



FACULTY OF ENGINEERING TECHNOLOGY
— GENT —

EMBEDDED DESIGN 1 – LABO LABO 1 & 2

Simplicity Studio & ARM Energy Modes

Brecht Van Eeckhoudt – Chris Thoen

MELICTE

28 oktober 2018

Inhoudsopgave

1	Doelstellingen	2
2	Achtergrond	3
2.1	Energie-modes	3
3	Opdrachten	4
3.1	Blink	4
3.1.1	Doelstellingen en opdracht	4
3.1.2	Uitwerking – Code	4
3.1.3	Uitwerking – Stroomverbruik	5
3.2	Energy Modes	7
3.2.1	Doelstellingen en opdracht	7
3.2.2	Uitwerking	7
3.2.3	Om verder over na te denken	9
3.3	Energy-efficient blink	9
3.3.1	Doelstellingen en opdracht	9
3.3.2	Uitwerking	9
4	Besluit	11
4.1	Wat hebben we verwezenlijkt?	11
4.2	Wat kan er beter en hoe kunnen we dit tot stand brengen?	11
Referenties		11
A	Code ‘blink.c’	12

1 Doelstellingen

Voor het labo ‘Embedded System Design 1’ wordt gebruik gemaakt van de *Simplicity Studio 4* IDE. Deze Eclipse-gebaseerde omgeving bevat een groot aantal tools die helpen bij het ontwikkelen van embedded toepassingen die gebruik maken van *Silicon Labs* microcontrollers. Zo is er naast een *editor*, *debugger/programmer*, bijvoorbeeld ook een *energy profiler* aanwezig die toelaat om het stroomverbruik in real time te analyseren.

In deze labo’s maken we gebruik van twee Silicon Labs Development Kits. De STK3800 is gebaseerd op een *EFM32 ‘Wonder Gecko’ Cortex-M4* en heeft een hardware *FPU*. De STK3400 heeft een *EFM32 ‘Happy Gecko’ Cortex-M0* aan boord.

Voor deze opgave is het van geen belang welke van de twee kits er gebruikt wordt. In dit labo is er gebruik gemaakt van het ‘Happy Gecko’ **SLSTK3400A** bordje gebruikt met een *Cortex-M0*.

2 Achtergrond

2.1 Energie-modes

De EFM32 beschikt over vijf energie-modes waarmee men verschillende functies van het bordje kan uitschakelen en dus energie mee kan besparen. De beschikbare functionaliteit neemt logischerwijs af naarmate men in een lagere energie-mode komt. De ‘hoogste’ energie-mode is de **EM0**, wat de default mode is waarin het bordje werkt. In tabel 1 zijn de belangrijkste eigenschappen van elke energie-mode weergegeven.

Tabel 1: EM-modes EFM 32 [1].

EM0 – Energie-mode 0 (Run mode)	De CPU ‘werkt’ en verbruikt zo’n $230 \mu A/MHz$. Alle randapparatuur kan actief zijn.
EM1 – Energie-mode 1 (Sleep mode)	De CPU bevindt zich in ‘slaapstand’ en verbruikt ongeveer $67 \mu A/MHz$. Alle randapparatuur zijn nog steeds beschikbaar.
EM2 – Energie-mode 2 (Deep Sleep Mode)	De hoogfrequent-oscillator is uitgeschakeld maar de $32.768 kHz$ oscillator blijft nog actief. De lage-energie randapparatuur (LCD, RTC, LETIMER, PCNT, WDOG, LEUART, I ² C, ACMP, LESENSE, OPAMP, USB) zijn nog steeds beschikbaar. In deze mode kan men het verbruik tot $0.95 \mu A$ terugdringen waarbij de RTC ingeschakeld blijft.
EM3 – Energie-mode 3 (Stop Mode)	Hier is ook de lage-frequentie oscillator uitgeschakeld, maar de CPU en het RAM-geheugen blijft behouden. Power-on Reset, Pin reset, EM4 wake-up en Brown-out detectie is beschikbaar. Hierbij wordt er slechts $0.65 \mu A$ verbruikt. De low-power ACMP, asynchrone externe interrupt, PCNT en I ² C kunnen het apparaat wakker maken in slechts een aantal microseconden.
EM4 – Energie-mode 4 (Shutoff Mode)	De microcontroller verbruikt hier $20 nA$ en alle functionaliteit is uitgeschakeld, behalve: pin reset, GPIO pin wake-up, GPIO pin behoud, Backup RTC en Power-On Reset. Alle pinnen worden in hun ‘reset’ status geplaatst.

