

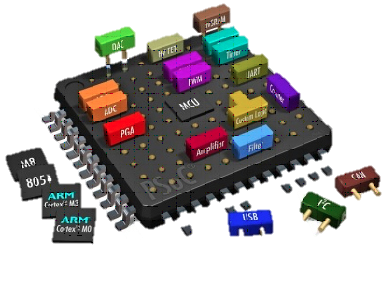
Formation STI2D SIN :

« Chaîne d'information et Microcontrôleur PSOC »

2ème Partie « PSoC 4 et le kit PIONNER »







**Objectifs de la formation :**

* Découvrir les microcontrôleurs PSoC 4,
* S'initier à leur programmation avec le logiciel « PSoC Creator » et le kit PIONNER,
* Mettre en œuvre le PSoC 4 avec différents périphériques dans la réalisation de projets.

**Compétences abordées : « SIN2 – Valider des solutions techniques »**

* CO2.1. Rechercher et choisir une solution logicielle ou matérielle au regard de la définition d'un système
* CO2.2. Etablir pour une fonction précédemment identifiée, un modèle de comportement à partir de mesures faites sur le système
* CO2.3. Traduire sous forme graphique l’architecture de la chaîne d’information identifiée pour un système et définir les paramètres d’utilisation du simulateur
* CO2.4. Identifier les variables simulées et mesurées sur un système pour valider le choix d’une solution.

**STI2D**

**Système d’information et numérique**

PADIOLLEAU Jean-Luc

14 mars 2014

[http://sti.ac-orleans-tours.fr](http://sti.ac-orleans-tours.fr/)

Contenu

1 Présentation du microcontrôleur « PSOC 4 » 5

1.1 Schéma fonctionnel interne 5

1.2 Caractéristiques de la CPU 6

1.3 Les fonctions intégrées 6

2 Kits d’évaluation « PIONNER PSOC 4 » 7

2.1 Extensibilité 7

2.2 Basse consommation 7

2.3 Débogage 7

2.4 Contenu du kit : 7

2.5 Schéma fonctionnel 8

2.6 Implantation 8

2.7 Brochage des connecteurs 9

3 Applications 10

3.1 Projet de base : commande de LED RGB 10

3.1.1 Câblage physique des éléments utilisés 10

3.1.2 Schéma sous PSoC Creator 10

3.1.3 Ordinogramme 10

3.1.4 Travail à effectuer 11

3.2 Variation de couleur de la LED RGB avec PWM. 11

3.2.1 Modulation de largeur d’impulsion 11

3.2.2 Schéma sous PSoc Creator 12

3.2.3 Ordinogramme 13

3.2.4 Travail à effectuer 14

3.3 Gestion de l’alimentation : « Mode hibernation » 15

3.3.1 Instruction de gestion des mode de consommation d’énergie (power mode) 15

3.3.2 Les interruptions 16

3.3.3 Description de projet 16

3.3.4 Travail à effectuer 17

3.4 Touche sensitive « Cap Sense » 18

3.4.1 Principe de fonctionnement (d’après Laurent PROUST – Académie de Poitiers) 18

3.4.2 Mise en œuvre d’un CapSense 19

3.4.3 Programme 20

3.4.4 Travail à effectuer 21

3.4.5 Brochage du Shield 23

3.4.6 Bibliothèque «glcd » 24

3.4.7 Mise en œuvre du Joystick 24

3.4.8 Application : « Afficher des images monochromes » 27

3.5 Gestion d’un shield « Bluetooth » 29

3.5.1 Présentation du Shield Seeeduino « 12A11 » 29

3.5.2 Mise en œuvre 30

4 Liens utiles 33

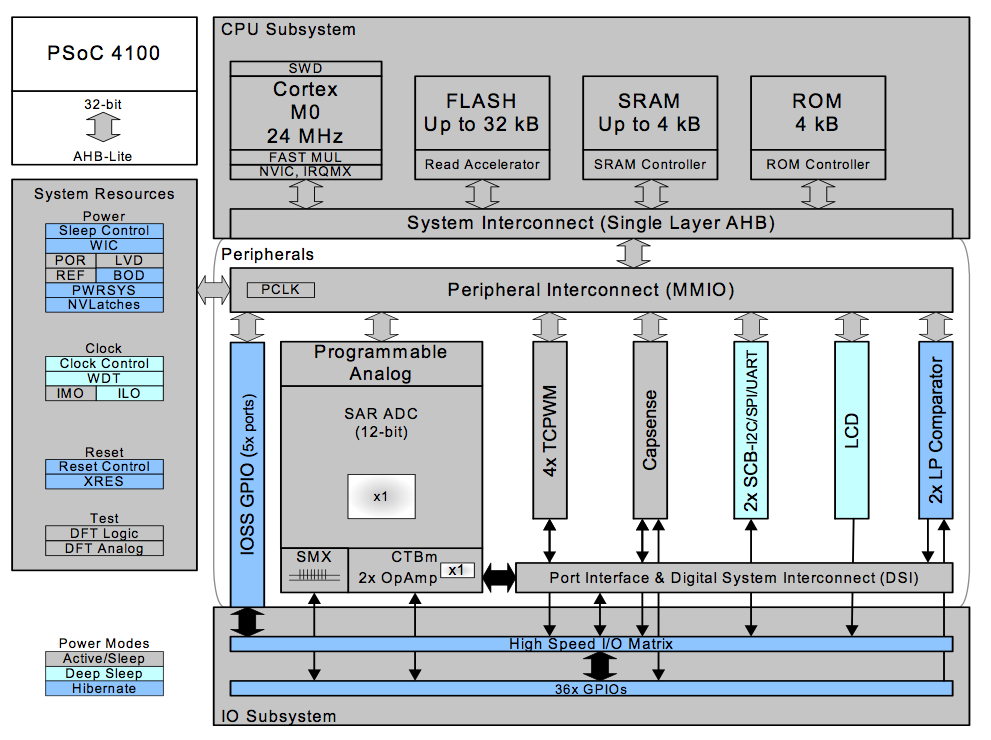
# Présentation du microcontrôleur « PSOC 4 »

La famille de microcontrôleurs PSOC 4, (programmable system on chip), est la dernière née de chez **Cypress**. Cette nouvelle génération se veut peu gourmande en énergie et adaptée aux systèmes embarqués.

Le cœur de processeur est un ARM® Cortex™-M0 32 bits.

Il existe deux sous-familles de PSoC 4 : CY8C4100 and CY8C4200.

