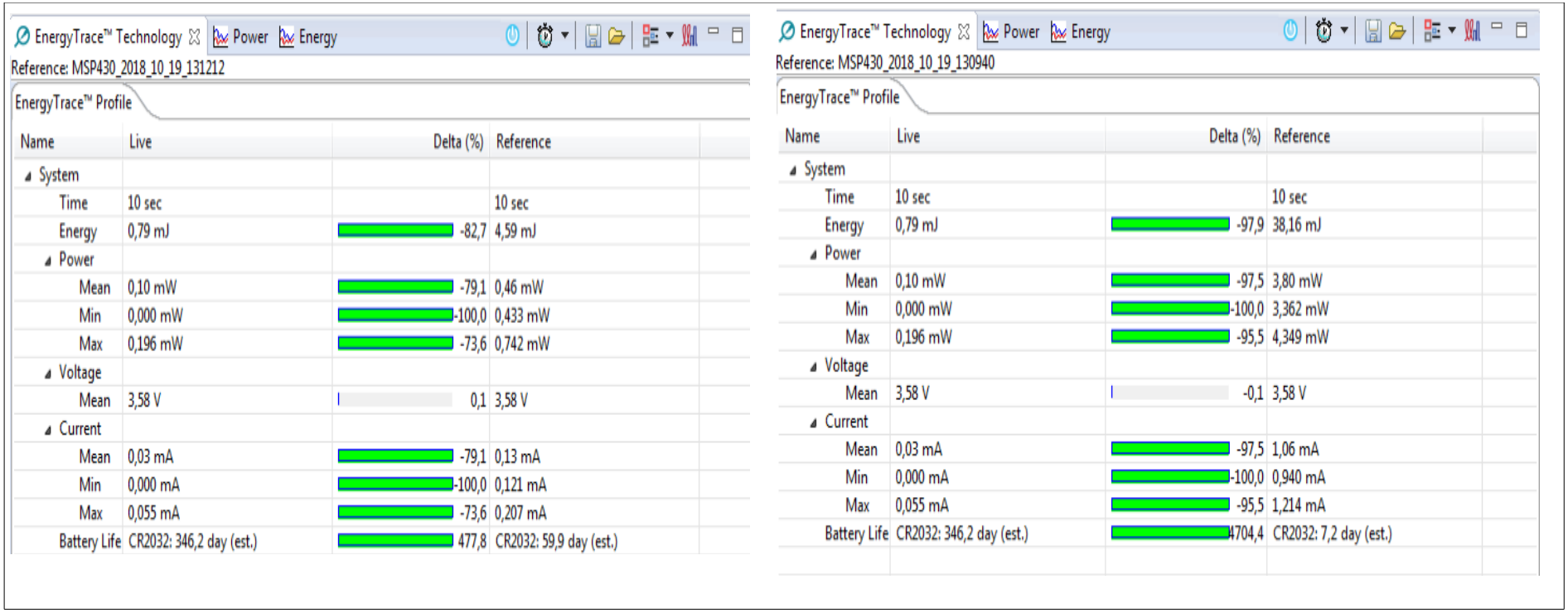


Bild 18: `MostEfficient.c` absolute Leistung/Energie



EnergyTrace™ Technology | Power | Energy

Reference: MSP430_2018_10_19_131212

EnergyTrace™ Profile

Name	Live	Delta (%)	Reference
System			
Time	10 sec		10 sec
Energy	0,79 mJ	-82,7	4,59 mJ
Power			
Mean	0,10 mW	-79,1	0,46 mW
Min	0,000 mW	-100,0	0,433 mW
Max	0,196 mW	-73,6	0,742 mW
Voltage			
Mean	3,58 V	0,1	3,58 V
Current			
Mean	0,03 mA	-79,1	0,13 mA
Min	0,000 mA	-100,0	0,121 mA
Max	0,055 mA	-73,6	0,207 mA
Battery Life	CR2032: 346,2 day (est.)	477,8	CR2032: 59,9 day (est.)

EnergyTrace™ Technology | Power | Energy

Reference: MSP430_2018_10_19_130940

EnergyTrace™ Profile

Name	Live	Delta (%)	Reference
System			
Time	10 sec		10 sec
Energy	0,79 mJ	-97,9	38,16 mJ
Power			
Mean	0,10 mW	-97,5	3,80 mW
Min	0,000 mW	-100,0	3,362 mW
Max	0,196 mW	-95,5	4,349 mW
Voltage			
Mean	3,58 V	-0,1	3,58 V
Current			
Mean	0,03 mA	-97,5	1,06 mA
Min	0,000 mA	-100,0	0,940 mA
Max	0,055 mA	-95,5	1,214 mA
Battery Life	CR2032: 346,2 day (est.)	4704,4	CR2032: 7,2 day (est.)

