



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

DEVELOPPEMENT D'UN SYSTEME EMARQUE COMMUNICANT ASSOCIE A UN PSOC POUR LA COMMANDE D'ORGANES

RAPPORT DE PROJET TUTORE

IUT DE BELFORT- MONTBELIARD

Guillermo Alan VEGA GONZALEZ

Diplôme préparé Licence
professionnelle VEGA

M. Christophe LOMBARD



Contents

1. Généralités du Projet	3
1.1 Introduction	3
1.2 Objectif du Projet	3
1.3 Description du Projet et contexte	3
2. Solution Proposée	3
2.1 Description de la Solution	3
2.2 Tâches à réaliser	4
2.3 Description des Composants	4
2.3.1 Accéléromètre et Communication I2C	4
2.3.2 PSoC 4 et Communication UART	4
2.3.3 Module Bluetooth	5
2.4 Description des Logiciels	5
2.4.1 PSoC Creator	5
2.4.2 Bootloader Host	5
2.4.3 TeraTerm	5
2.4.4 Design Spark PCB	5
2.4.5 FreeCAD	5
2.5 Organigramme de fonctionnalité	6
3. Mode d'usage du projet	6
3.1 Information général d'usage	6
3.2 Position Neutral	7
3.3 Changer le volume de l'autoradio	7
3.4 Changer la station de l'autoradio	7
4. Conclusion	8
5. Bibliographie	9
Bibliography	9
6. Annexes	10
6.1 Programmes	10
6.1.1 Programme Bootloader	10
6.1.2 Programme Bootloadable	11
6.1.3 Configuration du module Bluetooth	19
6.2 Conception Carte Autonome	20
6.2.1 Liste de composants	20
6.2.2 Schéma électrique	20
6.2.3 Design PCB	21
6.2.4 Visualisation 3D	21
6.3 Design Boîtier	22
6.4 Diagramme de Gantt	22



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

1. Généralités du Projet

1.1 Introduction

A nos jours, la technologie devient de plus en plus complexe avec plus d'applications dans nos activités quotidiennes afin d'augmenter le confort, la sécurité ou la fonctionnalité. C'est aussi le cas pour l'électronique dans le secteur automobile, elle est devenue si importante que l'électronique automobile est actuellement un domaine de formation et de recherche très grande dans le monde industriel.

A mesure que la technologie est développée, le nombre d'applications possibles dans l'automobile augmente. La manière d'interaction entre le conducteur et la voiture a évolué depuis quelques années, dans l'électronique on est arrivé au point de pouvoir interagir et commander la voiture même avec la voix afin d'augmenter le confort et au même temps la sécurité. C'est grâce à ce type d'application que ce projet est conçu.

1.2 Objectif du Projet

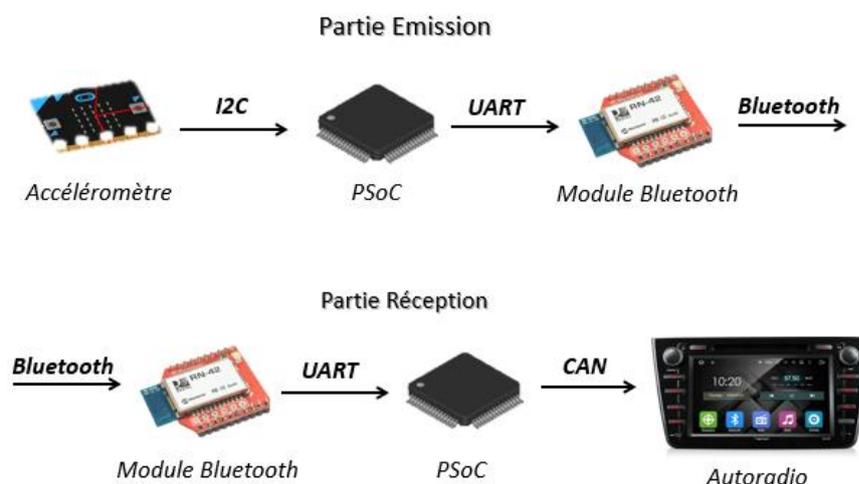
Interagir à distance sur des organes véhicule (ex : radio) par le biais d'un accéléromètre.

1.3 Description du Projet et contexte

Le projet met en œuvre un système embarqué autonome associé à un accéléromètre 3 axes (boîtier ou bracelet) connecté en Bluetooth à un système embarqué sur le véhicule. Celui-ci reçoit et décode les informations de l'accéléromètre pour ensuite piloter la radio (son et stations) par le réseau CAN.

2. Solution Proposée

2.1 Description de la Solution



La solution proposée pour ce projet c'est un système embarqué séparé en deux parties. La partie émission, utilisera un microcontrôleur PSoC 4 Cy8C4247AZI afin de recevoir les commandes d'un accéléromètre ADXL345 avec une liaison I2C et les envoyer au module Bluetooth RN42XVP via UART afin d'émètre par Bluetooth les commandes de l'utilisateur à la partie réception du projet pour manipuler l'autoradio.



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

La partie réception reçoit les commandes par Bluetooth avec un autre module RN42XVP et les envoie via UART à un autre PSoC qui finalement transmet cette information à l'autoradio avec les trames CAN correspondant par un réseau CAN Low Speed.

2.2 Tâches à réaliser

- A) Découvrir les composants à utiliser (l'accéléromètre, le PSoC et le module Bluetooth), ses logiciels (PSoC Creator, 232 Analyzer, TeraTerm et Design Spark) et les protocoles de communication à implémenter (I2C, UART et Bluetooth).
- B) Programmes de test afin de bien comprendre le fonctionnement des composants et communications.
- C) Programme de test avec le fonctionnement complet de la partie émission du projet
- D) Conception d'une carte électronique autonome dédiée à réaliser la partie émission du projet
- E) Assemblage du boîtier
- F) Test et validation du projet

*Note : Ce rapport ne contient que les tâches qui intéressent la partie émission

*Note : Dans la partie annexes vous pourrez trouver un diagramme de Gantt du projet avec toutes les activités.

2.3 Description des Composants

2.3.1 Accéléromètre et Communication I2C

Le composant ADXL345 c'est un accéléromètre numérique 3 axes, avec 10 bits de résolution, jusqu'à $\pm 16g$, et avec une sensibilité de 4mg/LSB.

Il supporte les communications I2C et SPI parmi lesquelles on a choisi une liaison I2C (Inter Integrated Circuit) avec une vitesse de 100Kbps qui est un protocole série, synchrone, bidirectionnel et simple à utiliser pour se communiquer avec le PSoC. Il y a une notion maître/esclave, c'est une communication avec deux fils, un SCL(Clock) qui synchronise la communication et un fil de data (SDA).

2.3.2 PSoC 4 et Communication UART

PSoC (*Programable System on Chip*), c'est une famille de microcontrôleurs développés par l'entreprise Cypress Semi-conducteur. Le PSoC utilisé dans ce projet est le PSoC 4 Cy8C4247AZI-M485 qui a une alimentation de 5 ou 3.3v. C'est un composant avec des horloges, PWM, ADC 12 bits et qui supporte les protocoles de communications tels que I2C, SPI, UART, CAN.

