



FACULTY OF ENGINEERING TECHNOLOGY  
— GENT —

EMBEDDED SYSTEM DESIGN 1  
LABO

---

## Aansturen van een sensor

---

Chris Thoen – Brecht Van Eeckhoudt

MELICTE

20 december 2018

## Inhoudsopgave

<b>1 Doelstellingen</b>	<b>3</b>
<b>2 Meten van zeegolven</b>	<b>3</b>
<b>3 Keuze van de sensor</b>	<b>4</b>
3.1 MEMS accelerometer	5
<b>4 De SPI interface</b>	<b>6</b>
<b>5 Praktische uitwerking</b>	<b>7</b>
5.1 Code	7
5.1.1 Header en Source files	7
5.1.2 Opbouw	8
5.1.3 Code-flow	10
5.2 Problemen bij het ontwikkelen van de code	11
5.3 Demo	13
5.4 Stroommetingen	14
5.4.1 ODR 12,5 Hz – activity detection uit/aan	14
5.4.2 ODR 100 Hz (reset default) – activity detection uit/aan	17
5.4.3 UART aan	20
5.4.4 Samenvatting	21
5.4.5 Besluiten bij stroommetingen	21
<b>6 Besluit</b>	<b>23</b>
6.1 Wat hebben we verwezenlijkt?	23
6.2 Wat kan er beter? (denk verder dan de opgave)	23
6.3 Hoe kunnen we dit tot stand brengen?	23
<b>Referenties</b>	<b>24</b>

## 1 Doelstellingen

In dit labo wordt er een sensor geselecteerd die geschikt is om metingen op een boei op zee uit te voeren. In de eerste plaats wordt er een sensor gezocht die interessante meetresultaten in desbetreffende situatie kan opleveren. Daarna wordt deze sensor aan het SLSTK3400A Happy Gecko ontwikkelingsbord gekoppeld, afhankelijk van de interface waarmee de sensor werkt. Vervolgens wordt er getracht om de sensor en het bordje te laten werken op een zo laag mogelijk energieverbruik.

## 2 Meten van zeegolven

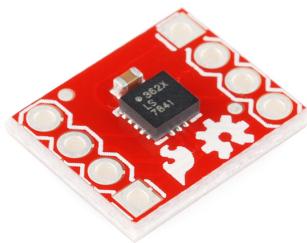
Het is mogelijk om met een accelerometer de frequentie van golven te bepalen. Hiervoor moet men een aantal samples nemen waarna men een FFT (Fast Fourier Transform) op deze data kan uitvoeren. Hiervoor kan er gebruik gemaakt worden van het FIFO register op de sensor. Op deze manier kan men de microcontroller langer in slaap houden voor er een *batch* van samples doorgestuurd wordt. Het nodige aantal samples kan men bepalen met het *niquist criterium*. De minimale samplefrequentie moet gelijk zijn aan het dubbele van de hoogst voorkomende frequentie in het spectrum. De meeste zeegolven hebben een frequentie tussen de 0,2 Hz tot 0,06 Hz [3]. Het minimum aantal samples is bijgevolg 0,4 samples per seconde.

Het is ook mogelijk om de golfhoogte te bepalen met een accelerometer. In het tijdschriftartikel waarin men dit aantoon heeft men echter ook nog gebruik gemaakt van een GPS en een magnetometer [4].

### 3 Keuze van de sensor

De gekozen sensor voor dit labo is de ADXL362. Dit is een ultralow power MEMS accelerometer die in 3 assen de acceleratie kan meten [1]. Er is voor een accelerometer gekozen omdat deze in staat is om heel wat nuttige informatie op zee te meten, zoals de golfslag en golf-frequentie. Specifiek is er voor de ADXL362 gekozen omdat deze ultralowpower is, wat te danken is aan het gebruik van de MEMS technologie waarmee deze sensor is gebouwd.

Voor dit labo is er gebruik gemaakt van het sparkfun breakoutbord van deze sensor, zoals te zien op figuur 1, zodat de sensor gemakkelijk aan het ontwikkelbord gekoppelt kan worden.



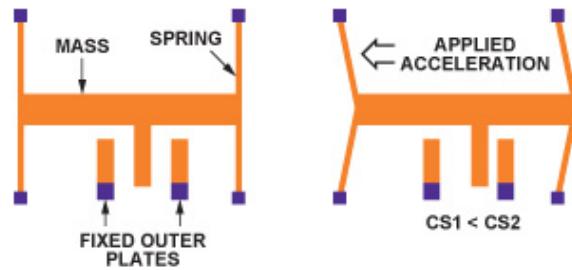
**Figuur 1:** Breakoutbord gebruikt voor dit labo [2].

De sensor kan gevoed worden met een spanning tussen de 1,6V en 3,5V en kan dus rechtstreeks verbonden worden met de SLSTK3400A. De meting van de sensorwaarde wordt digitaal doorgestuurd via SPI, waarmee in feite zijn registers uitgelezen worden waarin deze waarde zit. Tevens beschikt de accelerometer ook nog over twee configurerbare interrupt-pinnen.

### 3.1 MEMS accelerometer

*Micro-elekromechanische systemen*, of kortweg MEMS, is een technologie waarbij er mechanische structuren op silicium worden geëtst. Hierdoor kunnen deze structuren fracties van een micron bewegen.