## Schéma fonctionnel interne



## Caractéristiques de la CPU

|  |  |  |
| --- | --- | --- |
| **Vitesse** | | 24 / 48 MHz |
| **Taille du bus de données** | | 32 bits |
| **Mémoire programme** | | Flash de 16 ko à 32 ko. |
| **Mémoire de donnée** | | SRAM de 4 ko |
| **Alimentation** | | 1,71 V à 5,5 V. |
| **Consommation**  **(typique pour 5V)** | **Actif** | 6,7 mA pour une fréquence de 24 MHz |
| **Sommeil** | 1,7 mA |
| **Sommeil profond** | 15 µA |
| **Hibernation** | 150 nA |
| **Arrêt** | 20 nA |
| **Entrées / sorties** | | 32 |

## Les fonctions intégrées

La particularité des PSoC est qu’ils intègrent, par rapport à des microcontrôleurs classiques, des fonctions analogiques et numériques câblés. Voici celles dont est dotée la famille PSoC 4 :

|  |  |  |
| --- | --- | --- |
| **Fonction** | **Nombre** | **Caractéristiques** |
| Convertisseur analogique / numérique | 1 | 12 bits à approximation successive – 806 ksps |
| Convertisseur numérique / analogique | 2 | 7 et 8 bits à sortie courant (adaptés aux Cap Senses) |
| Amplificateur opérationnel | 2 | - |
| Comparateur | 2 | - |
| Timer / Compteur / PWM | 4 | 16 bits |
| Communication série | 2 | I2C, SPI et UART |
| Gestion LCD | 1 | Numérique et alphanumérique |
| Touche sensitive | 31 | Détection capacitive |

Le PSoC 4 est un modèle économique, il ne dispose pas d’autant de fonctions que les autres familles. Son prix avoisine les 2€ à l’unité.

# Kits d’évaluation « PIONNER PSOC 4 »

Le kit Pioneer PSoC 4 « CYC8CKIT-042 » est une carte de développement simple à utiliser et peu coûteuse doté du PSoC 4200.

## Extensibilité

Le kit Pioneer **PSoC 4** comprend les connecteurs compatibles pour les shields **Arduino ™**  et **Digilent Pmod ® ™**, permettant de relier un nombre important de cartes d'extension.

La carte dispose également d'un curseur CapSense ®, d’une LED RGB, d’un bouton poussoir utilisateur.

## ****Basse consommation****

L’architecture du PSoC 4 permet un mode de mise en veille prolongée ne consommant que 150 nA et un mode d'arrêt de 20 nA.

En mode actif, il offre des fonctionnalités analogiques entièrement fonctionnelles pour des tensions d’alimentation allant de 1,71 V à 5,5V.

Pour l’utilisation courante l’alimentation s’effectue par l’USB.

## ****Débogage****

Un dispositif à base de **PSoC 5LP** embarqué sert de programmeur et de débogueur.

L’interface entre le kit et un PC s’effectue via un connecteur USB.

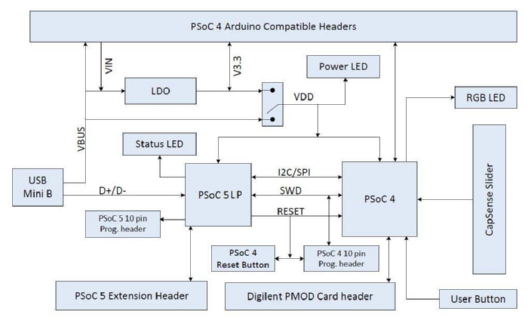
Avec le logiciel PSoC Creator ™ il est possible de d’exécuter un programme avec des points d’arrêt ou en mode pas à pas en utilisant le débogueur intégré.



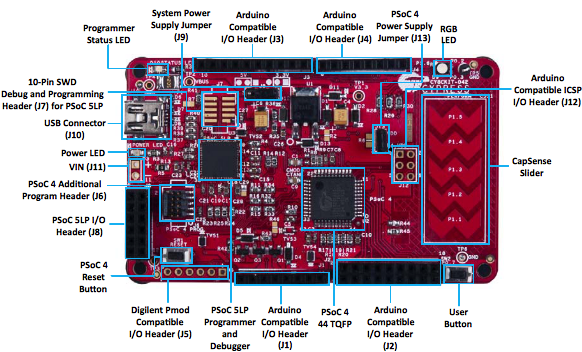
## Contenu du kit :

* Le kit Pioneer PSoC 4 ;
* Un guide de démarrage rapide ;
* Un cordon USB A vers mini-B ;
* 6 Fils cavalier.

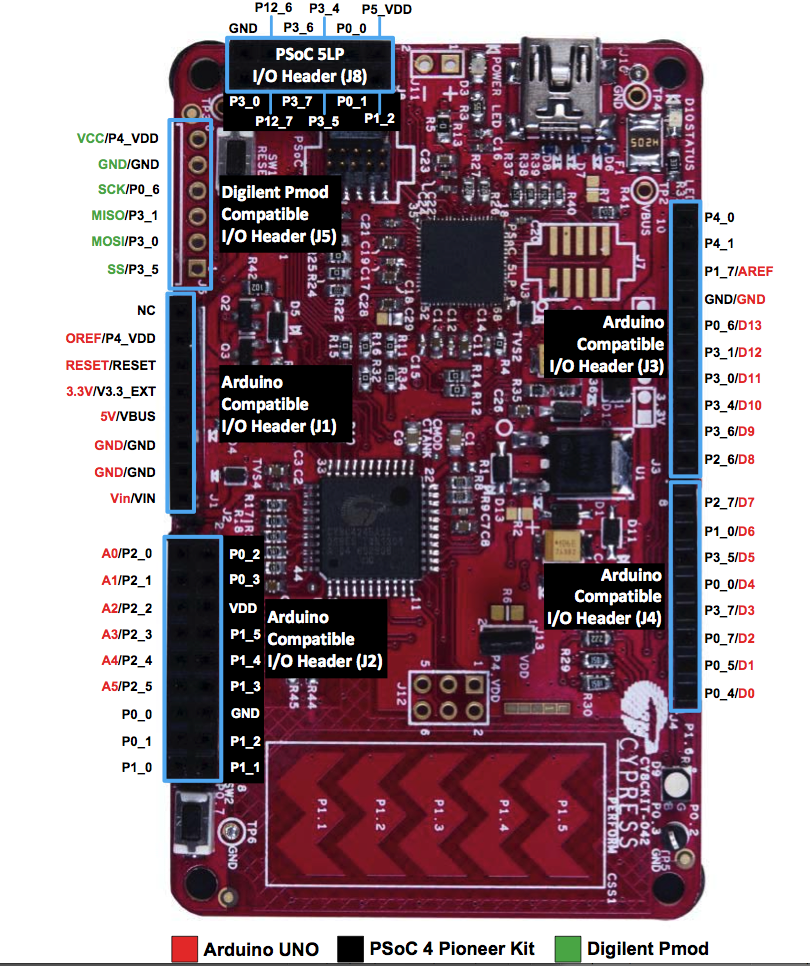
## Schéma fonctionnel



## Implantation



## Brochage des connecteurs



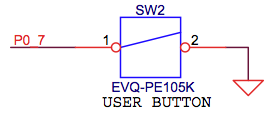
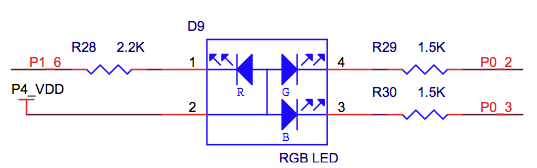
# Applications