En utilisant une liaison UART on se communique avec le module Bluetooth. Le protocole UART (*Universal Asynchronous Receiver Transmitter*) c'est une liaison série, asynchrone et bidirectionnel simple à utiliser. C'est une communication deux fils, un fil de transmission Tx et un fil de réception Rx qui sont unidirectionnel. Les paramètres de cette communication sont 115200 bps, 8 bits de data, 1 bit de stop et none parity bits.



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

2.3.3 Module Bluetooth

Le module Bluetooth utilisé c'est le RN42XVP de Microchip. Il est alimenté en 3.3v, il supporte les versions 2.0, 1.2 et 1.1 de Bluetooth. Afin de le configurer il faut utiliser soit une connexion Bluetooth ou soit un émulateur de terminal pour envoyer des commandes AT et établir les paramètres désirés pour établir la communication.

2.4 Description des Logiciels

2.4.1 PSoC Creator

PSoC Creator c'est un logiciel dédié à créer, éditer, debugger et compiler le hardware et firmware des systèmes PSoC et FM0+. C'est logiciel utilise une partie en langage C et en langage graphique afin de réaliser une architecture programmable.

2.4.2 Bootloader Host

Bootloader Host c'est un logiciel de téléchargement développé par l'entreprise Cypress Semi-conducteur qui permet le téléchargement de programmes Bootloadables dans un espace réservé pour eux par un programme Bootloader qui a été précédemment téléchargé sur un projet ou système.

2.4.3 TeraTerm

TeraTerm c'est un logiciel "open-source" qui permet l'émulation de terminal de communications de DEC VT100 à DEC VT382. Ce logiciel supporte telnet, SSH 1 & 2 et les connexions du port serial.

2.4.4 Design Spark PCB

Design Spark c'est un logiciel de design électronique dédié à la conception des cartes électroniques. Il y a la partie Schématique et la partie création du PCB.

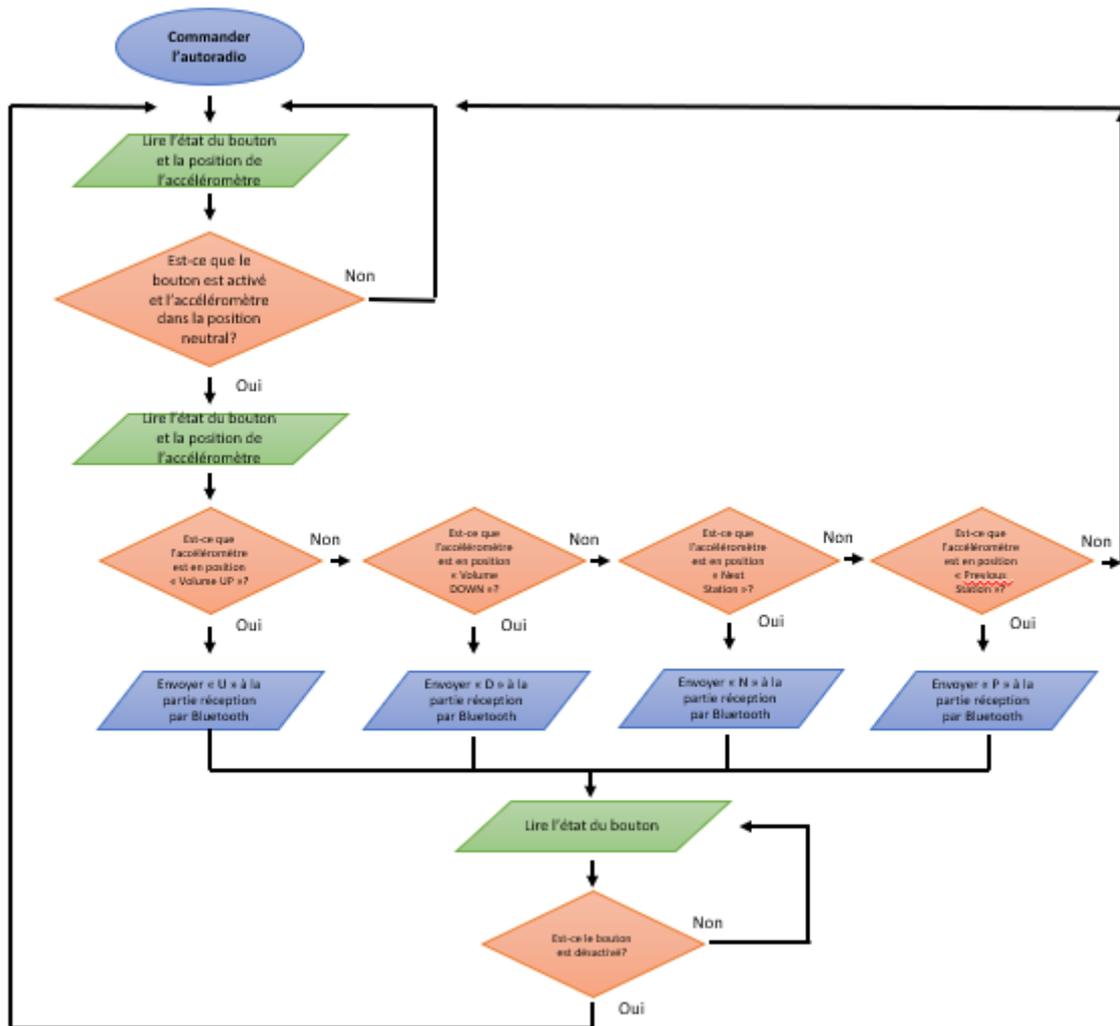
2.4.5 FreeCAD

FreeCAD est un logiciel de modélisation 3D libre pour la CAO, IAO, PLM, orientée vers le génie mécanique et le design de produits, mais qui vise également d'autres disciplines, telles que l'architecture ou d'autres branches de l'ingénierie.



2.5 Organigramme de fonctionnalité

Dans le schéma suivant on peut voir d'une façon graphique la fonctionnalité du projet de la partie émission.



3. Mode d'usage du projet

3.1 Information général d'usage

Le but du projet c'est la commande d'organes automobiles. Dans ce projet on commande l'autoradio en faisant des mouvements avec la main. L'utilisateur aura un bracelet autour de son poignet avec le système embarqué qui a l'accéléromètre pour identifier les mouvements réalisés par l'utilisateur et donc la commande désirée.

Pour commencer la requête d'un changement dans l'autoradio l'utilisateur devra d'abord avoir la main dans la position neutral et appuyer sur le bouton poussoir et faire qu'il reste poussé jusqu'à la fin des mouvements de la commande. Les mouvements doivent être réalisés doucement.



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

Si l'utilisateur souhaite réaliser une nouvelle commande il devra appuyer de nouveau sur le bouton, c'est-à-dire pour chaque commande on doit commencer dans la position neutral et appuyer sur le bouton après faire le mouvement correspondant et retourner à la position neutral et finalement lâcher le bouton.