De basis van een MEMS accelerometer is een balk met een set van ‘vingers’. Één set staat vast op een grondvlak op het substraat, de andere is verbonden met een veersysteem. Deze structuur is te zien op figuur 2. Doordat de capaciteit verschilt ten opzichte van de verschillende platen kunnen deze bewegingen geregistreerd worden waaruit uiteindelijk de acceleratie bepaald wordt.

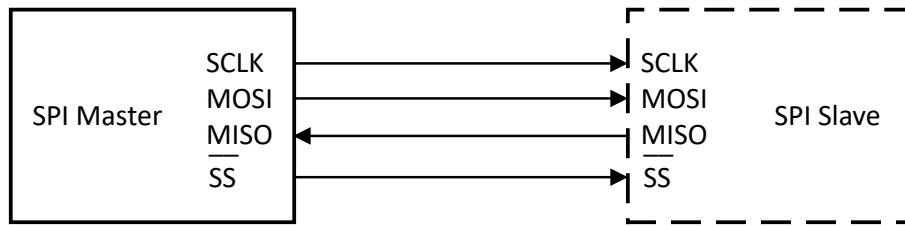


Figuur 2: MEMS accelerometer structuur [6].

## 4 De SPI interface

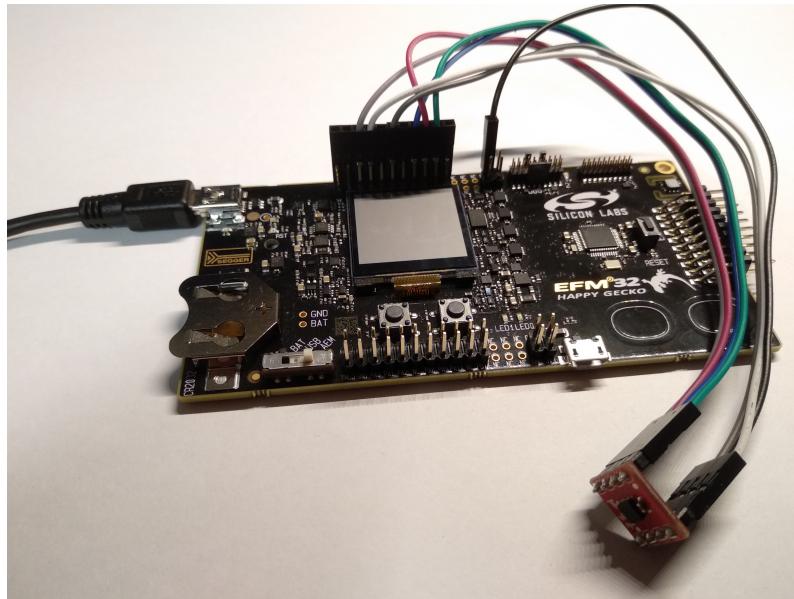
*Serial Peripheral Interface* of kortweg SPI is een seriële interface die gebruikt wordt voor korte afstandscommunicatie. Met SPI kan men full-duplex communiceren in een master-slave configuratie, waarbij het mogelijk is om meerdere slaves aan één master te verbinden. De gewenste slave wordt dan geselecteerd aan de hand van de *slave select* (of *chip select*) lijn die actief laag is.

De SPI bus bestaat uit vier lijnen: de eerder besproken slave select-lijn (SS of CS), Master Output Slave Input (MOSI), Master Input Slave Output (MISO) en de klok (SCLK). Op figuur 3 ziet men een master die verbonden is met een slave. [5]



Figuur 3: Schema SPI.

Indien er in dit labo slechts één sensor aangestuurd wordt en er bijgevolg maar één slave is kan men de SS-pin van de sensor aan massa hangen.



Figuur 4: Het Happy Gecko development-bordje met de accelerometre.

Op figuur 4 zien we hoe de sensor verbonden is met het development bordje. Op te merken is dat we de accelerometre voeden aan de hand van een GPIO pin, zodat we deze helemaal zonder voedingsspanning kunnen leggen zodat hij niets verbruikt.

## 5 Praktische uitwerking

### 5.1 Code

De code is, aangezien dit veel te veel extra bladzijden zou opleveren, niet opgenomen in dit verslag. Deze is wel te vinden op de *Github repository* van Brecht: [https://github.com/Fescron/Project-LabEmbeddedDesign1/tree/master/code/SLSTK3400A\\_ADXL362](https://github.com/Fescron/Project-LabEmbeddedDesign1/tree/master/code/SLSTK3400A_ADXL362)

#### 5.1.1 Header en Source files

Bij het ontwikkelen van code voor de Happy Gecko onderscheiden we zogenaamde **header** (.h) and **source** (.c) files. Deze worden respectievelijk geplaatst in de mappen ‘inc’ (*includes*) en ‘src’ (*sources*).

In de **header files** zetten we:

- Alle *includes* naar andere bestanden die we nodig hebben
- PORT, PIN en REGISTER definities
- Declaraties van (publieke) variabelen
- *Prototypes* van methoden

In de **source files** zetten we:

- Eén include naar de bijhorende header file
- Instantiaties van de (publieke) variabelen
- Documentatie die de methoden beschrijven
- De implementatie van de methoden zelf

### 5.1.2 Opbouw

Om het ontwikkelen van de code overzichtelijk te houden hebben we verschillende functionaliteit opgesplitst in verschillende bestanden. Deze worden hieronder iets meer in detail besproken.

