|  |  |
| --- | --- |
| http://www.mwq.be/servlet/Repository/?IDR=2988&IDQ=20 | Helha – Campus Technique  Master en Electronique  Année académique 2015-2016 |
| **Projet Electronique** | |

Projet Tilt and Shocks Sensor



Julien Delvaux - Michael Manzella - Loic Quinet

Contenu

[1. Introduction 1](#_Toc449196149)

[2. Analyse de l’existant et cahier des charges 2](#_Toc449196150)

[2.1. Analyse de l’existant 2](#_Toc449196151)

[2.2. Cahier des charges 2](#_Toc449196152)

[2.3. PICOS18 2](#_Toc449196153)

[2.4. Capteurs 2](#_Toc449196154)

[2.4.1. Température 2](#_Toc449196155)

[2.4.2. Accéléromètre 2](#_Toc449196156)

[2.5. EEPROM 2](#_Toc449196157)

[2.6. Android 2](#_Toc449196158)

[3. Programmation PIC 3](#_Toc449196159)

[3.1. PICOS18 3](#_Toc449196160)

[3.2. Accéléromètre 3](#_Toc449196161)

[3.3. Capteur température 3](#_Toc449196162)

[3.4. Mémoire EEPROM 4](#_Toc449196163)

[3.5. Android 4](#_Toc449196164)

[4. Programmation Android 4](#_Toc449196165)

[5. Résultats 4](#_Toc449196166)

[5.1. Liste des fonctionnalités implémentées 4](#_Toc449196167)

[5.2. PIC 4](#_Toc449196168)

[5.3. Application Android 4](#_Toc449196169)

[5.4. Consommation 4](#_Toc449196170)

[5.5. Conclusion 4](#_Toc449196171)

[6. Conclusion 4](#_Toc449196172)

[7. Bibliographie 5](#_Toc449196173)

# Introduction

# Analyse de l’existant et cahier des charges

## Analyse de l’existant

Le projet tel que nous l’avons reçu était au stade de preuve de concept. Cela implique que le code était brouillon, non-optimisé. De plus, nous n’avons reçu aucune documentation concernant ce qui avait été fait. Il nous a fallu un peu de temps pour nous plonger dans le code, comprendre comment cela fonctionnait. Le challenge en plus du PICOS, était d’apprendre à manipuler le PIC avec lequel nous n’avions aucune connaissance.

Au niveau hardware, la carte électronique était déjà réalisée et avait été testée par Denis Alaime. Lors de notre prise en main, nous avons eu l’occasion de vérifier que tout fonctionnait correctement.

Certaines choses ont déjà été réalisées au niveau de la programmation du PIC. Le PICOS18 était fonctionnel et le programme réussissait à communiquer avec les capteurs (accéléromètre et température) et la mémoire EEPROM.

Denis Alaime utilisait une application Android développée par STMicroelectronics afin de vérifier qu’il arrivait à écrire correctement dans l’EEPROM. Il n’avait rien créé comme application Android.

## Cahier des charges

Nos objectifs pour ce projet étaient les suivants :

1. Garder le PICOS18
2. Faire clignoter la LED à intervalles réguliers
3. Ajouter des fonctionnalités au prototype.
4. Optimiser la consommation afin de faire durer la batterie le plus longtemps possible.
5. Développer l’application Android customisée pour ce projet.

Pour ce faire, nous avons divisé le groupe en deux :

* Mike Manzella a travaillé au développement de l’application Android
* Loic Quinet et Julien Delvaux ont travaillé avec la carte électronique : PIC et capteurs/mémoire

## PICOS18

Le PICOS18 est un OS temps réel (*real-time OS*) développé par la société Pragmatech sous licence GPL (open-source). La version actuelle, numérotée 3, est sortie en 2007. La norme OSEK permet de définir le fonctionnement interne de l’OS comme la liste des services du noyau et la gestion interne des tâches et des ressources.

Le noyau temps-réel fonctionne autour de trois grands axes :

* Le noyau : il regroupe un ensemble de fonctionnalités appelées services dont un exemple typique est le *scheduler*, responsable de la gestion de celles-ci. Il garantit également la stabilité du système et contrôle les ressources.
* Multitâches : Plusieurs tâches peuvent fonctionner de manière séparée et sont régulées par le noyau. Si celui-ci le permet, il est possible de donner l’impression qu’elles s’exécutent en parallèle. Le noyau est dit « multitâches préemptif » s’il est à même de gérer ce fonctionnement parallèle. Dans le cas contraire, on le dénomme « multitâche coopératif ».
* Temps-réel : Le noyau multitâche alloue un temps égal et une zone mémoire identique pour chaque tâche. Cependant, celle-ci ont rarement la même priorité et elles doivent être appelées le plus rapidement possible. Plutôt que d’essayer d’avoir un temps de réactivité quasi-nul (impossible en pratique), le noyau garantit temps de latence constant : c’est le déterminisme.

## Capteurs

## Température

## Accéléromètre

L’accéléromètre est le FXLS8471Q de Freescale Semiconductor. C’est un accéléromètre 3 axes. Voici quelques-unes de ses caractéristiques :

* Echelle de ± 2g, ± 4g ou ± 8g
* Fréquence de capture de 1.563 Hz à 800Hz
* ADC interne de résolution 14-bit
* Fonctionnalités programmables :
  + Détection de mouvement et chute libre
  + Détection d’orientation
  + Détection de « tap »
* Mode sleep
* Fonctionnement en I²C ou SPI

Son fonctionnement est décrit par la Figure 1. La fonction d’acquisition est réalisée par les blocs verts. Grâce aux transformations internes de l’accéléromètre, les valeurs acquises sont numérisées. Le bloc *Embedded DSP Functions* regroupe toutes les fonctionnalités possibles avec l’accéléromètre.