3 Opdrachten

3.1 Blink

3.1.1 Doelstellingen en opdracht

In deze opdracht wordt er kennis gemaakt met *Simplicity Studio* en enkele van de geïntegreerde tools. Hierbij worden de resultaten geverifieerd op basis van datasheets en schema's.

Gebruikmakend van de ‘*user LEDs*’ van het ontwikkelbordje moet er een een knipperlicht gemaakt worden (‘*blinking leds*’) vertrekende van een voorbeeldproject. De code die hierin wordt aangereikt moet aangepast worden zodat er slechts één LED gebruikt wordt en dat deze om de seconde gedurende 100 ms brandt. Ook moet er aangetoond worden dat het stroomverbruik binnen de te verwachten grenzen valt.

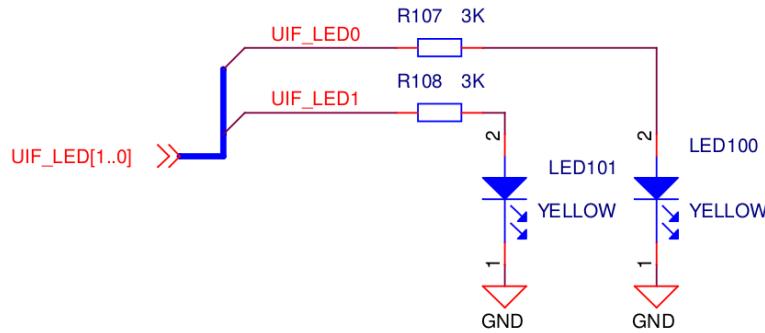
3.1.2 Uitwerking – Code

Het aanpassen van de aangereikte voorbeeldproject was vrij eenvoudig. Om de gewenste werking te bekomen hebben we enkel de code in de oneindige ‘`while (1)`’ loop moeten aanpassen. Het resultaat is hieronder weergegeven.

```
// Infinite loop
while (1) {
    BSP_LedClear(0);      // Disable LED0
    Delay(1000);          // 1000 ms (1 sec) delay
    BSP_LedSet(0);        // Enable LED0
    Delay(100);           // 100 ms delay
}
```

3.1.3 Uitwerking – Stroomverbruik

Om te controleren of het gemeten stroomverbruik al dan niet realistisch was zijn we begonnen met het opzoeken van het schema van het ontwikkelbordje. Hieruit hebben we onderstaand fragment (figuur 1) gehaald waarop we de waarden van de voorschakelweerstanden van de LED's kunnen aflezen.



Figuur 1: Fragment van het schema van het ontwikkelbordje [2].

Vervolgens hebben we met een multimeter spanningen gemeten die aanwezig waren als een ledje aanstond:

- **Voedingsspanning:** 3,3 V
- **Spanning over LED:** 1,824 V
- **Spanning over voorschakelweerstand:** 1,482 V

Met deze gegevens kunnen we de **praktisch vloeiente stroom** berekenen:

$$I_{resistor} = \frac{U_{resistor}}{R_{resistor}} = \frac{1,482 \text{ V}}{3 \text{ k}\Omega} = 494 \mu\text{A}$$