**main.c** In deze *source file* zit de `main` methode waar alles van vertrekt. Ook zijn in deze file twee *initialisatiemethoden* ondergebracht.

- De eerste methode configureert de microcontroller zodat hij wakker kan worden via PIN-interrupts, dewelke kunnen gegenereert worden door drukknoppen enerzijds of door de accelerometer anderzijds.
- De andere methode configureert de **Real Time Clock Comparator** (RTCC) waardoor de microcontroller zichzelf elke minuut kan wakker maken als hij slaapt.

**debugging.h** Deze *header file* wordt aangeroepen in elke andere file waar `dbprint` (UART) statements staan. In dit bestand is een *definition* ondergebracht waarmee we al dan niet alle UART statements uit de code kunnen weglaten om zo debug-functionaliteit aan- en uit te schakelen. Dit komt doordat we deze altijd met volgende statements hebben omringd:

```
#ifdef DEBUGGING /* DEBUGGING */
    dbprintln("This text is printed to the UART console.");
#endif /* DEBUGGING */
```

**util.c/h** Hierin hebben we extra functionaliteit ondergebracht zoals:

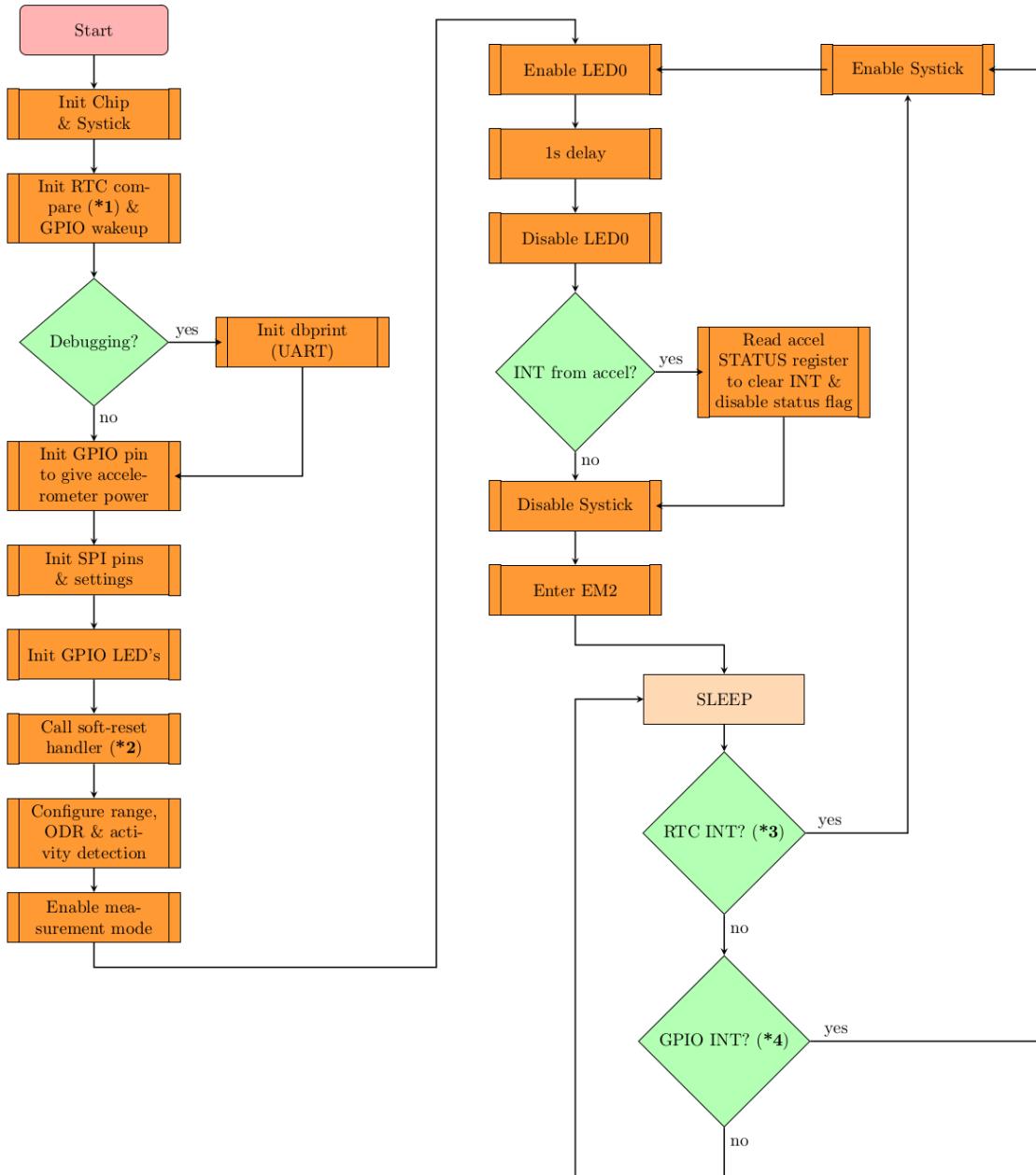
- Een functie om vertragingen te genereren met bijhorende *systick handler* en de mogelijkheid om deze *ticks* aan- en uit te zetten.
- Een methode om de LED's te initialiseren en om LED0 aan- en uit te zetten.
- Een methode om, indien er zich een probleem voordoet, het uitvoeren van de code te stoppen ('`error`') en dit weer te geven in de console én via flikkerende LED's.