## Projet de base : commande de LED RGB

Le but de cette première application est d’obtenir, avec la LED RGB D9, 7 couleurs différentes (plus un état où la LED est éteinte).

Le passage d’une couleur à la suivante se fait par appui sur le poussoir SW2.

### Câblage physique des éléments utilisés



Entier non signé codé sur 8 bits

DEBUT

RGB  C

SW2

Entier C  0

FIN

1

C  C+1

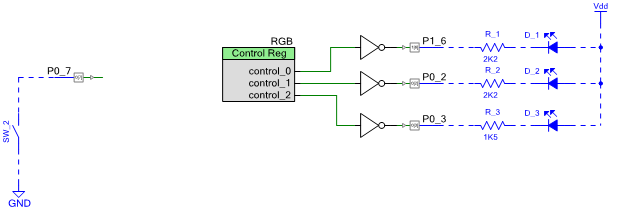
C<7

C  0

SW2

Tempo 100ms

### Schéma sous PSoC Creator



### Ordinogramme

### Travail à effectuer

1. Lancer l’environnement de développement « PSoC Creator 3 ».
2. Créer un nouveau projet à base de PSoC 4.
3. Saisir le schéma précédent.
4. Faire l’affectation des broches d’entrées /sorties.
5. Saisir le programme en langage C en s’inspirant de l’ordinogramme.
6. Compiler puis transférer le programme dans le kit.
7. Vérifier le fonctionnement.

## Variation de couleur de la LED RGB avec PWM.

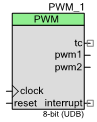
Le but de cette seconde activité est faire varier de façon aléatoire la couleur de la LED RGB en modulant par PWM ses 3 composantes rouge, verte et bleue.

### Modulation de largeur d’impulsion

Afin de faire varier l’intensité lumineuse de la chaque couleur, nous utiliserons une modulation de largeur d’impulsion au niveau de la commande.

Il existe sous « PSoC Creator » un composant « PWM » facile à utiliser en mode 8 ou 16 bits. Il est doté :

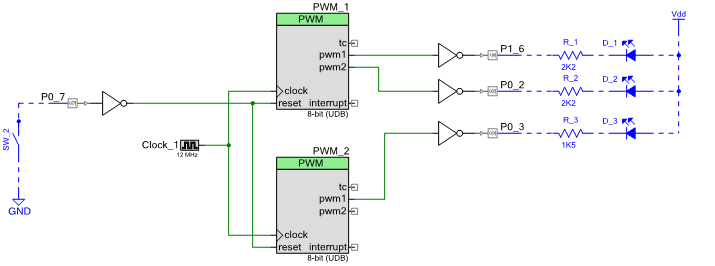
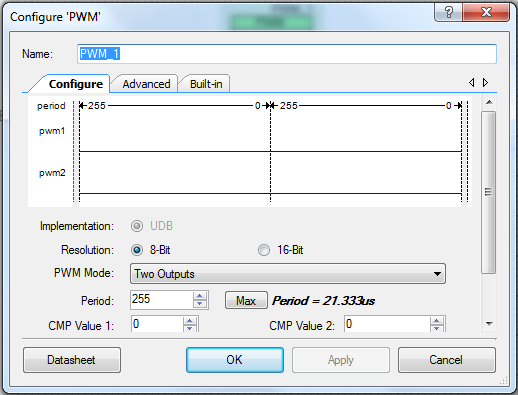
* de deux sorties modulées ;
* d’une sortie indiquant que le compteur est à 0 ;
* d’une sortie d’interruption ;
* d’une entrée d’horloge ;
* d’une entrée de remise à zéro.



D’un point de vue logiciel, nous utiliserons seulement 3 instructions :

1. « PWM\_Init() » pour initialiser le timer avec les données saisies dans la boîte de dialogue ;
2. « PWM\_WriteCompare1(int …) » pour définir la valeur du rapport cyclique de la sortie PWM1 ;
3. « PWM\_WriteCompare2(int …) » pour définir la valeur du rapport cyclique de la sortie PWM2.

### Schéma sous PSoc Creator



### Ordinogramme

Entiers non signés codés sur 8 bits

DEBUT

PWM-R  R

Entier R  0

Entier G  0

Entier B  0

Init. PWM

FIN

1

Aléatoire(R)

Aléatoire(G)

Aléatoire(B)

PWM-G G

PWM-B B

Tempo(500ms)

Information :

La fonction pour générer un nombre aléatoire compris en 0 et 255 est « rand()%256 »

### Travail à effectuer

1. Lancer l’environnement de développement « PSoC Creator 3 ».
2. l’environnement de développement « PSoC Creator 3 ».
3. Créer un nouveau projet à base de PSoC 4.
4. Saisir le schéma précédent.
5. Configurer les 2 TIMERS comme indiqué précédemment.
6. Faire l’affectation des broches d’entrées /sorties.
7. Saisir le programme en langage C en s’inspirant de l’ordinogramme.
8. Compiler puis transférer le programme dans le kit.
9. Vérifier le fonctionnement puis mesurer les 3 signaux PWM à l’oscilloscope.

## Gestion de l’alimentation : « Mode hibernation »

Une des particularités du PSoC 4 est de consommer très peu d’énergie dans certains modes de fonctionnement. C’est la cas notamment du mode « hibernation » où la consommation est d’environ 150 nA.

En hibernation, certains éléments restent « vigilant » et d’autres sont désactivés. Nous allons voir comment rentrer et sortir de ce mode.