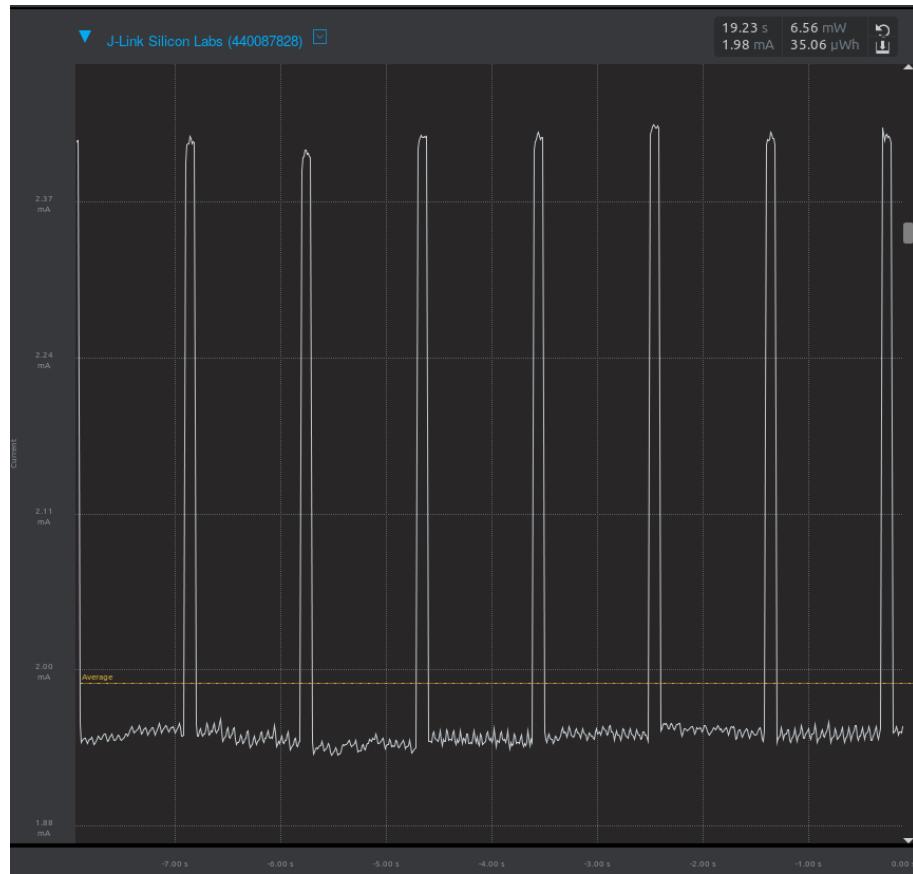
Om te kijken bij welke **klokfrequentie** de microcontoller werkt hebben we onderstaande lijn code even toegevoegd. Dit bleek 14 MHz te zijn.

```
uint32_t clockfreq = CMU_ClockFreqGet(cmuClock_CORE);
```

Hiermee kunnen we dan het **theoretische stroomverbruik van de microcontroller** berekenen (mode E0):

$$I_{MCU} = clockFrequency \times 230 \mu\text{A}/\text{MHz} = 14 \text{ MHz} \times 230 \mu\text{A}/\text{MHz} = 3,220 \text{ mA}$$

Vervolgens hebben we via de *energy-profiler* het praktisch verbruik nagekeken. Het resultaat hiervan zien we op figuur 2.



Figuur 2: Stroomprofiel bij de blink opdracht.

Nu kunnen we de theoretische en praktische waarden vergelijken. Dit doen we op tabel 2 en tabel 3¹. We merken op dat het **theoretisch stroomverbruik hoger ligt dan het praktisch verbruik**. Dit kan te wijten zijn aan het feit dat de microcontroller niet zo een *energy-intensive tasks* moet verrichten bij het aan-en uitschakelen van een LED.

Tabel 2: Stroomverbruik (LED uit).

Theoretisch	Praktisch
3,22 mA	2,01 mA

Tabel 3: Stroomverbruik (LED aan).

Theoretisch	Praktisch
3,714 mA	2,53 mA

Als laatste kunnen we vermelden dat we hier een **gemiddeld stroomverbruik** hebben van **ongeveer 1,98 mA**.

¹We hebben de ‘theoretische’ waarde verkregen door het berekende stroomverbruik van de MCU samen te tellen bij de stroom door de weerstand en LED die we bekomen hebben via een meet- en bereken combinatie (weergegeven op de vorige bladzijde).

3.2 Energy Modes

3.2.1 Doelstellingen en opdracht

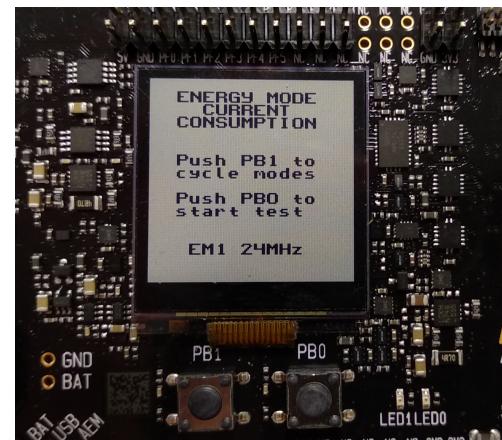
In volgende opdracht leren we de impact kennen van de verschillende *Energy Modes (EM)* van ARM Cortex microcontrollers. Dit kunnen we doen aan de hand van een voorbeeld-project in combinatie met de *energy-profiler*.