**handlers.c/h** Hier zijn de *interrupt handlers* verzameld om een RTC compare interrupt of odd/even pin interrupts correct af te handelen.

**accel.c/h** Hier zijn alle methoden verzameld die iets te maken hebben met de accelerometer. Deze zijn:

- Een methode om de GPIO pin die de accelerometer voedt te initialiseren alsook een SPI initialisatiemethode.
- Een methode om de voeding naar de sensor aan- en uit te zetten.
- Een testmethode die alle ODR (Output Data Rate) waarden doorloopt om te zien wat het effect is op het stroomverbruik.
- Een methode die de X-Y-Z acceleratiewaarden uitleest en in een globale array plaatst met bijhorende methode die deze data weergeeft in de console.
- Een methode die via een andere methode het ID van de sensor controleert en al dan niet via nog een andere methode een *soft-reset* of *power-off/on* uitvoert. Zie ook nog (\*2).
- Een methode om een byte data te lezen of schrijven naar een register van de accelerometer.
- Een methode om de *ODR*, *range* en *activity threshold* van de sensor in te stellen.
- Een methode om de sensor al dan niet metingen te laten nemen.
- Een methode om via een formule een sensorwaarde om te zetten naar een ‘g’ waarde.

### 5.1.3 Code-flow



**Figuur 5:** De flowchart die de code-flow weergeeft.

Zoals we op bovenstaand flowchart kunnen zien wordt buiten het al dan niet *clearen* van de interrupt van de accelerometer (door het uitlezen van zijn statusregister) de acceleratiedata niet echt ingelezen. Dit terwijl hier wel al een methode voor bestaat. We zetten enkel de LED aan- en uit op de plaats waar we dit zouden doen. In de toekomst zou hier nog extra functionaliteit moeten toegevoegd worden om deze data te verwerken.

### Opmerkingen bij de flowchart:

- (\*1): De RTC compare functie zal elke minuut een interrupt genereren zodat de microcontroller zichzelf kan wakker maken.
- (\*2): Deze handler probeert het ID van de sensor uit te lezen om te controleren of deze correct aangesloten is. Indien dit mislukt zal de microcontroller proberen om de accelerometer te *soft-resetten*, vervolgens een seconde te wachten om daarna opnieuw zijn ID proberen uit te lezen. Indien dit nogmaals mislukt zal de voeding naar de accelerometer voor een seconde uit- en aangezet worden. Hierna wordt voor de laatste keer het ID gecontroleerd. Indien dit opnieuw mislukt zal gestopt worden met code uit te voeren en zal de ‘error’ methode aangeroepen worden.
- (\*3): Elke minuut wordt een interrupt gegenereert door de Real Time Counter Compare functionaliteit.
- (\*4): Deze interrupt kan ofwel van een knop (PB0 of PB1) komen, ofwel van een interrupt die gegenereert wordt door de accelerometer als deze detecteerd dat de ingestelde *activity threshold* is overschreden.

## 5.2 Problemen bij het ontwikkelen van de code

Bij het ontwikkelen van de code hebben we enkele problemen ondervonden die de ons serieus vertraagt hebben. Hieronder wordt het grootste probleem dat we zijn tegengekomen wat besproken.

Eerst hadden we de optie ‘autoCsEnable’ in de USART configuratie aan laten staan. Aangezien we met de juiste connecties en (volgens ons) goede instellingen geen antwoord kregen van de accelerometer hebben we de SPI-bus bekijken met een *logic analyzer*. Dit wordt hieronder weergegeven, waar we de accelerometer probeerden te *soft-resetten* (schrijven van een bepaalde waarde naar een specifiek register).

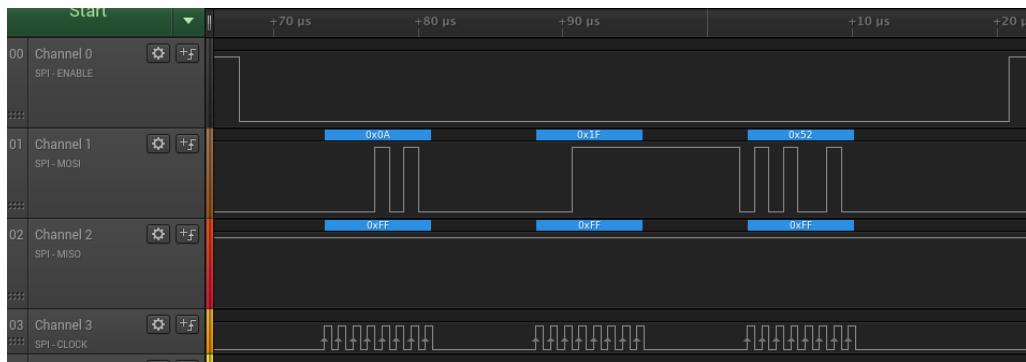


Figuur 6: Fout chipselect gedrag bij het *soft-resetten* van de accelerometer.

Uit de afbeelding hierboven kunnen we besluiten dat de chipselect lijn laag gaat bij elke byte. Volgens de datasheet [1] zou deze echter laag moeten blijven tijdens een transfer van drie bytes (‘adres register’ – ‘read/write command’ – ‘waarde om te lezen/schrijven’).