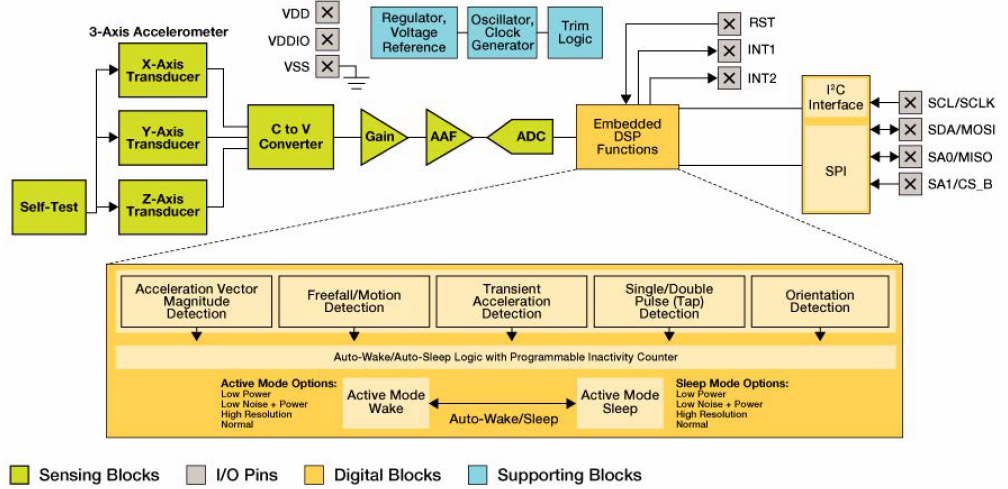


Figure 1 Schéma fonctionnement FXLS8471Q

## EEPROM

## Android

# Développement

## PICOS18

Le microcontrôleur utilise le PICOS18 pour fonctionner. Dans notre cas, nous avons une tâche principale (*TASK\_Main*) et deux tâches secondaires nécessaire par l’OS : une pour l’I2C et une autre pour l’UART (utile pour le débug).

Le fonctionnement de la tâche principale est illustré à la Figure 2. L’idée est de mettre le PIC en mode *SLEEP* dès que possible afin de minimiser sa consommation autant que possible. Uniquement lorsqu’il y a une interruption, on va passer le microcontrôleur en mode *RUN* afin de traiter celle-ci. Il y a trois sources d’interruptions possible :

1. RTCC : la *Real-Time Clock and Calendar* va permettre de faire clignoter la LED (toutes les 2 secondes), de lire la configuration au niveau de l’EEPROM (toutes les 20secondes si la variable isRF\_WIP\_BUSY a été modifiée) et réécrire la date et heure dans l’EEPROM (toutes les 10 minutes).
2. EEPROM : Dès qu’une personne approche son GSM, une variable (isRF\_WIP\_BUSY) est modifiée.
3. Accéléromètre : Lorsqu’un retournement ou une détection d’accélération a été détectée.

Le clignotement de la LED est effectué grâce au lancement d’un TIMER afin de réduire de manière significative la consommation du microcontrôleur.



Figure Fonctionnement de la tâche principale

## Accéléromètre

La programmation de l’accéléromètre n’a pas été facile à réaliser. Les fichiers sources donnés par Denis Alaime contenaient un peu de configuration, comment récupérer les valeurs mais ce n’était pas lisible et facile à s’y plonger en venant de l’extérieur.

J’ai choisi de revoir la manière dont était faite cette librairie afin de (me) faciliter le travail par la suite. L’idée étant d’avoir des fonctions et des arguments qui soient suffisamment explicites que même quelqu’un n’ayant pas lu la datasheet puisse comprendre l’idée générale.

Il y a trois grand types de fonctions dans cette librairie :

1. Fonctions pour faire la communication I2C avec l’accéléromètre : regroupe les fonctions nécessaires par l’OS pour la communication I2C et des fonctions pour l’écriture/lecture de registre.
2. Fonctions privées à la librairie : appelées par les fonctions publiques.
3. Fonctions publiques : fonctions auxquelles le développeur peut faire appel dans son code.

Bien souvent, les fonctions publiques font appel aux fonctions privées (rarement à la première catégorie) avant de faire appel aux fonctions I2C. Une fois que la première catégorie a été réalisée, ce qui a été un gros challenge, le développement du reste n’a pas posé de réel soucis.

L’initialisation se passe de la façon suivante et est illustré par la Figure 3 :

1. Test de connexion avec le module
2. Configuration et calibration de l’accéléromètre

Lorsqu’un évènement non autorisé par l’utilisateur (dans notre cas : retournement et accélération trop importante) est détecté par l’accéléromètre, une interruption a lieu sur le microcontrôleur. Celui-ci passe alors par une fonction dans la librairie de l’accéléromètre afin de connaître la source de cette interruption. Si l’interruption était justifiée, un message est écrit dans l’EEPROM. Ce fonctionnement est illustré par la Figure 4.



Figure Initialisation de l'accéléromètre



Figure Interruption accéléromètre

## Capteur température

## Mémoire EEPROM

## Android

# Résultats

## Liste des fonctionnalités implémentées

## PIC

## Application Android

## Consommation

## Conclusion

# Conclusion

# Bibliographie

Références principales :

|  |  |
| --- | --- |
| [ 1 ] | L. Jones, “Coordination and Control for Multi-Quadrotor UAV Mission”, Thesis, Naval Postgraduate School, 2012. |