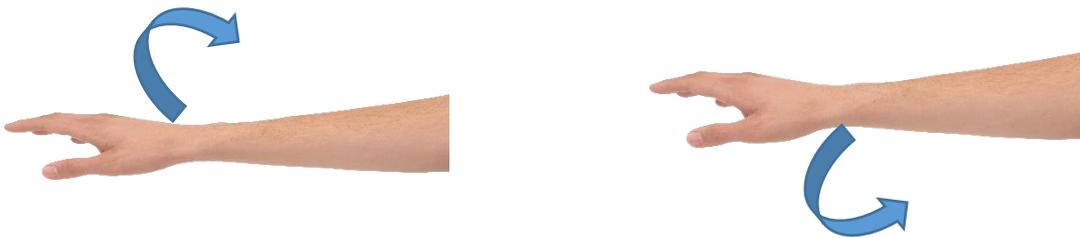
3.2 Position Neutral

La position neutral c'est soutenir la main horizontalement afin de commencer une commande.



3.3 Changer le volume de l'autoradio

Afin d'augmenter le volume de l'autoradio on doit commencer la commande dans la position neutral et après lever la main 90° en position vertical. Pour diminuer le volume de l'autoradio on doit commencer la commande dans la position neutral et après descendre la main 90° en position vertical.



3.4 Changer la station de l'autoradio

Afin de changer à la station suivante on doit commencer la commande dans la position neutral et après tourner le poignet 90° vers la droite. Pour changer à la station précédente on doit commencer la commande dans la position neutral et après tourner le poignet 90° vers la gauche.





GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

4. Conclusion

Le développement de ce projet m'a permis de mettre en œuvre mes connaissances techniques dans le domaine de l'électronique analogique et numérique, la conception de cartes électroniques, le design de pièces mécaniques en logiciel et surtout m'a permis d'augmenter mes connaissances dans le domaine de la programmation. En plus, ce projet en particulier m'a poussé à développer mes capacités d'autonomie et recherche grâce à tous les logiciels utilisés tout au long du développement du projet, lesquels je n'avais aucune connaissance avant de commencer tels que PSoC Creator, FreeCAD, TeraTerm, Design Spark, Bootloader Host et 232 Analyzer.

Parmi les problèmes techniques rencontrés au début du projet, j'ai eu une difficulté avec la complexité du PSoC parce que c'est une technologie très puissante pourtant elle n'est pas très utilisée et donc il n'y a pas autant d'information de support que pour les autres microcontrôleurs, alors la compréhension et la prise en main du logiciel et du composant lui-même a été vraiment difficile.

Après afin d'utiliser l'accéléromètre avec le PSoC, on doit configurer et lire les registres du composant, normalement cela est facile à faire grâce aux bibliothèques pour les microcontrôleurs cependant en PSoC il n'y a pas de bibliothèques et donc il a fallu que je la fasse, cela a représenté vraiment un défi très grand au niveau de la programmation.

On a eu un retard dans le développement du projet à cause d'une hésitation à propos du composant Bluetooth à utiliser et finalement on est resté avec le module RN42XVP grâce à sa facilité de mise en place. Aussi on a eu une grande période de retard à cause d'un délai d'environ 2 mois d'une commande de composants parmi lesquels était le PSoC, et donc même si on avait les programmes finals prêts à utiliser sur les modules de test qui marchait correctement, on n'avait pas la possibilité de réaliser de tests avec le PSoC isolé du module car il n'était pas arrivé et parce que c'est un composant « CMS » (composant monté en surface) qu'on ne peut pas tester sur une plaque LABDEC.

Ensuite on a eu de problèmes dans le design de la carte électronique, les pistes étaient très petites, et il y avait quelques mauvaises connexions qui ont été faites, pourtant on a réussi à tout régler.

Finalement on se rencontre avec un problème qu'on n'a pas réussi à résoudre. Lorsqu'on avait fini la carte on n'a pas réussi à la programmer, dans le logiciel on a un message d'un problème de communication avec la carte et donc ce problème ne nous permet pas de programmer le PSoC et donc on ne peut pas utiliser la carte autonome mais le module sur lequel on a fait les essais. Le problème avec la carte d'après le logiciel c'était un souci d'acquiescement avec le composant, aussi on avait un grand problème de consommation de courant avec la carte lorsqu'on essayait d'alimenter en 5V, la carte demandait jusqu'à 8mA qui est trop d'énergie pour un microcontrôleur. Après tous les essais afin de résoudre ce problème, je l'impression que le PSoC ne marche plus car il y a un court-circuit quelque part à l'intérieur du PSoC.

**GEII**Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

Le projet a été vraiment un défi pour moi, quasiment tout était nouveau, depuis les logiciels et les composants jusqu'au l'environnement de travail à cause de la langue laquelle a représenté un petit challenge au début. Pourtant avec les conseils de mes enseignants, la documentation des composants, les vidéos et information disponibles sur internet, j'ai réussi à réaliser presque la totalité du projet. C'est pour cette raison là que finalement je suis satisfait grâce à toutes les nouvelles connaissances acquissent.

5. Bibliographie

Bibliography

Components, R. (2013). *Design Spark PCB Prise en Main*. Oxford: RS Components.

Devices, A. (2011). *3-Axis, ± 2 g/ ± 4 g/ ± 8 g/ ± 16 g Digital Accelerometer ADXL345*. Norwood: Analog Devices.

Networks, R. (2012). *RN41XV & RN42XV Bluetooth Module*. Los Gatos: Roving Networks.

Networks, R. (2013). *Bluetooth Data Module Command*. California: Roving Networks.

Printer, C. (2016, December 11). *Freecad Tutoriel 01: Bien débiter, Navigation en 3D et 1ère Pièce*.

Retrieved from Youtube: https://www.youtube.com/watch?time_continue=4&v=SCPF2ob3p1Y

SEMICONDUCTOR, C. (2015). *CY8CKIT-043 PSoC 4 M-Series Prototyping Kit*. San José: CYPRESS SEMICONDUCTOR.

Semiconductor, C. (2015). *CY8CKIT-043 PSoC® 4 M-Series Prototyping Kit Guide*. San José: Cypress Semiconductor.

Su, K. S. (2015, July 13). *PSoC4 I2C & UART Tutorial: Reading Over I2C*. Retrieved from Youtube: <https://www.youtube.com/watch?v=hZW6Ytx1BGM&t=2s>

Su, K. S. (2015, July 15). *PSoC4 I2C & UART Tutorial: Sending Data Over UART to MATLAB*. Retrieved from Youtube: <https://www.youtube.com/watch?v=tRZ0t7pupLs>

Su, K. S. (2015, July 14). *PSoC4 I2C & UART Tutorial: Writing Over I2C and Reading in Data*. Retrieved from Youtube: <https://www.youtube.com/watch?v=9XmfQcOTwuk&t=4s>

Williams, N. (2015, February 12). *Connect a pair of RN-42 modules together*. Retrieved from Youtube: <https://www.youtube.com/watch?v=6ZMysCpwlM8>



6. Annexes

6.1 Programmes

Lorsqu'on développe une carte autonome on doit penser à qu'elle soit vraiment autonome en utilisant les programmes Bootloader et Bootloadable afin que la carte autonome soit reprogrammable par n'importe quel programmeur et ne pas uniquement par le programmeur du même fabricant du microcontrôleur.