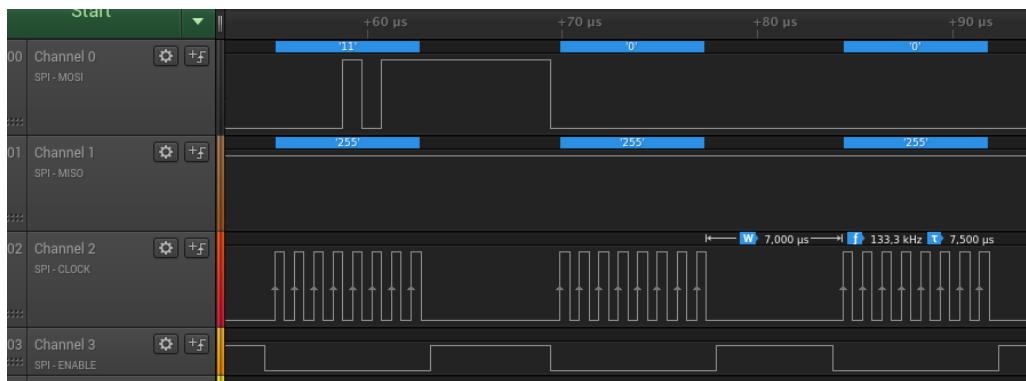
Nadat we de code hebben aangepast en we manueel de **chipselect** lijn bestuurden kregen we onderstaand (correct) gedrag indien we de accelerometer *soft-resetten*.

Op te merken is dat ‘*channel 0*’ nu de **chipselect** lijn is en de rest naar onder zijn verschoven. Ook is de de notatie van ASCII naar HEX verandert ten opzichte van figuur 6.

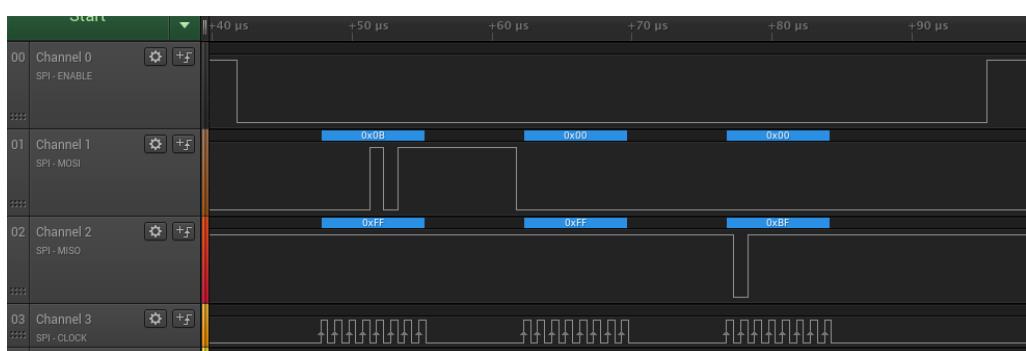


**Figuur 7:** Correct **chipselect** gedrag bij het *soft-resetten* van de accelerometer.

Hetzelfde foute **chipselect** gedrag zagen we indien we een register probeerden uit te lezen. Dit is weergegeven op figuur 8. De juiste werking is weergegeven op figuur 9.



**Figuur 8:** Fout **chipselect** gedrag bij het lezen van een register.



**Figuur 9:** Correct **chipselect** gedrag bij het lezen van een register.

### 5.3 Demo

Op figuur 11 zien we een voorbeeld van UART console-output waarbij de microcontroller alles configureert en vervolgens gaat slapen. Hierna wordt hij achtereenvolgens wakker door een interrupt van de accelerometer, van drukknop 1 en opnieuw van de accelerometer, waarna hij telkens opnieuw gaat slapen. Vervolgens wordt hij twee keer via de RTC compare functie wakker waarna hij opnieuw gaat slapen.

```
##> /dev/ttyACM0 - PuTTY
### UART initialized (no interrupts) ###
INFO: This is an info message.
WARN: This is a warning message.
CRIT: This is a critical error message.
### Start executing programmed code ###

INFO: Accelerometer powered
INFO: Accelerometer SPI initialized
WARN: Accelerometer powered down
INFO: Accelerometer powered
WARN: Soft reset ADXL done, had to "hard reset" (3 retries)
INFO: Measurement mode +- 4g selected
INFO: ODR set at 12.5 Hz
INFO: Activity configured; 3 g
INFO: Odd numbered GPIO interrupt triggered.
INT1-PD7
INFO: Measurement enabled

INFO: Disabling systick & going to sleep...
H(=) Even numbered GPIO interrupt triggered.
INT1-PD7
INFO: Disabling systick & going to sleep...
H(=) Even numbered GPIO interrupt triggered.
PB1
INFO: Odd numbered GPIO interrupt triggered.
INT1-PD7
INFO: Disabling systick & going to sleep...
H(=) INFO: Disabling systick & going to sleep...
H(=) INFO: Disabling systick & going to sleep...
```

Figuur 10: UART console output.

Op figuur 11 zien we tenslotte een voorbeeld van UART console-output waarbij de microcontroller elke 100 ms de X-Y-Z acceleratie registerwaarden uitleest en weergeeft in de console, en dus in feite de methode `readValuesADXL()`; wordt uitgevoerd.

```
##> /dev/ttyACM0 - PuTTY
### UART initialized (no interrupts) ###
INFO: This is an info message.
WARN: This is a warning message.
CRIT: This is a critical error message.
### Start executing programmed code ###

INFO: Accelerometer powered
INFO: Accelerometer SPI initialized
WARN: Accelerometer powered down
INFO: Accelerometer powered
WARN: Soft reset ADXL done, had to "hard reset" (3 retries)
INFO: Measurement mode +- 4g selected
INFO: ODR set at 12.5 Hz
INFO: Measurement enabled
[623] X: 992 mg | Y: 434 mg | Z: 310 mg
```

Figuur 11: UART console output.