3.2.2 Uitwerking

Om de impact van de verschillende energie-modes te bestuderen hebben we gebruik gemaakt van het voorbeeldproject ‘SLSTK3400A_emode’. Indien men dit project upload naar het bord, ziet men op het LCD een menu verschijnen waarbij men verschillende energie-modes bij verschillende frequenties kan selecteren. Dit is weergegeven op figuur 3 en figuur 4.



Figuur 3: Tekst op het LCD.

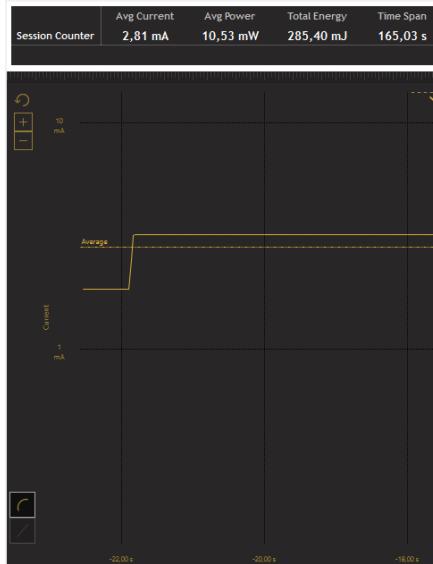


Figuur 4: Tekst op het LCD.

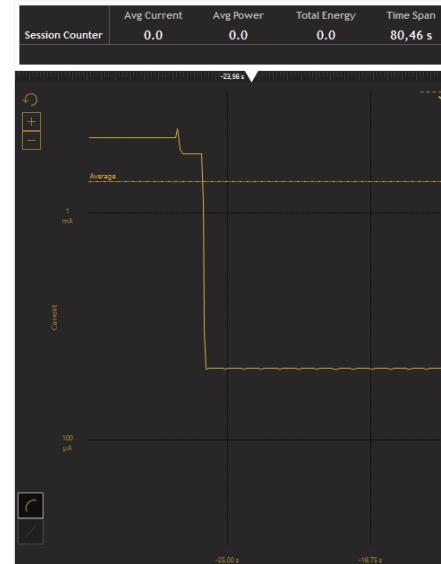
Indien we via drukknop PBO een geselecteerde test starten (via drukknop PB1 kunnen we van test veranderen) begint de microcontroller *primitieve getallen* te berekenen om hem op deze manier te beladen.

Op figuur 5, figuur 6, figuur 7 en figuur 8 zien we de gemeten waarden via de *energy-profiler* bij verschillende EM-modes en klokfrequenties². We zien duidelijk dat het verbruik telkens lager ligt in mode EM1, ook wanneer de klokfrequentie lager is.

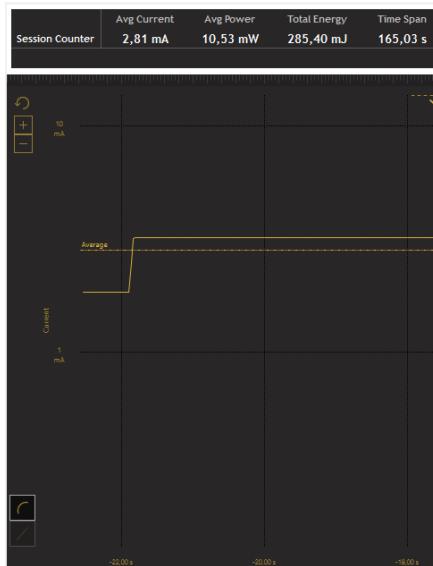
Verder zien we op figuur 5 en figuur 7 dat het verbruik na het selecteren van de specifieke energie-mode nog oploopt. Dit komt omdat de microcontroller zwaardere berekeningen moet doen dan indien hij gewoon wat op het display moet weergeven waardoor hij dus logischerwijs meer verbruikt.