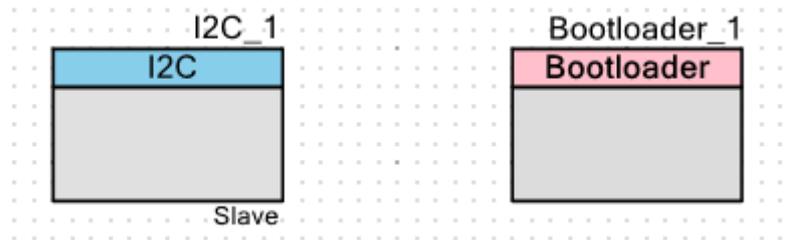
Le programme Bootloader c'est un programme qui sera inséré dans la mémoire Flash du microcontrôleur avec une communication tels que I2C ou SPI et gardera une espace pour le programme Bootloadable.

Le programme Bootloadable aura le fonctionnement désiré du système et l'adresse du Bootloader pour arriver à lui trouver et s'insérer dans cette espace de mémoire. Alors lorsqu'on souhaite reprogrammer le système, on modifiera le Bootloadable et pas le Bootloader.

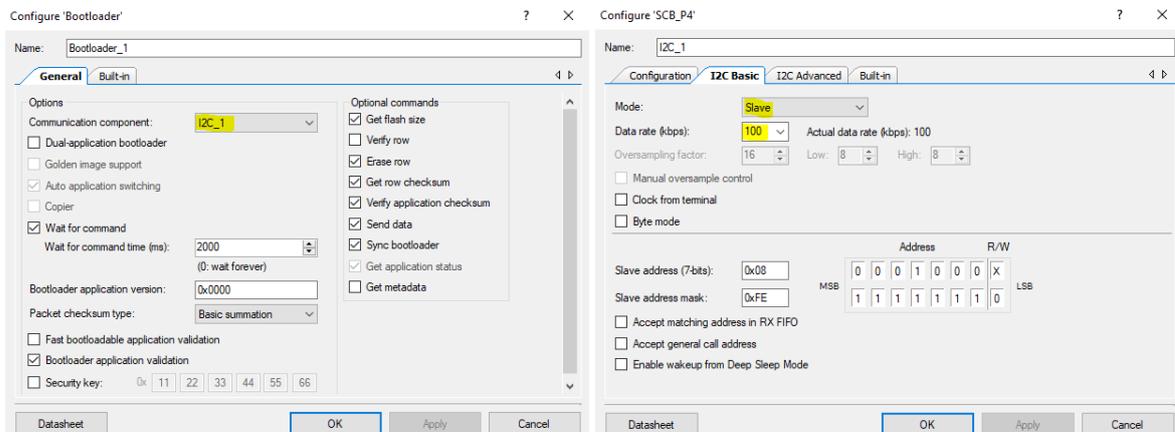
6.1.1 Programme Bootloader

6.1.1.1 Partie Graphique

Dans la fenêtre « Top Design » on a placé les composants Bootloader et I2C (SCB mode).



La configuration du composant Bootloader c'est simplement choisir la communication I2C et pour le composant I2C on doit choisir en mode « Slave » avec une vitesse de 100 kbps.





GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

6.1.1.2 Configuration de Pins

Pour les Pins on doit selectionner le port P4.0 pour le SCL et P4.1 pour le SDA.

	Name	Port	Pin	Lock
<input checked="" type="checkbox"/>	\I2C_1:scl\	P4[0]	27	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	\I2C_1:sda\	P4[1]	28	<input checked="" type="checkbox"/>

6.1.1.3 Partie Code C

La partie code C c'est simplement une ligne qui sert à mettre en route le bloque Bootloader.

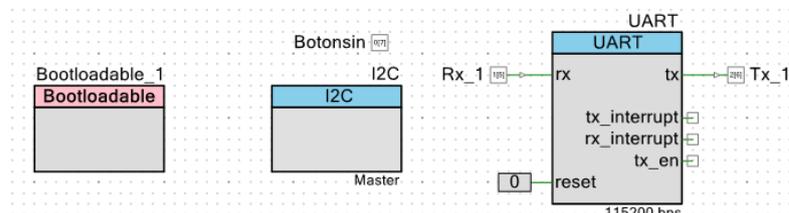
```
12 #include "project.h"
13
14 int main(void)
15 {
16     CyGlobalIntEnable; /* Enable global interrupts. */
17
18     /* Place your initialization/startup code here (e.g. MyInst_Start()) */
19     CyBldr_Start();
20
21     for(;;)
22     {
23         /* Place your application code here. */
24     }
25 }
26
27 /* [] END OF FILE */
```

Finalement on doit compiler et télécharger le programme normalement sur le PSoC en cliquant sur l'onglet « Build » et sélectionner « Build (Nom du Projet) » et après dans l'onglet « Debug » sélectionner « Program », de cette façon le programme Bootloader sera stocké dans la mémoire Flash du PSoC.

6.1.2 Programme Bootloadable

6.1.2.1 Partie Graphique

Dans la fenêtre « Top Design » on a placé les composants Bootloadable, I2C (SCB mode), UART et un Digital Input appelé « Botonsin ».



La configuration du composant Bootloadable c'est sélectionner la route des fichiers « HEX file » et « ELF file » du projet Bootloader qui ont été créés au moment de la compilation du programme Bootloader. Cela sert au programme Bootloadable afin de trouver la location du programme Bootloader dans la mémoire flash du PSoC et qu'il puisse se stocker dans cette partie réservée pour lui.