## 5.4 Stroommetingen

Normaal werd de accelerometer gevoed door een GPIO pin, maar aangezien dit problemen gaf bij de stroommetingen in *Simplicity Studio* werd deze voorlopig aan VMCU gehangen. De stroommetingen zijn gedaan bij code waarbij de accelerometer is ingesteld met een *range* van  $\pm 2g$  zonder UART functionaliteit en met mogelijks een threshold van  $1g$ , tenzij anders vermeld.

### 5.4.1 ODR 12,5 Hz – activity detection uit/aan

Op figuur 12 staat het stroomverbruik afgebeeld als de microcontroller wakker is en de accelerometer de acceleratie meet maar **activity detection uit** staat, en hij dus geen interrupts zal genereren. Hier initialiseert de microcontroller *peripherals* en brandt ook de LED. **We meten een gemiddeld stroomverbruik van ongeveer 2,30 mA.**



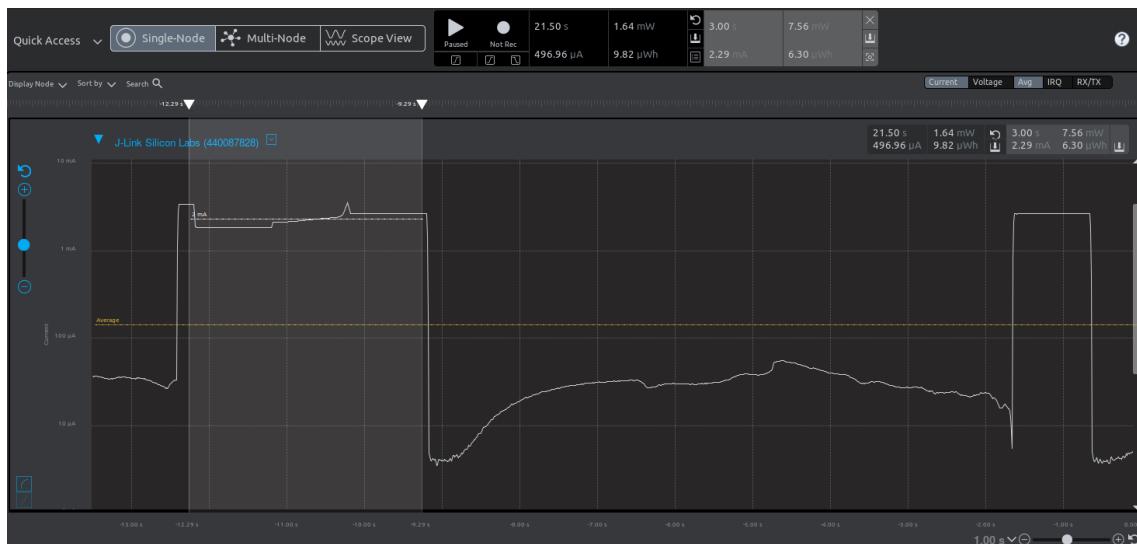
**Figuur 12:** Stroompuls als de microcontroller wakker is (activity detection staat uit).

Op figuur 13 staat het stroomverbruik afgebeeld als de microcontroller slaapt en de accelerometer de acceleratie meet maar **activity detection uit** staat. De ‘*storing*’ die we hier halverwege meten doet zich voor als er met de accelerometer bewogen wordt. **We meten een gemiddeld stroomverbruik van ongeveer 31,98  $\mu$ A.**



**Figuur 13:** Stroompuls als de microcontroller slaapt (activity detection staat uit).

Op figuur 14 staat nu het stroomverbruik afgebeeld als de microcontroller wakker is, de accelerometer de acceleratie meet én **activity detection aan** staat, en hij dus wel interrupts zal genereren. Hier initialiseert de microcontroller opnieuw *peripherals* en brandt ook de LED. **We meten een gemiddeld stroomverbruik van ongeveer 2,29 mA.**



**Figuur 14:** Stroompuls als de microcontroller wakker is (activity detection staat aan).

Op figuur 15 staat tenslotte het stroomverbruik afgebeeld als de microcontroller slaapt, de accelerometer de acceleratie meet én **activity detection aan** staat. **We meten een gemiddeld stroomverbruik van ongeveer 28,64  $\mu$ A.**

Op het einde genereert de accelerometer een interrupt waardoor de microcontroller een seconde wakker wordt en zijn LED laat branden. We zien dat deze periode veel korter is als de eerste periode waarin de microcontroller wakker is aangezien hij niet meer alles moet instellen.