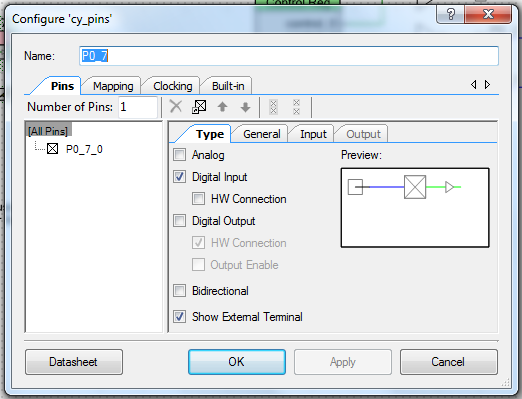
### Instruction de gestion des mode de consommation d’énergie (power mode)

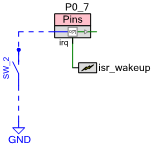
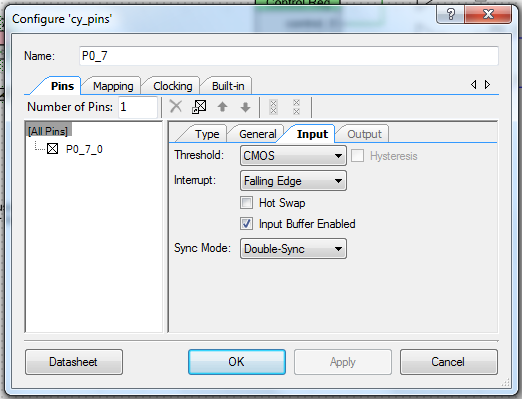
Le compilateur de « PSoC creator » est doté d’une bibliothèque « cyPm.h » dédiée aux différents modes de gestion de l’énergie. Les principales instructions sont :

|  |  |
| --- | --- |
| *Instruction* | *Rôle* |
| void CySysPmUnfreezeIo(void) ; | Dégèle les états des entrées / sorties |
| void CySysPmFreezeIo(void); | Gèle les états des entrées / sorties |
| uint32 CySysPmGetResetReason(void); | Détermine, après un redémarrage, le mode de réveil. Voici les différentes valeurs retournées indiquant le mode actif :  CY\_PM\_RESET\_REASON\_WAKEUP\_STOP pour Stop  CY\_PM\_RESET\_REASON\_WAKEUP\_HIB pour Hibernation ou Sommeil,  CY\_PM\_RESET\_REASON\_XRES pour reset. |
| void CySysPmSleep(void); | Active le mode sommeil |
| void CySysPmDeepSleep(void); | Active le mode sommeil profond |
| void CySysPmStop(void); | Active le mode arrêt |
| void CySysPmHibernate(void); | Active le mode hibernation |

Pour utiliser cette bibliothèque : #include <cyPm.h>

### Les interruptions

Dans « PSoC creator » la déclaration d’une interruption se fait grâce à l’éditeur de schéma. Une entrée ou tout autre source peut déclencher une interruption. Pour cela il suffit de la relier à un composant « isr ». Voici une interruption liée à la broche PO\_7 :



Dans le programme il suffit de définir les actions de l’interruption dans la fonction CY\_ISR avec pour paramètre le nom du composant d’interruption :

CY\_ISR(isr\_wakeup)

{

…

…

}

### Description de projet

Le projet a pour objectif de piloter la LED RGB en faisant défiler les 7 couleurs (plus le noir) toutes les secondes. A la fin du cycle, lorsque nous appuyons sur la touche « SW2 », le microcontrôleur rentre en mode hibernation. La LED RGB reste figée à la couleur rouge pour indiquer ce mode.

Un nouvel appui sur la touche « SW2 » fait sortir le microcontrôleur de ce mode. C’est l’interruption qui permet le réveil du PSoC.

Lors du redémarrage, une couleur de la LED RGB doit clignoter 8 fois à 5 Hz pour indiquer l’origine du réveil.

|  |  |
| --- | --- |
| *Couleur clignotante* | *Origine du réveil* |
| Rouge | Sortie du mode stop |
| Verte | Sortie du mode hibernation ou sommeil |
| Bleue | Remise à zéro |
| Blanche | Autre origine |

### Travail à effectuer

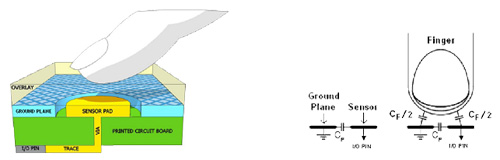
1. Dessiner l’ordinogramme ou écrire l’algorithme correspondant au cahier des charges de ce projet.
2. Lancer l’environnement de développement « PSoC Creator 3 ».
3. Créer un nouveau projet à base de PSoC 4.
4. Saisir le schéma correspondant.
5. Configurer la broche d’entrée correspondante afin d’activer le mode interruption sur front descendant.
6. Faire l’affectation des broches d’entrées /sorties.
7. Saisir le programme en langage C en s’inspirant de l’ordinogramme.
8. Compiler puis transférer le programme dans le kit.
9. Vérifier le fonctionnement.

## Touche sensitive « Cap Sense »

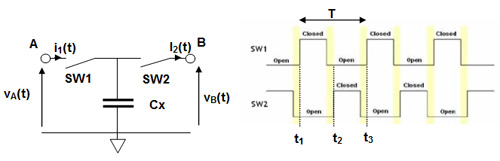
Cypress est le leader mondial dans les technologies de détection capacitive. Le « CapSense » va des simples boutons et curseurs à des solutions plus sophistiquées. Ces composants permettent de réduire le coût des interfaces hommes / machines.

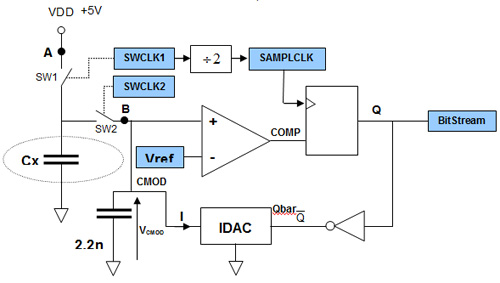
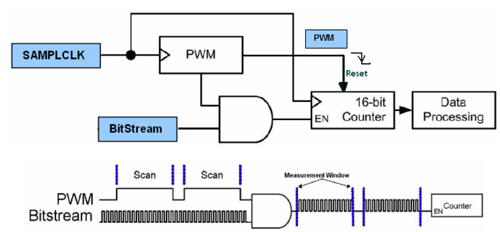
### ****Principe de fonctionnement (d’après Laurent PROUST – Académie de Poitiers)****

Lorsqu’un objet conducteur est placé à proximité des deux électrodes (Ground Plane et Sensor Pad), la capacité totale Cx (Cx = CP initialement) change et augmente pour devenir au maximum Cx = CP + CF. La technologie CapSense CSD (CSD pour CapSense® Sigma Delta) permet de détecter le changement de la capacité Cx.