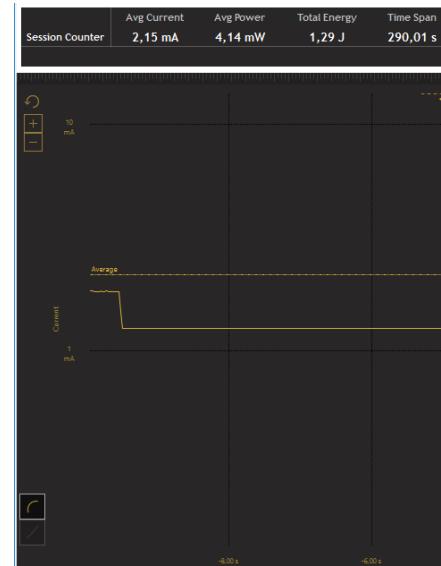
Figuur 5: EM0 @ 24 MHz.



Figuur 6: EM0 @ 1,2 MHz.



Figuur 7: EM0 @ 24 MHz.



Figuur 8: EM1 @ 24 MHz.

²We mogen niet vergeten dat de verticale stroomschaal **logaritmisch** is wat soms een vertekend beeld kan geven.

3.2.3 Om verder over na te denken

Waarom is het niet altijd beter om te kiezen voor een lage klokfrequentie?

Indien er voor een lagere klokfrequentie wordt gekozen, betekent dit ook dat het langer zal duren om bepaalde instructies af te werken. Tijdens het verwerken van instructies zal de processor volledig actief blijven. Hoe sneller instructies afgewerkt worden, hoe sneller de processor zich daarna mogelijks in een slaaptoestand kan plaatsen.

Om het laagst mogelijke verbruik te bekomen zal men dus een afweging moeten maken waarbij de processor net niet te lang actief zal blijven en waarbij de frequentie zo laag mogelijk blijft.

3.3 Energy-efficient blink

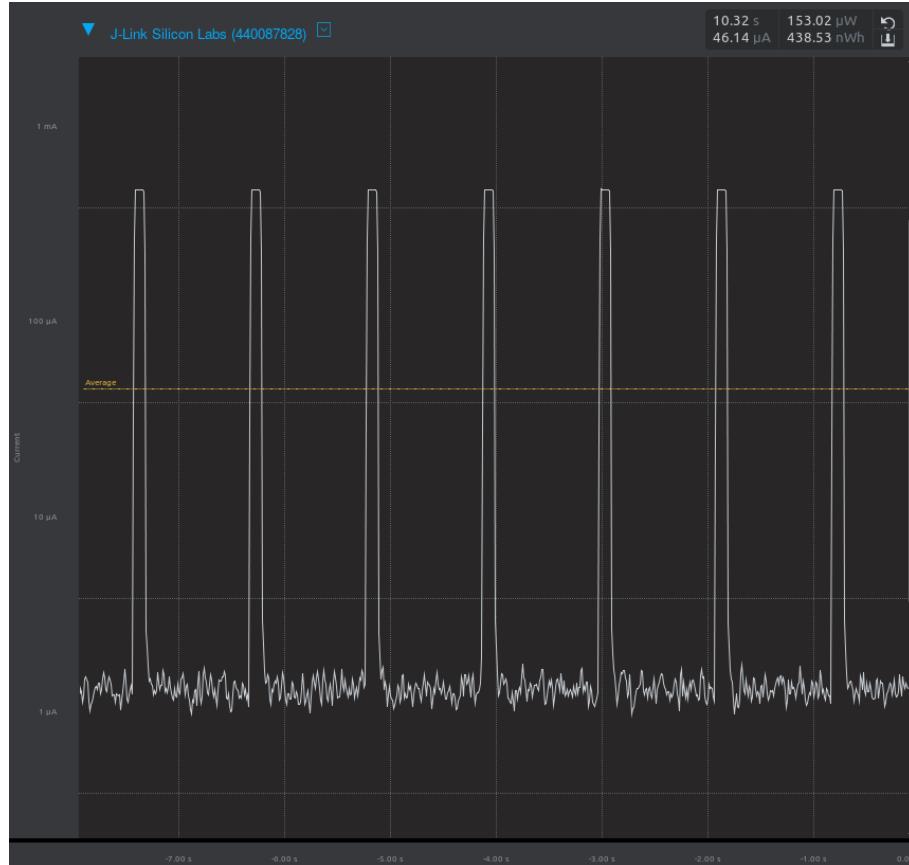
3.3.1 Doelstellingen en opdracht

In deze opdracht wordt de blink-opdracht aangepast zodat deze energie-efficiënt is. Hierbij wordt er gebruik gemaakt van de EM-modes die aan bod komen in paragraaf 3.2.