**Figuur 15:** Stroompuls als de microcontroller slaapt (activity detection staat aan).

### 5.4.2 ODR 100 Hz (reset default) – activity detection uit/aan

Op figuur 16 staat het stroomverbruik afgebeeld als de microcontroller wakker is en de accelerometer de acceleratie meet maar **activity detection uit** staat, en hij dus geen interrupts zal genereren. Hier initialiseert de microcontroller *peripherals* en brandt ook de LED. **We meten een gemiddeld stroomverbruik van ongeveer 2,27 mA.**



**Figuur 16:** Stroompuls als de microcontroller wakker is (activity detection staat uit).

Op figuur 17 staat het stroomverbruik afgebeeld als de microcontroller slaapt en de accelerometer de acceleratie meet maar **activity detection uit** staat. De ‘*storing*’ die we meten doet zich voor als er met de accelerometer bewogen wordt. **We meten een gemiddeld stroomverbruik van ongeveer 36,33  $\mu$ A.**



**Figuur 17:** Stroompuls als de microcontroller slaapt (activity detection staat uit).

Op figuur 18 staat nu het stroomverbruik afgebeeld als de microcontroller wakker is, de accelerometer de acceleratie meet én **activity detection aan** staat, en hij dus wel interrupts zal genereren. Hier initialiseert de microcontroller opnieuw *peripherals* en brandt ook de LED. **We meten een gemiddeld stroomverbruik van ongeveer 2,30 mA.**



**Figuur 18:** Stroompuls als de microcontroller wakker is (activity detection staat aan).

Op figuur 19 staat tenslotte het stroomverbruik afgebeeld als de microcontroller slaapt, de accelerometer de acceleratie meet én **activity detection aan** staat. **We meten een gemiddeld stroomverbruik van ongeveer 29,67  $\mu$ A.**

Op het einde genereert de accelerometer een interrupt waardoor de microcontroller een seconde wakker wordt en zijn LED laat branden. We zien opnieuw dat deze periode veel korter is als de eerste periode waarin de microcontroller wakker is aangezien hij niet meer alles moet instellen.



**Figuur 19:** Stroompuls als de microcontroller slaapt (activity detection staat aan).

### 5.4.3 UART aan

Om nu eens te kijken of UART een grote invloed heeft op het stroomverbruik hebben we de `dbprint statements` vervolgens laten staan in de code.

Volgende twee metingen zijn gebeurt waarbij de accelerometer ingesteld was op een *range* van  $\pm 4g$  met een *threshold* voor *activity detection* van  $3g$  en een ODR van 100 Hz.

Op figuur 20 **meten we een gemiddeld stroomverbruik van ongeveer 2,33 mA** als de microcontroller wakker is en initialisaties verricht.



**Figuur 20:** Stroompuls als de microcontroller wakker is.

Op figuur 21 **meten we een gemiddeld stroomverbruik van ongeveer 32,60 μA** als de microcontroller slaapt. Op het einde genereert de accelerometer opnieuw een interrupt waardoor de microcontroller een seconde wakker wordt en zijn LED laat branden.



**Figuur 21:** Stroompuls als de microcontroller slaapt.

#### 5.4.4 Samenvatting

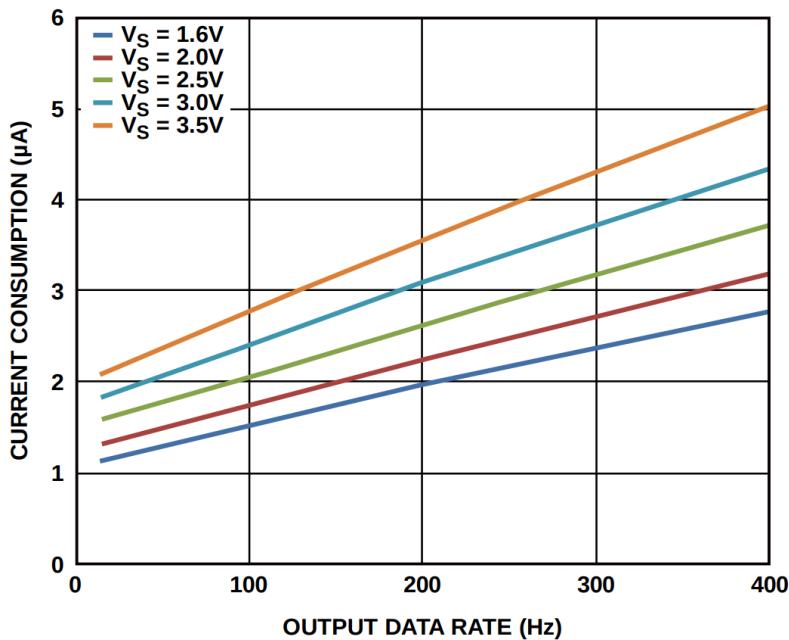
Indien we nu voorgaande gegevens in een tabel samenvoegen zodat we deze gemakkelijker kunnen vergelijken krijgen we onderstaand resultaat.

**Tabel 1:** Stroommetingen.

UART	Range	Threshold	ODR	$I_{INIT}$	$I_{SLEEP}$
Neen	$\pm 2g$	–	12,5 Hz	2,30 mA	31,98 $\mu$ A
Neen	$\pm 2g$	1g	12,5 Hz	2,29 mA	28,64 $\mu$ A
Neen	$\pm 2g$	–	100 Hz	2,27 mA	36,33 $\mu$ A
Neen	$\pm 2g$	1g	100 Hz	2,30 mA	29,67 $\mu$ A
Ja	$\pm 4g$	3g	100 Hz	2,33 mA	32,60 $\mu$ A

#### 5.4.5 Besluiten bij stroommetingen

Op figuur 22 zien we een grafiek uit de datasheet van de accelerometer [1] waarbij de verbruiksstromen zijn weergegeven bij verschillende voedingsspanningen en ODR waarden. We zien dat het verbruik bij een ODR onder 100 Hz minder dan 3  $\mu$ A zou moeten bedragen bij alle voedingsspanningen.