Bild 19: Vergleich `MostEfficient.c` mit `Efficient.c` und `Inefficient.c` Leistung/Strom/Energie

Tabelle 9 zeigt den Energieverbrauch für jede Codeversion in **meiner Verifikation** und diesen in der **TI Fallstudie.**

Code File	Energy (mJ)	Energy (%)	TI Case study (mJ)
`Inefficient.c`	**38,16**	**100**	10,03
`Efficient.c`	**4,59**	**12,02**	4,48
`MostEfficient.c`	**0,79**	**2,07**	0,81

Tabelle 9: Vergleich der **Absolute-Energie** Messung

Die Ergebnisse unterscheiden sich gering bis auf diese für `Inefficient.c`. In der Endversion der Energieopimierung des Codes, `MostEfficient.c`, wird die von mir gemessene Energieaufnahme bis auf **2,07%** (TI Case study 8.1%) vom Anfangsniveau reduziert! Das ist ein beachtliches Ergebnis der Anwendung von angemessenen **Programmiertechniken, die MPU/Interface Low-Power Modi, mehrere Taktmodi, Interrupts statt Schleifen und ADD-Shift Operationen statt MUL/DIV einsetzen**.

Es lohnt sich diese Programmiertechniken zu erlernen und anzuwenden!

5. *Zusammenfassung der Möglichkeiten der* MSP430™ Advanced Power Optimierung

Die Optimierung der Energieaufnahme durch Anwendung geeigneter Techniken in dafür konstruierten Mikrocontroller ist ein kritischer Schritt zur Anwendung von Embedded Systems in so wichtigen Bereichen wie IoT, Wireless Sensor networks, mobile Anwendungen insbesondere mit Energy Harvesting u.v.m. Es sind viele, für MSR-Anwendern ungewohnte Schritte, zu gehen, um die Energieaufnahme ihrer Anwendungen drastisch zu reduzieren und sie so in vielen Fällen erst möglich zu machen. Dieser Vortrag hat versucht eine Übersicht darüber zu geben, wie mit Hilfe von geeigneten Werkzeugen dieses Ziel zu erreichen ist. Der Einsatz von **ULP Advisor™** und **EnergyTrace technology™** im Entwicklungsprozess hilft Energieineffizienzen im Code zu finden, mit verschiedenen Techniken zu überwinden, die Ergebnisse zu messen und in überschaubarer Form zu präsentieren. So kann die Energieeffizienz der Anwendung dramatisch reduziert werden.

6. Schlussfolgerungen und Ausblick auf x86 Low-Power Programmiertechniken

Die Programmiertechniken für Embedded Systems zur Reduktion der Energieaufnahme von Anwenderprogrammen können mit einigen Ausnahmen und Einschränkungen in x86-basierten Servern und **H**igh **P**erformace **C**omputern angewandt werden, um die Herausforderungen zu begegnen, die im Kapitel „Motivation“ dieses Vortrages dargestellt wurden. Hier wollen wir einen Ausblick auf Low-Power Programmiertechniken für x86-basierten Computern wagen, die in einem weiteren Vortrag detailliert und mit Messergebnissen vorgestellt werden sollen.

A. Anzahl der CPU Takte minimieren

Die verbrauchte elektrische Leistung beim Ablauf einer Codesequenz in einem **Low-Power** digitalen System wird durch die Formel beschrieben:

P = ΔT * ~F * U * ~I

wo **~F** die durchschnittliche Taktfrequenz, **ΔT** die Rechenzeit, **U** die Betriebsspannung und **~I** die durchschnittliche Stromstärke sind. Die Reduktion von **~F** über die Rechenzeit wird unter Punkt **B** diskutiert. Die Reduktion von **ΔT** bedeutet die Anzahl der CPU/Interface Takte mit geeigneten Programmiertechniken auf **HLL-Ebene** oder mit **Parallel Assembly Instructions** zu reduzieren. **U** ist technologiebedingt.

ΔT und ~F sind das Ergebnis der Low-Power Programmieranstrengungen,

~F und ~I beschreiben die reduzierte Energieaufnahme.

B. Nutzung von Processor power management technologies [02], [03]

* Processor idle sleep states (x86)

Moderne x86 Architekturen unterstützen mehrere CPU/Interface „**Lehrlauf** (idle)“ und „**Schlaf** (sleep)“ Zustände, s.g. C-Zustände. Je länger CPU/Interface in einem je „tieferen“ „Schlafzustand“ verweilen desto geringer die Leistungsaufnahme und Energieverbrauch. C-Zustandsübergänge haben einen direkten Einfluss auf Energieverbrauch und Reaktionszeit.

* Power performance states (x86)

P-Zustände bieten die programmierbare Möglichkeit die Taktfrequenz und die Betriebsspannung an Baugruppen des Rechnerarchitektur, je nach algorithmischem Bedarf, zu skalieren, um den Energieverbrauch zu senken. CPU/Interface in aktiven P-Zustünde genannt Thermal Design Power haben den höchsten Energieverbrauch.