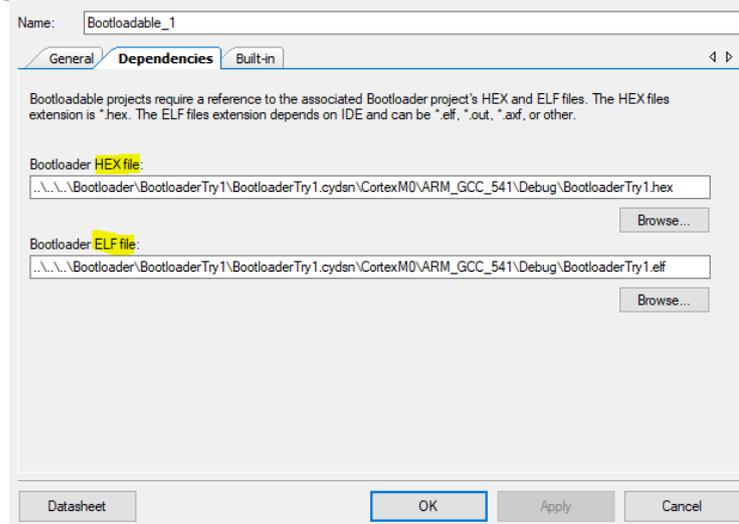
GEII

Département Génie Électrique

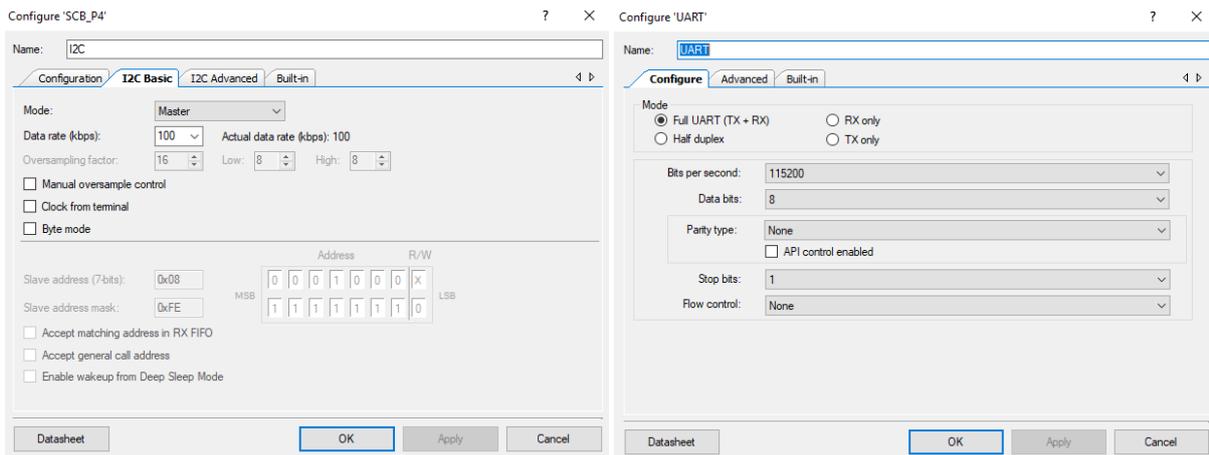
& Informatique Industrielle

IUT Belfort-Montbéliard

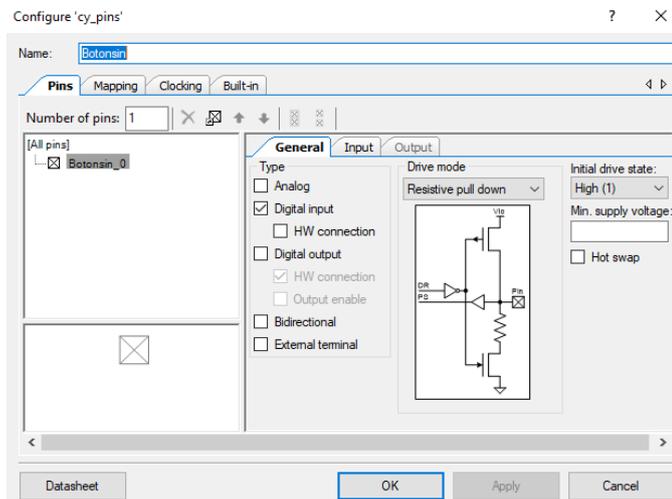
Configure 'Bootloadable'



Pour le composant I2C on doit choisir en mode « Master » avec une vitesse de 100 kbps et pour le bloque UART, 115200 bps, 8 bits de data, aucun bit de parité et 1 bit de stop.



Pour l'entrée numérique « Botonsin » on aura les configurations suivantes : un « Drive Mode » de « Resistive Pull Down » et un état initial « High (1) ».





GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

6.1.2.2 Configuration de Pins

Afin d'avoir la bonne configuration des Pins on doit sélectionner le port P1.0 pour le SCL, P1.1 pour le SDA, P0.7 pour « Botonsin », P1.5 pour la terminal Rx et P2.6 pour la terminal Tx de la communication UART.

	Name	Port	Pin	Lock
<input type="checkbox"/>	\I2C:scl\	P1[0]	58	<input type="checkbox"/>
<input type="checkbox"/>	\I2C:sda\	P1[1]	59	<input type="checkbox"/>
<input type="checkbox"/>	Botonsin	P0[7]	46	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Rx_1	P1[5]	63	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Tx_1	P2[6]	8	<input checked="" type="checkbox"/>

6.1.2.3 Partie code C

L'accéléromètre c'est un composant qui a besoin de beaucoup de configurations. Lorsqu'on l'utilise avec un Arduino ou un Microcontrôleur il y a déjà bibliothèques qui sont conçu avec tous les paramètres nécessaires de l'accéléromètres afin de l'implémenter dans les projets facilement. Pourtant ce ne pas le cas avec un PSoC, donc il y a été nécessaire de créer des fichiers supplémentaires afin de pouvoir manipuler ce composant.

D'abord on a créé un fichier « Header » appelé « I2C.h » dans lequel on a défini tous les registres à utiliser de l'accéléromètre avec ses adresses d'accord à la Data Sheet (DS) du composant.

```
11 |L*/
12 | #include <project.h>
13 | #define ACCEL_ADDR (0x53) //0x1D if ALT ADDRESS is high or 0x53 if ALT ADDRESS is GND (page 18 DS)
14 | #define DEVID (0x00) //Device ID (Page 23 DS)
15 | #define POWER_CTL (0x2D) //Power-Saving features Control (Page 23 DS)
16 | #define DATA_FORMAT (0x31) //Control of Data Format
17 | #define BW_RATE (0x2C) // Data Rate and Power mode control
18 |
19 | #define DATA0 (0x32) //LSB X-Axis
20 | #define DATA1 (0x33) //MSB X-Axis
21 | #define DATA0 (0x34) //LSB Y-Axis
22 | #define DATA1 (0x35) //MSB Y-Axis
23 | #define DATA0 (0x36) //LSB Z-Axis
24 | #define DATA1 (0x37) //MSB Z-Axis
25 |
26 |
27 | uint8 accel_ReadReg(uint8 Reg);
28 | void accel_WriteReg(uint8 Reg, uint8 value);
29 |
```

Ensuite on a créé un fichier « Source » appelé « I2C.c ». Dans ce fichier on a créé une fonction (accel_ReadReg) afin de lire les données dans l'accéléromètre pour récupérer l'information qu'on a besoin, c'est-à-dire son accélération dans les 3 axes et une fonction (accel_WriteReg) qui sert à écrire les configurations qu'on désire dans les registres afin d'avoir la bonne configuration dans le composant. La fonction de lecture a un seul paramètre qui est le registre à lire (Reg) et la fonction d'écriture a ce même paramètre et un plus une autre qui est la valeur (Value) à écrire dans le registre.