3.3.2 Uitwerking

Voor de toepassing *blink* is het enkel nodig om de *status* van de LED om de zoveel tijd te wijzigen. Zoals we in tabel 1 kunnen zien moet het bordje, om de status van de pin die de LED aanstuurt te wijzigen, zich in mode **EM0** bevinden. Terwijl de microcontroller in de tussentijd ‘*wacht*’ moeten we in feite enkel de tijd bijhouden. Dit kan aan de hand van de lage-frequentie oscillator. Dit wil zeggen dat we gedurende deze ‘*wachttijd*’ het bordje in mode **EM2** kunnen zetten, wat een aanzienlijke energiebesparing oplevert.

Deze aanpak resulteert in de code die terug te vinden is in bijlage A. Na het inladen van de code op het bordje kunnen we via de energy-profiler het stroomverbruik monitoren. Het resultaat hiervan is afgebeeld op figuur 9³. Hierop zien we duidelijk de hoge stroompieken wanneer de LED brand.



Figuur 9: Stroomprofiel bij de low-power blink opdracht.

We hebben uiteindelijk een gemiddeld stroomverbruik van ongeveer $46,74 \mu A$. Oorspronkelijk was dit ongeveer $1,98 mA$.

Belangrijke kanttekening: Tijdens het debuggen hebben we gezien dat na het uploaden van nieuwe code op de microcontroller deze manueel moet gereset worden via de drukknop om de ‘juiste’ stromen te verbruiken.

³Opnieuw maken we de kanttekening dat de verticale stroomschaal **logaritmisch** is wat soms een vertekend beeld kan geven.

4 Besluit

4.1 Wat hebben we verwezenlijkt?

In dit labo hebben we kennisgemaakt met de ontwikkelomgeving *Simplicity Studio* en het SLSTK3400A developmentbordje. Met het **oog op low-power applicaties** is er gekeken naar mogelijkheden om het energieverbruik van dit developmentbordje zo veel mogelijk te beperken. Hiervoor kan men gebruik maken van **verschillende energie-modes** waarin men de microcontroller kan plaatsen.

Het gebruikte bordje beschikt over vijf energie-modes, waarbij des te hoger de mode (op het vlak van het ‘nummer’ van de energie-mode), des te lager het energieverbruik zal zijn. Dit heeft echter als gevolg dat de **functionaliteit van het bord beperkt wordt** naarmate de energie-mode toeneemt. Men kan makkelijk overschakelen naar een andere energie-mode, echter moet men er wel rekening mee houden dat dit enige ‘*omschakeltijd*’ in beslag neemt.

In dit labo is er van deze energie-modes gebruik gemaakt om een **low-power blink toe-passing** voor het bordje te schrijven. Hiervoor werd er gebruik gemaakt van de **externe 32.768 kHz oscillator** die onder energie-mode 2 actief blijft. Om de toestand van de pin die verbonden is met de LED te wijzigen dient het bord zich echter in energie-mode 0 te bevinden, waardoor we moeten **afwisselend mode 2 en 0** moeten selecteren. Door van deze werkmethode gebruik te maken is het energieverbruik aanzienlijk gedaald ten opzichte van de oorspronkelijke blink-voorbeeldcode.

4.2 Wat kan er beter en hoe kunnen we dit tot stand brengen?

Uit voorgaande meetresultaten kunnen we zien dat het **energieverbruik voornamelijk bepaald wordt door de LED**. We zouden het energieverbruik verder kunnen beperken door **PWM-modulatie** op deze LED toe te passen. Dit heeft als gevolg dat de LED minder hard zal branden maar indien dit geen probleem is zal hij wel aanzienlijk minder verbruiken. Het verbruik zal afnemen naarmate de *duty-cycle* afneemt.