L’électrode Sensor est connectée à l’une des broches du PsoC et **l’utilisation du composant Capsense CSD** intègre alors le condensateur Cx dans le schéma de principe suivant :



ormule de calcul indiquant la valeur de la résistance RX en fonction de la valeur de la capacité CX.La cellule ainsi constituée entre les points A et B est une Capacité commutée et se ramène à une résistance dont la valeur est :

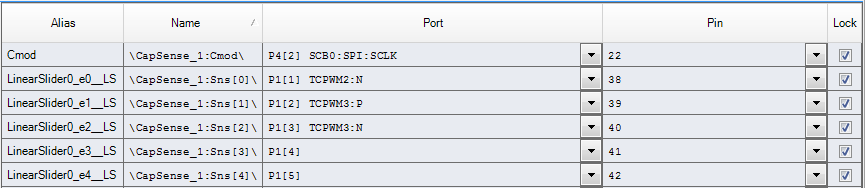
* DAC est un générateur de courant programmable (par défaut réglé à 255µA) et commandé par l’état de Qbar (NL 0).
* Le circuit de capacité commutée entre A et B, associé avec CMOD, revient à étudier une cellule RxCMOD classique. Le comparateur relève l’évolution de la tension aux bornes de CMOD et indique un dépassement de la tension de référence.
* L’évolution de VCMOD est rapide si Rx est petite, donc si le doigt est présent. Quand cette tension atteint le seuil VREF (1,024V), l’état logique est mémorisé (front montant de SAMPLCLK). Cela est répercuté sur la décharge de CMOD par IDAC (quasi-courant constant hormis l’influence de Rx...). Ce qui entraîne la mise à zéro de l’état de Q (après front montant de SAMPLCLK).
* Le flux de donnée série Bitstream possède donc une fréquence directement liée à la valeur de CX. Une autre structure (à suivre) va permettre l’exploitation de ce flux afin de produire un nombre binaire image de la fréquence. Dés lors, des détections sur les valeurs (fonction C de contrôle \_voir API du composant CAPSENSE sous PSOC CREATOR\_ ) peuvent permettre directement de savoir si la touche a été sollicitée ou non (différence de fréquences mémorisées).
* Chaque touche déclarée dans le module CAPSENSE devra bénéficier d’un temps d’analyse (SCAN) permettant d’identifier l’appui.

### Mise en œuvre d’un CapSense

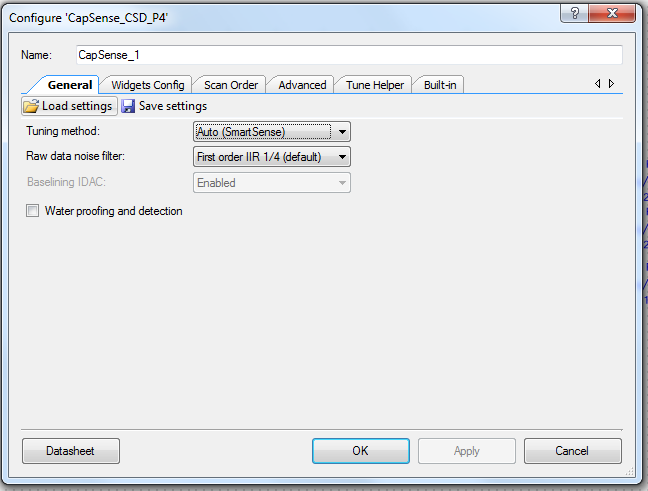
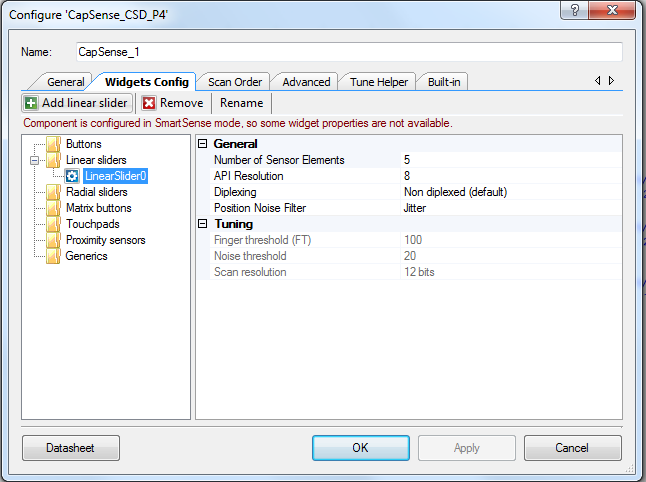
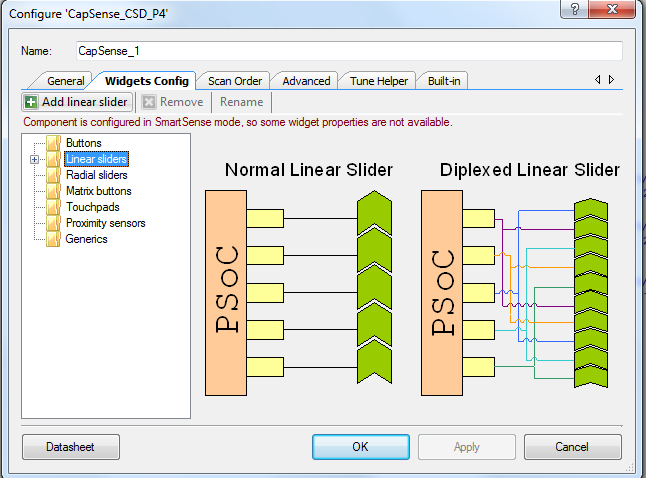
L’idée est ici avant tout de profiter de la simplicité de mise en œuvre offerte par l’interface de management du composant CAPSENSE dans PSOC CREATOR. Ce type de technologie pourra être intégrée dans des projets présents ou à venir.

* Le kit Pionner est équipé d’un « Slider Cap Sense » constitué de 5 capteurs.
* Nous souhaitons mettre en œuvre ce Slider pour modifier la couleur (8) de la LED RGB.
* Sous PSoC Creator, un composant est disponible pour effectuer le traitement des « CapSense ».