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

```
14 /* To read we must:
15     1) Write te register address we want to read from to the slave device (accelerometer)
16     2)Use MasterReadBuf to store what is contained in the register into the read buffer
17 */
18 #include <i2c.h>
19
20 uint8 accel_ReadReg(uint8 Reg){
21     uint8 Write_Buf[1]={0}; //Buffer that will contain the register that will be read from
22     Write_Buf[0]=Reg;
23
24     uint8 Read_Buf[1]={0}; //Buffer that will contain the value read from the register
25
26     I2C_I2CMasterWriteBuf(ACCEL_ADDR, (uint8 *)Write_Buf,1,I2C_I2C_MODE_NO_STOP); //Instruction to write
27     while((I2C_I2CMasterStatus() & I2C_I2C_MSTAT_WR_CMPLT)==0){}
28
29     I2C_I2CMasterReadBuf(ACCEL_ADDR, (uint8 *)Read_Buf,1,I2C_I2C_MODE_REPEAT_START); //Instruction to Read
30     while((I2C_I2CMasterStatus() & I2C_I2C_MSTAT_RD_CMPLT)==0){}
31
32     return Read_Buf[0];
33 }
34
35 /* To Write we must write 2 values:
36     1)The register address we want to write to
37     2)The value we want to write
38 */
39 void accel_WriteReg(uint8 Reg, uint8 value){
40     uint8 Write_buf[2]={0};
41     Write_buf[0]=Reg;
42     Write_buf[1]=value;
43     I2C_I2CMasterWriteBuf(ACCEL_ADDR, (uint8 *)Write_buf, 2, I2C_I2C_MODE_COMPLETE_XFER);
44     while((I2C_I2CMasterStatus() & I2C_I2C_MSTAT_WR_CMPLT)==0){}
45     return;
46 }
47 /* [] END OF FILE */
```

Dans le fichier « main.c » on a écrit le code principal du programme. D'abord dans les directives on doit inclure les fichiers « I2C.h » et « I2C.c » qu'on a écrit. Ensuite dans le code, on a activé deux résistances afin d'avoir un bon fonctionnement avec l'accéléromètre. Les résistances permettent de changer le niveau logique des signaux afin d'avoir l'état haut et bas. Après on commence la fonction des communications I2C et UART. Puis on a déclaré les variables qu'on a utilisé dans le programme.

En utilisant la fonction d'écriture de l'accéléromètre on a configuré les paramètres du composant afin d'avoir le fonctionnement désiré, on a stipulé un mode de mesure avec l'octet 0x08 sur le registre POWER_CTL ; une mode de résolution complet, un format justifié à droite LSB et une portée de +-16g avec l'octet 0x0B sur le registre DATA_FORMAT ; une fréquence de sortie de données de 100 Hz et un largeur de bande de 50Hz avec l'octet 0x0A sur le registre BW_RATE.



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

```
12 #include "project.h"
13 #include <i2c.h>
14 #include <i2c.c>
15
16 int main(void)
17 {
18
19     /* Pull Up Resistors */
20     I2C_scl_SetDriveMode(I2C_scl_DM_RES_UP);
21     I2C_sda_SetDriveMode(I2C_sda_DM_RES_UP);
22
23     I2C_Start();
24     UARI_Start();
25
26     CyGlobalIntEnable; /* Enable global interrupts. */
27
28     uint8 X0, X1, Y0, Y1, Z0, Z1; // Stocker the data.
29     char X,Y,Z,instruction;
30     int move,sent;
31
32
33     /* Register's Configuration */
34     accel_WriteReg(POWER_CTL, 0x08); //0x08 Mode Measurement Page 25 DS.
35     accel_WriteReg(DATA_FORMAT, 0x0B); //0x03 = Full Resolution mode, Right-justified LSB, Range bits +-16g.
36     accel_WriteReg(BW_RATE, 0x0A); //0x0A = Output Data Rate 100Hz, Bandwith 50Hz.
37
```

En utilisant la fonction de lecture de l'accéléromètre on a récupéré les valeurs des axes X, Y et Z afin de les stocker dans les variables X0, X1, Y0, Y1, Z0 et Z1 pour les utiliser dans le programme.

```
38     for (;;)
39     {
40         /*      GETING DATA FROM ACCELEROMETER      */
41
42         /* Read axis values */
43         X0=accel_ReadReg(DATA_X0);
44         X1=accel_ReadReg(DATA_X1);
45
46         Y0=accel_ReadReg(DATA_Y0);
47         Y1=accel_ReadReg(DATA_Y1);
48
49         Z0=accel_ReadReg(DATA_Z0);
50         Z1=accel_ReadReg(DATA_Z1);
51
```

Dans la partie suivante du programme on fait la comparaison des données de l'accélérateur et des profils de mouvement connus afin de repérer la commande réalisée par l'utilisateur (UP, DOWN, RIGHT, LEFT).

On commence par lire l'état du bouton pour savoir si une commande est demandée, si le bouton n'est pas appuyé on ne fait aucune action. Lorsque le bouton est appuyé on commence à comparer la position du système pour vérifier qu'il soit bien sur la position neutre, si ce le cas on compare les mouvements afin de bien définir la commande demandée par l'utilisateur. Finalement l'utilisateur devra lâcher le bouton afin que la commande soit envoyée.



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

```
77 | /*      SET THE RIGHT INSTRUCTION TO SEND      */
78 |
79 | if(Botonsin_Read())
80 | {
81 |     move=0;
82 |     sent=0;
83 | }
84 |
85 | if(!Botonsin_Read()) //Button to notify that an action is requested
86 | {
87 |     if(((X0<50) || (X0>240)) && ((Y0<100) || (Y0>170))) //Neutral point
88 |     {
89 |         CyDelay(100);
90 |
91 |         // Up Volume Conditions
92 |         if((X1<5) && (X0>50) && ((Y0<100) || (Y0>170)))
93 |         {
94 |             CyDelay(100);
95 |             if((X1<5) && (X0>100) && ((Y0<100) || (Y0>170)))
96 |             {
97 |                 CyDelay(100);
98 |                 if((X1<5) && (X0>150) && ((Y0<100) || (Y0>170)))
99 |                 {
100 |                     move=1; //The move is defined as UP Volume
101 |                 }
102 |             }
103 |         }
104 |     }
```



GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

```
105 // Down Volume Conditions
106 if((X1>250)&&(X0<200)&&((Y0<100)|| (Y0>170)))
107 {
108     CyDelay(100);
109     if((X1>250)&&(X0<150)&&((Y0<100)|| (Y0>170)))
110     {
111         CyDelay(100);
112         if((X1>250)&&(X0<100)&&((Y0<100)|| (Y0>170)))
113         {
114             move=2; //The move is defined as Turn Down the Volume
115         }
116     }
117 }
118
119 // Next Station Conditions
120 if(((X0<70)|| (X0>200))&&(Y0>50)&&(Y1<5))
121 {
122     CyDelay(100);
123     if(((X0<70)|| (X0>200))&&(Y0>100)&&(Y1<5))
124     {
125         CyDelay(100);
126         if(((X0<70)|| (X0>200))&&(Y0>150)&&(Y1<5))
127         {
128             move=3; //The move is defined as Next Station
129         }
130     }
131 }
132
133 // Previous Station Conditions
134 if(((X0<70)|| (X0>200))&&(Y0<200)&&(Y1>250))
135 {
136     CyDelay(100);
137     if(((X0<70)|| (X0>200))&&(Y0<150)&&(Y1>250))
138     {
139         CyDelay(100);
140         if(((X0<70)|| (X0>200))&&(Y0<100)&&(Y1>250))
141         {
142             move=4; //The move is defined as Previous Station
143         }
144     }
145 }
146
147 }
```

Dans la dernière partie du programme, on envoie une lettre sur le réseau UART qui arrivera au module Bluetooth afin de l'envoyer à la partie réception du projet. La lettre à envoyer correspond au mouvement réalisé, c'est-à-dire pour augmenter le volume on a défini la lettre « U » de « *UP VOLUME* », pour diminuer le volume de l'autoradio on utilise la lettre « D » de « *DOWN VOLUME* », afin de changer la station de la radio on utilise la lettre « N » de « *NEXT STATION* » pour la station suivante et la lettre « P » de « *PREVIOUS STATION* » pour changer à la station précédente.