**Figuur 22:** Stroomverbruik van de ADXL362 accelerometer [1].

Deze waarden zijn praktisch moeilijk te controleren aangezien we met de *energy profiler* alleen moeilijk kunnen bepalen wat het stroomverbruik van enkel de accelerometer is. Door een externe stroommeter te gebruiken waarbij we de accelerometer als aparte DUT (Device Under Test) kunnen beschouwen zou dit mogelijks wel duidelijker worden.

Bij het vorige labo (**blink** voorbeeld) kwamen we een slaapstroom van de microcontroller uit dewelke iets boven de  $1 \mu\text{A}$  lag. Dit ligt nu een pak hoger, waarschijnlijk omdat de microcontroller nu ook interrupts moet controleren.

Verder zien we op tabel 1 dat voor een hogere ODR de slaapstroom ook iets hoger ligt. Waarom de stroom lager is indien er wel een threshold is ingesteld is niet helemaal duidelijk. Ook zien we dat bij het gebruik van UART we een lichte toename van het stroomverbruik waarnemen bij initialisatie.

We moeten wel de kanttekening maken dat de metingen in deze tabel niet 100% correct en vergelijkbaar zijn aangezien we de accelerometer niet telkens hetzelfde hebben bewogen en we ook niet op exact dezelfde plaats de stroompulsen hebben kunnen meten. Verder zijn dit lage stromen waarbij het meetsysteem mogelijk niet helemaal correct is.

## 6 Besluit

### 6.1 Wat hebben we verwezenlijkt?

In dit labo is een sensor aangestuurd met een microcontroller die informatie kan verwerven vanop een boei. Er is gekozen om een (MEMS) accelerometer te gebruiken. Hiermee kan men verschillende informatie over de zee opmeten, zoals golffrequentie en golfhoogtes. De gebruikte sensor wordt aangestuurd en uitgelezen via SPI. Men kan informatie opvragen door een byte naar de sensor te sturen, waarna de gevraagde informatie teruggestuurd wordt.

We hebben onze code zo modulair mogelijk proberen ontwikkelen om zo ook het overzicht beter te bewaren. Verder hebben we manieren proberen toepassen om het ontwerp een zo laag mogelijk vermogen te laten hebben. Als laatste hebben we verschillende stroommetingen gedaan en deze proberen interpreteren.

### 6.2 Wat kan er beter? (denk verder dan de opgave)

Zoals al eerder vermeld lezen we eigenlijk niet echt data uit de accelerometer uit maar *clearen* we enkel de interrupt ervan (als dit nodig is) door het uitlezen van zijn statusregister. Dit terwijl wel al een methode bestaat om accelatiewaarden uit te lezen. We zetten enkel de LED aan- en uit op de plaats waar we dit zouden doen. In de toekomst zou hier nog extra functionaliteit moeten toegevoegd worden om deze data te verwerken.

Verder zouden we deze data dan kunnen verwerken om iets te vertellen over de golfhoogte en/of golffrequentie. Ook kunnen we nog de accelerometer in een *wakeup-mode* instellen waarbij hij ongeveer  $270\text{ nA}$  zal verbruiken en ongeveer zes keer per seconde controleert of er al dan niet beweging is. Indien hij beweging detecteert kan hij dan op een autonome manier reageren door bijvoorbeeld een interrupt te genereren of sneller te gaan meten.

### 6.3 Hoe kunnen we dit tot stand brengen?

De golffrequentie zou kunnen gemeten worden door minimum 0,4 samples per seconde te meten. Men zou dit kunnen verwezenlijken met een goed gekozen ODR in samenwerking met het FIFO register van de accelerometer. Zolang dit register niet vol is kan de microcontroller zich in slaap houden. Indien deze wel vol zit kan de accelerometer een interrupt genereren naar de microcontroller die dan deze FIFO uit kan lezen en de data eruit verwerken.

## Referenties

- [1] Datasheet adxl362. <https://www.analog.com/media/en/technical-documentation/data-sheets/adxl362.pdf>.
- [2] Sparkfun triple axis accelerometer breakout - adxl362. <https://www.sparkfun.com/products/11446>.
- [3] What dictates the frequency of waves? <https://www.thenakedscientists.com/articles/questions/what-dictates-frequency-waves>.
- [4] J. N. Walpert en S. D. Howden L. C. Bender III, N. L. Guinasso JR. A comparison of methods for determining significant wave heights—applied to a 3-m discus buoy during hurricane katrina. <https://journals.ametsoc.org/doi/pdf/10.1175/2010JTECH0724.1>.
- [5] Inc. Motorola. Spi block guide v04.01. [https://www.nxp.com/files-static/microcontrollers/doc/ref\\_manual/S12SPIV4.pdf](https://www.nxp.com/files-static/microcontrollers/doc/ref_manual/S12SPIV4.pdf).
- [6] Kieran Harney Rob O'Reilly, Alex Khenkin. Sonic nirvana: Using mems accelerometers as acoustic pickups in musical instruments. <https://www.analog.com/en/analog-dialogue/articles/mems-accelerometers-as-acoustic-pickups.html>.