Voici sa configuration pour notre application :

Indique la valeur maximum du Slider

Instructions utilisées pour le Slider :

|  |  |
| --- | --- |
| ***Instruction*** | ***Rôle*** |
| void CapSense\_Start(void); | Initialise et lance l’interface pour les CapSenses |
| void CapSense\_InitializeAllBaselines(void); | Initialise toute les lignes de CapSense |
| void CapSense\_UpdateEnabledBaselines(void); | Met à jour les lignes de CapSense actives |
| void CapSense\_ScanEnabledWidgets(); | Scanne l’état de tous les éléments CapSenses actifs |
| uint16 CapSense\_GetCentroidPos(uint8); | Retourne la valeur de position du Slider.  Si rien n’est actif, la valeur retournée est 0xFFFF. Le paramètre indique le numéro de l’élément à lire (0 dans notre cas). |
| CapSense\_IsBusy(void); | Indique si l’ensemble des CapSenses peut être lus |

### Programme

#include <device.h>

uint16 curPos, oldPos;

int main()

{

/\* Enable global interrupts \*/

CyGlobalIntEnable;

CapSense\_1\_Start();

/\* Initialisation baselines \*/

CapSense\_1\_InitializeAllBaselines();

for(;;)

{

/\* Met ‡ jour les lignes de CapSenses \*/

CapSense\_1\_UpdateEnabledBaselines();

/\* Lance le scan de tous les CapSenses \*/

CapSense\_1\_ScanEnabledWidgets();

/\* Attente de la fin du scan \*/

while(CapSense\_1\_IsBusy() != 0);

/\* Affiche l'état du Slider \*/

curPos=CapSense\_1\_GetCentroidPos(0);//CapSense\_1\_LINEARSLIDER0\_\_LS

if (curPos<8) oldPos=curPos;

RGB\_Write(oldPos);

}

}

### Travail à effectuer

1. Lancer l’environnement de développement « PSoC Creator 3 ».
2. Créer un nouveau projet à base de PSoC 4.
3. Saisir le schéma correspondant.
4. Configurer le composant « CapSense » comme indiqué précédemment.
5. Faire l’affectation des broches d’entrées /sorties.
6. Saisir le programme en langage C.
7. Compiler puis transférer le programme dans le kit.
8. Vérifier le fonctionnement.

## Gestion d’un shield « LCD Graphique »

La particularité du kit que nous mettons en œuvre est de pouvoir s’interfacer avec tous les SHIELDS compatible ARDUINO.

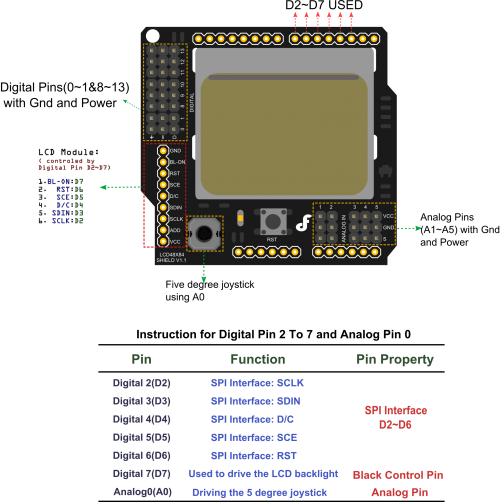
Nous allons utiliser le SHIELD « LCD4884 » qui comprend un écran LCD graphique monochrome de 84 colonnes par 48 lignes et un joystick à sortie analogique.

L’interface avec le microcontrôleur s’effectue par liaison série synchrone du type SPI.

Les données transmises sont transmises par octet ce qui correspond, pour l’affichage, à 8 pixels.

|  |  |  |  |
| --- | --- | --- | --- |
| *Signal* | *Rôle* | *Broche ARDUINO* | *Broche PSoC* |
| SCK | Horloge de synchronisation série | D2 | P0\_7 |
| SDIN | Signal de donnée série | D3 | P3\_7 |
| D/C | Signal de distinction entre une donnée (1) et commande (0) | D4 | P0\_0 |
| SCE | Signal de sélection du circuit de gestion du LCD (actif à 0) | D5 | P3\_5 |
| RST | Réinitialisation du circuit de gestion du LCD | D6 | P1\_0 |
| BL | Activation du rétro-éclairage (actif à 1) | D7 | P2\_7 |
| JOYSTICK | Signal analogique (5 valeurs de tensions) du joystick | A0 | P2\_0 |

### Brochage du Shield

****

### Bibliothèque «glcd »

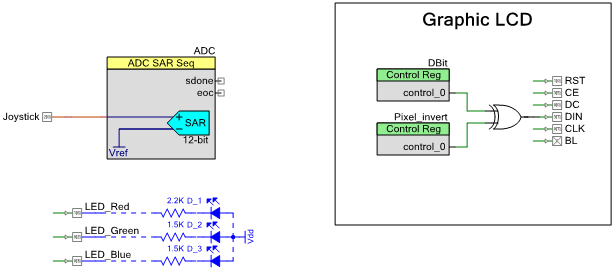
Une bibliothèque en langage C permet de piloter l’afficheur. Voici les différentes fonctions implémentées :

|  |  |
| --- | --- |
| void LcdInit(void); | Initialisation du LCD |
| void LcdGotoXY(uint8 x, uint8 y); | Positionnement du LCD en X,Y |
| void LcdBitmap(uint8 my\_array[]); | Affichage d’une image Bitmap |
| void LcdCharacter(char character); | Affichage d’un caractère |
| void LcdString(char \*characters); | Affichage d’une chaine de caractère |
| void LcdClear(void); | Effacement de l’écran |
| void LcdWrite(char data\_or\_comm, char dat); | Ecriture de donnée ou de commande |
| void LcdSendData(char val); | Ecriture vers l’afficheur |
| void LcdPrintBigNumber(uint8 x); | Affichage de chiffre grand format |

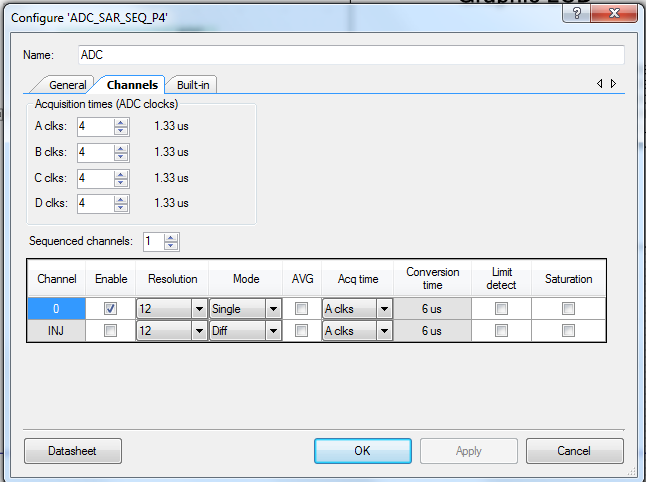
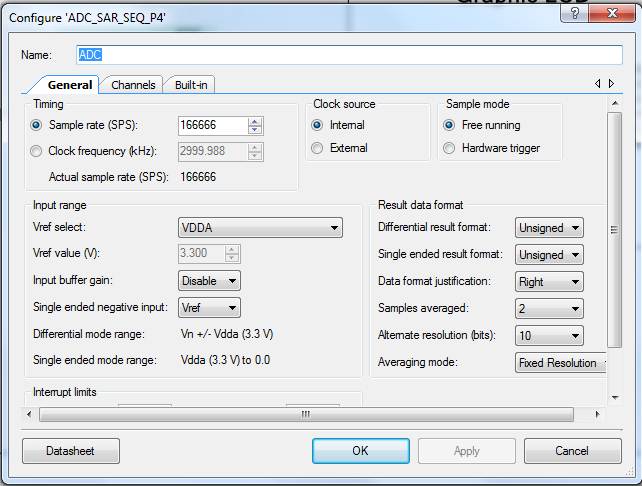
### Mise en œuvre du Joystick