Referenties

- [1] Mouser, *EM-states*,
https://www.mouser.com/pdfdocs/d0233_efm32wg_reference_manual.pdf
- [2] Silicon Labs, *EFM32 Happy Gecko STK*,
<https://www.silabs.com/documents/public/schematic-files/efm32hg-stk3400-schematics.pdf>

A Code ‘blink.c’

```
1  ****//*****************************************************************************/**/
2  * @file
3  * @brief Simple LED Blink Demo for SLSTK3400A_EFM32HG
4  * @version 5.6.1
5  ****//*****************************************************************************/*
6  * # License
7  * <b>Copyright 2015 Silicon Labs, Inc. http://www.silabs.com</b>
8  ****//*****************************************************************************/*
9  *
10 * This file is licensed under the Silabs License Agreement. See the file
11 * "Silabs_License_Agreement.txt" for details. Before using this software for
12 * any purpose, you must agree to the terms of that agreement.
13 *
14 ****//*****************************************************************************/*
15
16 #include <stdint.h>
17 #include "em_device.h"
18 #include "em_chip.h"
19 #include "em_cmu.h"
20 #include "em_emu.h"
21 #include "em_rtc.h"
22 #include "bsp.h"
23
24 #define DELAY_1SEC 1.0
25 #define DELAY_100MSEC 0.1
26 #define LFXOFREQ      32768
27 #define COMPARE_1SEC   (DELAY_1SEC * LFXOFREQ)
28 #define COMPARE_100MSEC (DELAY_100MSEC * LFXOFREQ)
29
30 uint32_t ledOn = 0;
31
32 ****//*****************************************************************************/*
33 * @brief RTCC interrupt service routine
34 ****//*****************************************************************************/*
35 void RTC_IRQHandler(void)
36 {
37     // If the led is ON: Turn it off & set the RTC compare value to 1 s
38     if (ledOn == 0) {
39         ledOn = 1;
40
41         // Set RTC compare value for RTC 0
42         RTC_CompareSet(0, COMPARE_1SEC);
43     }
44
45     // If the led is OFF: Turn it on & set the RTC compare value to 100 ms
46     else {
47         ledOn = 0;
48
49         // Set RTC compare value for RTC 0
50         RTC_CompareSet(0, COMPARE_100MSEC);
51     }
52
53     // Reset counter
54     RTC_CounterReset();
55
56     // Clear the interrupt source
57     RTC_IntClear(RTC_IFC_COMPO);
```

```
58     // Toggle LED 0
59     BSP_LedToggle(0);
60 }
61
62 /**
63 * @brief RTCC initialization
64 */
65
66 void rtcSetup(void)
67 {
68     // Enable the oscillator for the RTC
69     CMU_OscillatorEnable(cmuOsc_LFX0, true, true);
70
71     // Turn on the clock for Low Energy clocks
72     CMU_ClockEnable(cmuClock_HFLE, true);
73     CMU_ClockSelectSet(cmuClock_LFA, cmuSelect_LFX0);
74
75     // Turn on the RTC clock
76     CMU_ClockEnable(cmuClock_RTC, true);
77
78     // Set RTC compare value for RTC 0
79     RTC_CompareSet(0, COMPARE_1SEC);
80
81     // Allow channel 0 to cause an interrupt
82     RTC_IntEnable(RTC_IEN_COMPO);
83     NVIC_ClearPendingIRQ(RTC_IRQn);
84     NVIC_EnableIRQ(RTC_IRQn);
85
86     // Configure the RTC settings
87     RTC_Init_TypeDef rtc = RTC_INIT_DEFAULT;
88
89     // Initialise RTC with pre-defined settings
90     RTC_Init(&rtc);
91 }
92
93 /**
94 * @brief Main function
95 */
96
97 int main(void)
98 {
99     // Chip errata
100    CHIP_Init();
101
102    // Initializations
103    BSP_LedsInit();
104    BSP_LedSet(0);
105
106    rtcSetup();
107
108    // Infinite loop
109    while(1)
110    {
111        EMU_EnterEM2(true); // "true": Save and restore oscillators, clocks and voltage
112        // scaling
113    }
114 }
```