6.1.3 Configuration du module Bluetooth

En utilisant le logiciel TeraTerm on pourra établir les paramètres du module Bluetooth afin d'avoir une connexion automatique au moment de l'alimentation des modules Bluetooth.

Les paramètres seront configurés en utilisant des commandes AT pour avoir une vitesse 115,200 kbps, 8 bits de data, aucun bit de parité et un bit de stop pour la communication UART du module.

Pour le projet on souhaite que les modules soient connectés entre eux dès qu'ils soient alimenté pour qu'ils puissent communiquer. Pour cela on veut se connecter sans la demande d'un mot de passe et alors on doit configurer l'authentification à 0 et garder l'adresse du module avec lequel sera connecté dans le paramètre « *Remote Adresse* ». Aussi on doit configurer le module en « *Pairing Mode* » qui sert à que le module se connecte de façon automatique avec le module stocké dans le « *Remote Adresse* ».

```
COM5 - Tera Term VT
File Edit Setup Control Window Help
Ver 6.15 04/26/2013
(c) Roving Networks
***Settings***
BTA=0006666BB640
BTName=RNBT-B640
Baudr(SW4)=115K
Mode =Pair
Authen=0
PinCod=1234
Bonded=0
Rem=0006666C665B
***ADVANCED Settings***
SrvName= SPP
SrvClass=0000
DevClass=1F00
InqWindw=0100
PagWindw=0100
CfgTimer=255
StatuStr=NULL
HidFlags=200
DTRtimer=8
KeySwapr=0
***OTHER Settings***
Profile= SPP
CfgChar= $
SniffEna=0
LowPower=0
TX Power=0
IOPorts= 0
IOValues=0
Sleptmr=0
DebugMod=0
RoleSwch=0
```

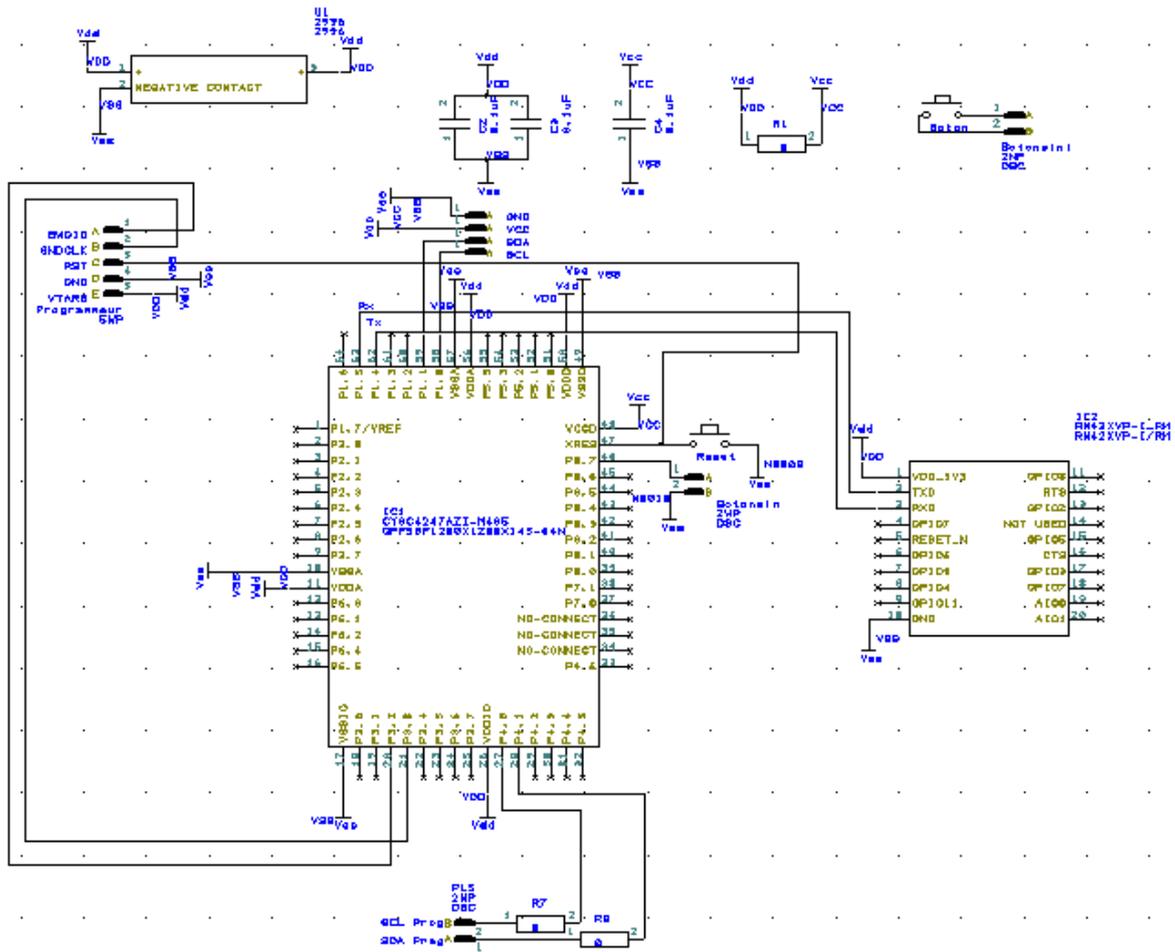


6.2 Conception Carte Autonome

6.2.1 Liste de composants

Quantité	Composant	Reference	Quantité	Composant	Reference
1	PSoC 4	Cy8C4247AZI-M485	3	Condensateurs	0.1uF
1	Module Bluetooth	RN42XVP	3	Résistances	0 ohms
1	Pile	3.3v	1	Support Pile	2696
2	Boutons	SW_FSM2JH	4	Connecteur	1WP
3	Connecteur	2WP	1	Connecteur	5WP