On souhaite utiliser l’afficheur afin de visualiser la position du joystick à l’écran. Ce composant génère une tension différente selon la touche active. Afin de connaître les valeurs à prendre en compte, un premier projet doit permettre d’afficher les valeurs issue d’une conversion analogique / numérique de la tension provenant du joystick.

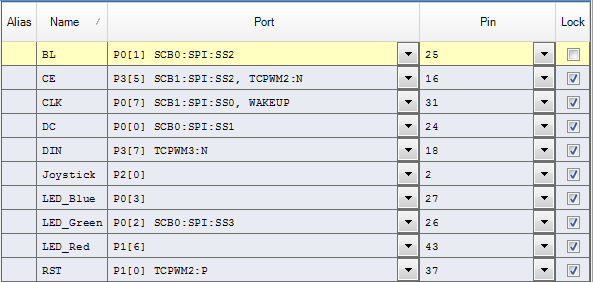
* **Schéma sous PSoC Creator**



* **Configuration du convertisseur A/N**

****

* **Affectation des broches**

****

* **Programme**

#include <device.h>

#include <stdio.h>

#include "Display.h"

void main()

{

/\* Initialise variables \*/

uint16 Joystick;

char Chaine[16];

CyGlobalIntEnable; /\* Uncomment this line to enable global interrupts. \*/

/\* Start components \*/

ADC\_Start();

ADC\_StartConvert();

LcdInit();

LcdClear();

LED\_Red\_Write(0);

CyDelay(1000);

LED\_Red\_Write(1);

LED\_Green\_Write(0);

CyDelay(1000);

LED\_Green\_Write(1);

LED\_Blue\_Write(0);

CyDelay(2000);

LcdString("Bonjour");

CyDelay(2000);

LcdClear();

for(;;)

{

LcdGotoXY(0,0);

LcdString(" ");

Joystick=ADC\_GetResult16(0);

sprintf(Chaine,"%d",Joystick);

LcdGotoXY(0,0);

LcdString(Chaine);

CyDelay(100);

}

}

1. Lancer l’environnement de développement « PSoC Creator 3 ».
2. Créer un nouveau projet à base de PSoC 4.
3. Saisir le schéma correspondant.

|  |  |
| --- | --- |
| Touche du Joystick | Valeur numérique |
| Haut |  |
| Bas |  |
| Gauche |  |
| Droite |  |
| Central |  |

1. Configurer le composant « ADC » comme indiqué précédemment.
2. Faire l’affectation des broches d’entrées /sorties.
3. Saisir le programme en langage C.
4. Compiler puis transférer le programme dans le kit.
5. Vérifier le fonctionnement puis relever les valeurs numériques de chaque touche du joystick.

### Application : « Afficher des images monochromes »

Dans cette application il s’agit d’utiliser le joystick pour faire défiler un ensemble de 4 images (ou plus) monochrome à l’écran.

Les images sont obtenues grâce à l’application « LCD assistant » qui convertit une image BMP en un fichier C.

Voici les 4 images, d’une résolution 84x48 pixels, à afficher :

|  |  |
| --- | --- |
| Touche du Joystick | Action |
| Haut | Défilement automatique des images toutes les secondes dans le sens croissant |
| Bas | Défilement automatique des images toutes les secondes dans le sens décroissant |
| Gauche | Défilement manuel des images dans le sens croissant |
| Droite | Défilement manuel des images dans le sens décroissant |
| Central | Arrêt du défilement automatique |

Les images sont converties en fichier de type « C » ayant l’extension « .h » et déclarées sous forme de tableaux de 504 octets.

Pour pouvoir les intégrer dans le programme, il faut rajouter « #include « NomImage.h » » puis intégrer le fichier dans le projet de PSoC Creator.

Pour accéder plus facilement aux images il est possible de toutes les stocker dans un tableau à 2 dimensions.

**Voici le fonctionnement attendu**:

1. Modifier le projet et le programme en langage C pour répondre à ce cahier des charges.
2. Compiler puis transférer le programme dans le kit.
3. Vérifier le fonctionnement.

## Gestion d’un shield « Bluetooth »

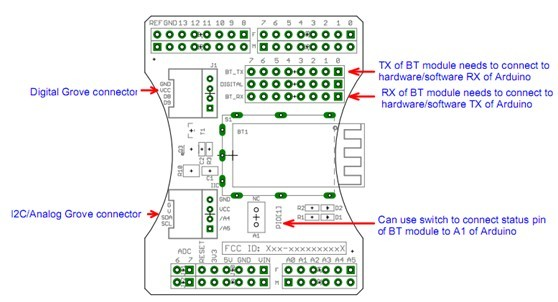
Le « Bluetooth » permet au kit PSoC de communiquer avec un ordinateur, un smartphone ou une tablette. Une multitude de Shield ayant cette fonctionnalité existe. Nous allons utiliser celui de chez « Seeeduino » ayant pour référence « 12A11 ».

### Présentation du Shield Seeeduino « 12A11 »

La communication entre le kit et le Shield se fait une liaison série asynchrone « Tx / Rx ».

La carte offre la possibilité de choisir les broches (entre D0 et D7) qui vont être utilisées pour le dialogue bidirectionnel. Par défaut la vitesse de communication est établie à 38400 bauds. Il y a possibilité de lire l’état du bluetooth en connectant un signal à la broche A1 par l’intermédiaire d’un switch.

En plus du module bluetooth, le Shield est équipé d’un connecteur d’extension, où sont présente D8 et D9, et un bus I2C.