Embedded Systems	x86 servers and HPC
Ersetzen von Standardfunktionen in math.h mit Low-Power Operationen und Berechnungen.	Ersetzen von Standardfunktionen in math.h mit Low-Power Operationen und Berechnungen.
Ersetzen von FP-Standardfunktionen mit Integer-Calculus aus **Fixed Point Libraries.**	Replace standard FP functions with **Streaming SIMD Extensions** (SSE/SSE2) using Parallel Data Types in Assembly Intrinsic.
Ersetzen der Int DIV Operation mit deren reziproker Wert außerhalb von Schleifen und Einsetzen mit IntMUL in der Schleife.	Ersetzen der Int DIV Operation mit deren reziproker Wert außerhalb von Schleifen und Einsetzen mit IntMUL in der Schleife.
Immer interrupts für Datentransfer mit Interface nutzen.	Not possible, because of Thread-based Multi Tasking OS.
Timer nutzen, um Zeitkonstanten zu definieren.	Timer nutzen, um Zeitkonstanten zu definieren.
Low-Power Modi und Taktarten für CPU/Interface nutzen.	Use x86 Processor idle and Power performance states [2], [3]

Tabelle 10: Ähnlichkeiten und Unterschiede zwischen Low-Power Programmiertechniken – ES vs. x86:

blue (Ähnlichkeiten), red (Unterschiede)

7. Literatur

[01] Texas Instruments, MSP430™ Advanced Power Optimizations: ULPAdvisor™ Software and EnergyTrace™ Technology, June 2014, http://www.ti.com/lit/an/slaa603/slaa603.pdf

[02] Hewlett-Packard Co.,Intel Co., Microsoft Co., Phoenix Technologies Ltd., and Toshiba Co. Advanced Configuration and Power Interface Specification, November 13, 2013.
http://www.acpi.info/spec.htm

[03] Microsoft Co., CPU Power Management, Hewlett-Packard Co., Intel Co., Microsoft Co., Phoenix Technologies Ltd., Toshiba Co., 31.05.2018,
https://docs.microsoft.com/de-de/previous-versions/windows/desktop/xperf/cpu-power-management

[04] vom Orde,H., Durner, A.: Grunddaten Jugend und Medien 2019, Aktuelle Ergebnisse zur Mediennutzung von Jugendlichen in Deutschland, Internationales Zentralinstitut für das Jugend-und Bildungsfernsehen, 2019, https://www.br-online.de/jugend/izi/deutsch/Grundddaten_Jugend_Medien.pdf

[05] The Shift Project: TOWARDS DIGITAL SOBRIETY, 2019, https://theshiftproject.org/wp-content/uploads/2019/03/Lean-ICT-Report_The-Shift-Project_2019.pdf

[06] Heizwert/Brennwert, Wikipedia: https://de.wikipedia.org/wiki/Heizwert

[07] Kraftstoffverbrauch, Wikipedia: https://de.wikipedia.org/wiki/Kraftstoffverbrauch#Berechnung_der_CO2-Emission_auf_Basis_des_Kraftstoffverbrauchs

[08] Wolframalpha: https://www.wolframalpha.com/input/?i=log-linear+plot+x%5E2+log+x,+x%3D1+to+10

[09] Wolframalpha: https://www.wolframalpha.com/input/?i=log-linear+plot+x%5E1.1+log+x,+x%3D1+to+10

[10] Hans-Werner Sinn. European Economic Review 99, Oktober 2017, S. 130-150 und CESifo Working Paper Nr. 5950, Juni 2016, http://www.hanswernersinn.de/de/2017_EER_Buffering_Volatility, https://www.50hertz.com/de/Transparenz/Kennzahlen/Windenergie.

[11] Bevölkerung - Zahl der Einwohner in Deutschland nach Altersgruppen am 31. Dezember 2017 (in Millionen), https://de.statista.com/statistik/daten/studie/1365/umfrage/bevoelkerung-deutschlands-nach-altersgruppen/

[12] Nettostromverbrauch in Deutschland in den Jahren 1991 bis 2018 in TWh, https://de.statista.com/statistik/daten/studie/164149/umfrage/netto-stromverbrauch-in-deutschland-seit-1999/

[13] Entwicklung des Stromverbrauchs nach Sektoren, https://www.umweltbundesamt.de/daten/energie/stromverbrauch

[14] Liste der größten Kohlenstoffdioxidemittenten, https://de.wikipedia.org/wiki/Liste_der_gr%C3%B6%C3%9Ften_Kohlenstoffdioxidemittenten

„Wenn Sie die Art und Weise ändern, wie Sie die Dinge betrachten, ändern sich die Dinge, die Sie betrachten.“

Max Planck
https://beruhmte-zitate.de/autoren/max-planck/

Liebe Kollegen, danke für Ihre Aufmerksamkeit!