6.2.2 Schéma électrique

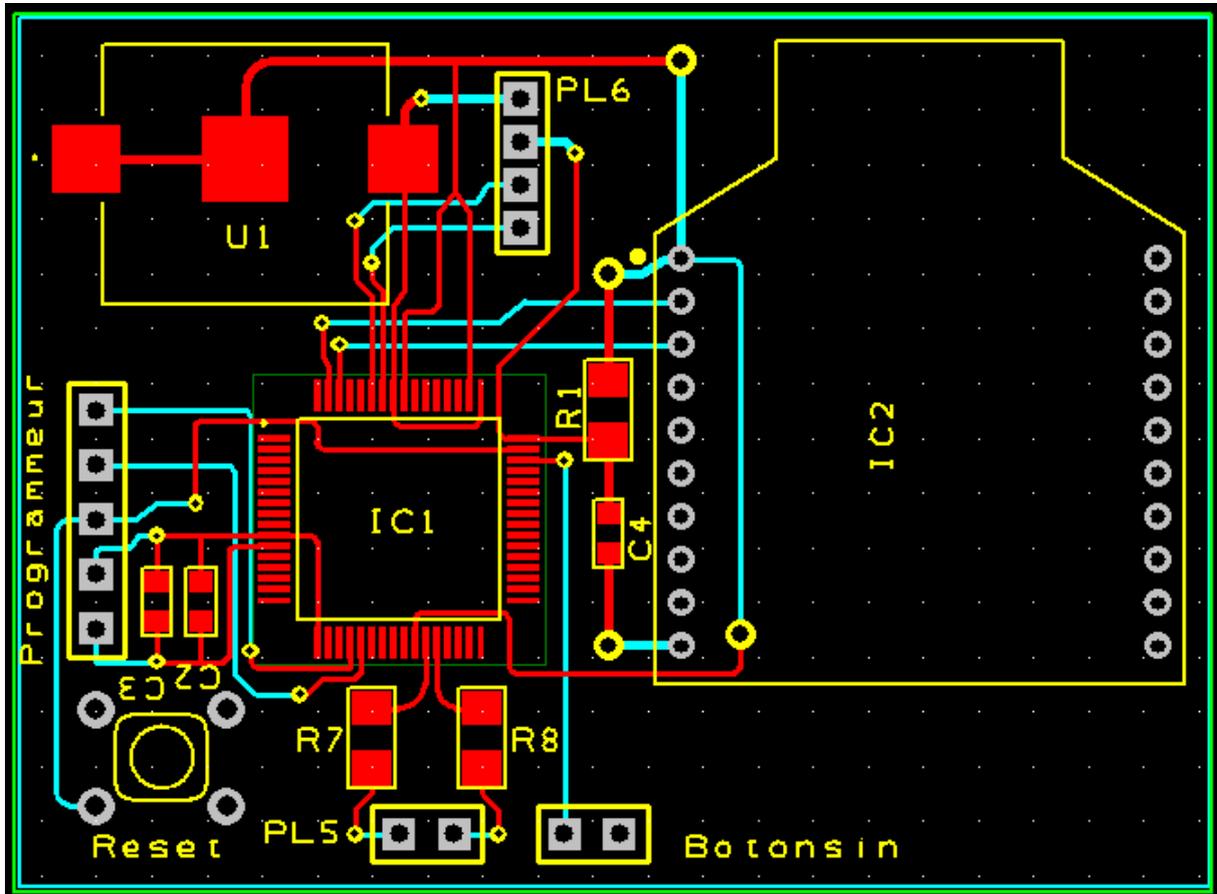




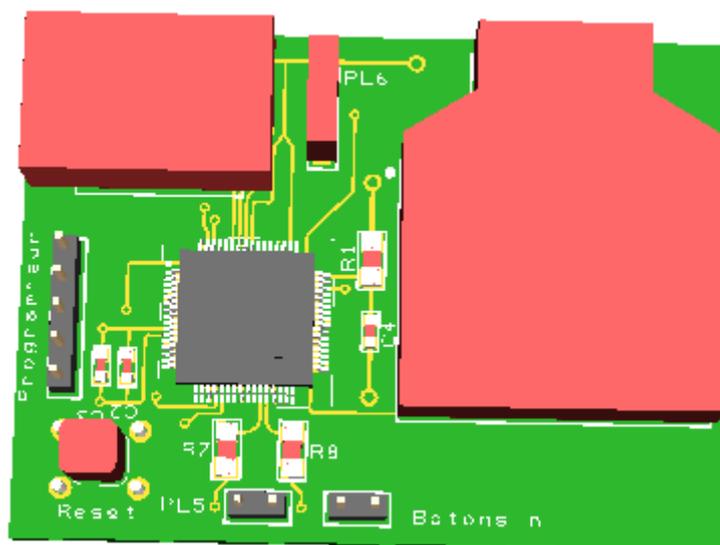
GEII

Département Génie Électrique
& Informatique Industrielle
IUT Belfort-Montbéliard

6.2.3 Design PCB

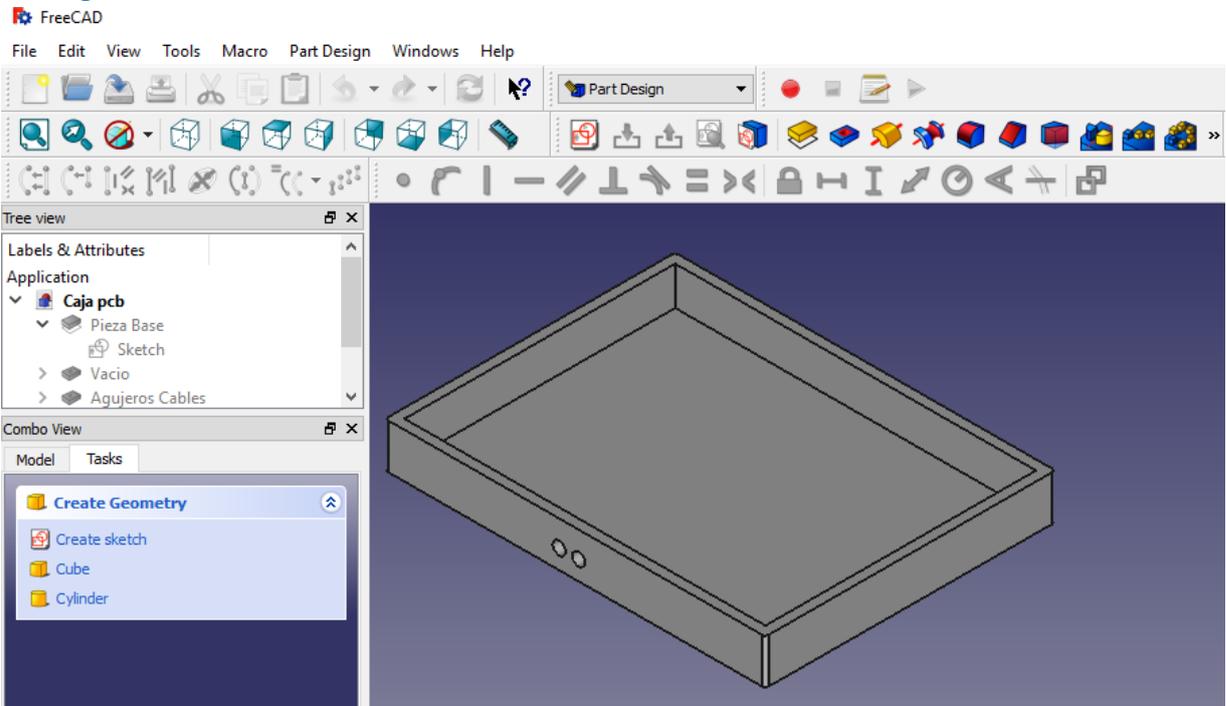


6.2.4 Visualisation 3D





6.3 Design Boîtier



6.4 Diagramme de Gantt