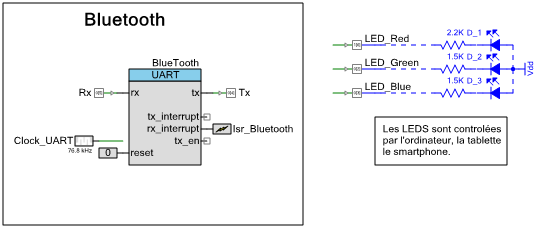
|  |
| --- |
| Tension d’entrée : 3.3V |
| Vitesse de communication : 9600, 19200, 38400, 57600, 115200, 230400, 460800 bauds |
| Compatible Seeeduino/Arduino |
| Distance de communication: 10m sans obstacle |
| UART interface (TTL) with programmable baud rate (SPP firmware installed) |
| Configuration par défaut 38400, Data bits: 8, Stop bit: 1, Parity: No parity |
| PINCODE par défaut ”0000” |
| Configuration par commandes AT |
| Antenne sur le circuit imprimé |

### Mise en œuvre

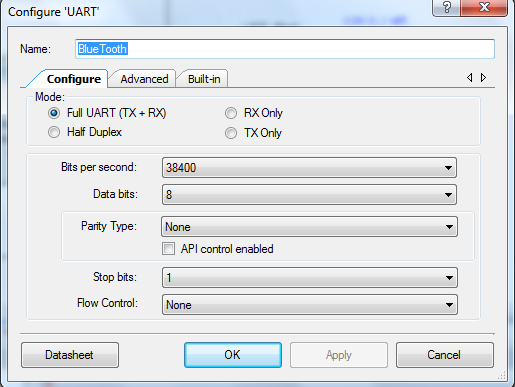
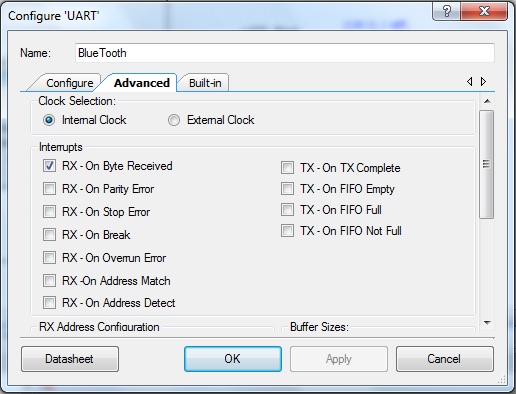
Le projet présenté permet de contrôler la LED RGB à partir d’une ordinateur, d’une tablette ou un smartphone.

Une fois l’appairage établi, les commandes peuvent être saisies dans un **terminal**. Voici le fonctionnement attendu :

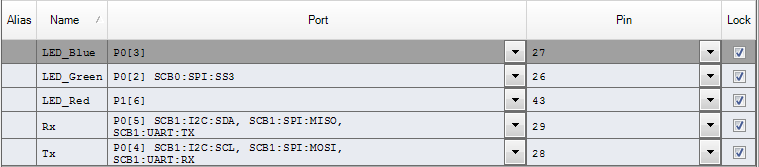
* + La transmission du caractère « R » à pour effet de commuter la LED rouge, en retour le PSoC transmet l’état de cette LED ;
  + La transmission du caractère « G » à pour effet de commuter la LED verte, en retour le PSoC transmet l’état de cette LED ;
  + La transmission du caractère « B » à pour effet de commuter la LED bleue, en retour le PSoC transmet l’état de cette LED
* **Schéma PSoC Creator**



* **Configuration de la liaison série**

* **Affectation des entrées / sorties**

****

* **Programme en langage « C »**

#include <device.h>

void InitBlueTooth(void)

{

BlueTooth\_Start();

BlueTooth\_PutString("");

BlueTooth\_PutString("\r\n+STWMOD=0\r\n"); // Bluetooth en mode esclave

BlueTooth\_PutString("\r\n+STNA=PSOC4\r\n"); // Nom du bluetooth = "PSOC4"

BlueTooth\_PutString("\r\n+STOAUT=1\r\n"); // Autorise l'apairage

BlueTooth\_PutString("\r\n+STAUTO=0\r\n"); // Pas d'autoconnexion

CyDelay(2000);

BlueTooth\_PutString("\r\n+INQ=1\r\n"); // Bluetooth joignable

CyDelay(2000);

}

void main()

{

/\* DÈclaration variables \*/

uint8 BT\_Rx;

CyGlobalIntEnable; /\* Uncomment this line to enable global interrupts. \*/

/\* Start components \*/

InitBlueTooth();

//Isr\_Bluetooth\_Start();

LED\_Red\_Write(0);

CyDelay(1000);

LED\_Red\_Write(1);

LED\_Green\_Write(0);

CyDelay(1000);

LED\_Green\_Write(1);

LED\_Blue\_Write(0);

CyDelay(1000);

LED\_Blue\_Write(1);

CyDelay(1000);

for(;;)

{

/\* Lecture caractËre en rÈception bluetooth\*/

BT\_Rx = BlueTooth\_GetChar();

if(BT\_Rx == 'R')

{

if (LED\_Red\_Read()==1)

{

LED\_Red\_Write(0);

BlueTooth\_PutString("LED rouge ON\n\r");

}

else

{

LED\_Red\_Write(1);

BlueTooth\_PutString("LED rouge OFF\n\r");

}

}

else if(BT\_Rx == 'G')

{

if (LED\_Green\_Read()==1)

{

LED\_Green\_Write(0);

BlueTooth\_PutString("LED verte ON\n\r");

}

else

{

LED\_Green\_Write(1);

BlueTooth\_PutString("LED verte OFF\n\r");

}

}

else if(BT\_Rx == 'B')

{

if (LED\_Blue\_Read()==1)

{

LED\_Blue\_Write(0);

BlueTooth\_PutString("LED bleue ON\n\r");

}

else

{

LED\_Blue\_Write(1);

BlueTooth\_PutString("LED bleue OFF\n\r");

}

}

}

}

# Liens utiles

|  |  |
| --- | --- |
| Contenu | Lien |
| Fabricant du PSoC « Cypress » | http://www.cypress.com |
| Fil de discussion Element 14 | http://www.element14.com/community/community/knode/dev\_platforms\_kits/element14\_dev\_kits/psoc4\_pioneer\_kit |
| Wiki du Shield « LCD4884 « | http://www.dfrobot.com/wiki/index.php?title=LCD4884\_Shield\_Fro\_Arduino\_(SKU:DFR0092) |
| Wiki du Shield « Bluetooth » | http://www.seeedstudio.com/wiki/Bluetooth\_Shield |