# Introduction au PSoCPSoC.jpg

Le PSoC (Programmable System on Chip) est un composant comportant un microcontrôleur (ARM® Cortex™-M3 de 32-bit), de la mémoire et des blocs numériques et analogiques configurables.

Malgré sa complexité, le logiciel « PSoC Creator » fourni avec rend son utilisation extrêmement simple :

* Choix des blocs configurables et de leurs liens sur le schéma TopDesign.cysch ;
* Choix des broches pour chaque port (bloc) du schéma dans le fichier \*.cydwr ;
* Documentations séparées des blocs disponibles par le schéma ou directement par la bibliothèque ;
* Programmation du microcontrôleur en C ;
* Fonctions d’utilisation des blocs préexistantes et documentées ;
* Compilation, implémentation et débogage intégrés.

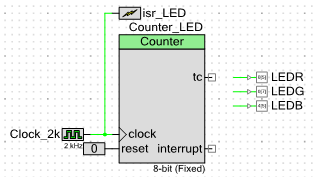
PSoC Creator nous permet donc un choix entre du matériel et du logiciel pour toutes les fonctions à implémenter.

# Introduction au programme

Le programme doit lire les fréquences retournées par chaque capteur, retrouver les couleurs correspondantes et contrôler les 2 roues motrices du robot suiveur de ligne. L’organigramme ci-dessous informe des liens entre les fonctions (appelantes en haut, appelés en bas), de leurs fichiers et de la page de leurs descriptions dans ce rapport. Les fonctions *CY\_ISR* sont des interruptions.

# Changement de LED

La LED allumée est choisie par un compteur allant de 0 à 3. Une horloge de 2 kHz commande ce compteur et une interruption. Voici le schéma correspondant :



À chaque interruption d’*isr\_LED*, les LEDs doivent changer et la composante RGB correspondante à la dernière LED allumée ou la référence doit être acquise avec la fonction *acquisition*. Si les 4 valeurs sont correctement acquises, *isr\_LED* appelle aussi les fonctions d’étalonnage e*talonnage* s’il y a appui sur les boutons et de comparaison *convert\_RGB\_to\_HSL* pour déterminer si les capteurs sont sur la ligne ou pas.

La variable *is\_acquired* de la fonction d’interruption et d’acquisition est un tableau global de 2 cases (LEFT = 0 et RIGHT = 1). Il compte le nombre de fronts envoyés par un récepteur de lumière. S’il est inférieur à 2 lors d’une acquisition de composante, il est fixé à 3 pour signaler qu’au moins une composante RGB est fausse. Il est initialisé à 0 avant chaque acquisition de référence.

Le type *T\_sensors* est une union entre une structure (*unit*) de 2 booléens et leur concaténation (*group*).

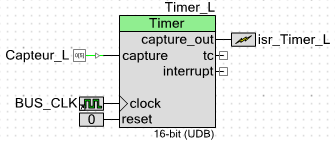
# Réception de période

Le but de cette partie est de convertir le signal carré de fréquence variable et de rapport cyclique ½ (à luminosité constante) provenant du récepteur en une variable proportionnelle à la fréquence et mise à jour le plus souvent possible.

Pour la rapidité, on mesure une demi-période (entre 2 fronts) avec un temporisateur (timer en anglais). On capture la valeur de son décompteur à chaque front du signal d’entré dans un buffer interne au temporisateur et on génère une interruption pour l’utiliser.

À chaque interruption, la valeur capturée est soustraite à celle de la précédente interruption, ce qui donne une valeur proportionnelle à la demi-période, puis est enregistrée dans la variable statique *previous\_data*. Des débordements du compteur interne au temporisateur surviennent périodiquement. Lorsqu’on soustrait ces deux valeurs un modulo 2N intervient grâce au débordement de la soustraction. Il permet de trouver la bonne valeur lorsque le compteur a débordé mais il ne doit jamais déborder plus d’une fois par mesure. C’est pourquoi la période du compteur doit toujours être supérieure au temps mesuré. En utilisant une horloge de 24MHz (précision maximum disponible de 42 ns), un compteur 16 bits nous autorise à mesurer un maximum de 2,73 ms. Le signal d’entrée doit donc être comprit entre 183 Hz et 12 MHz ce qui est notre cas.

Voici le schéma correspondant dans le cas du capteur gauche :



Le bloc *Capteur\_L* est une entrée numérique reliée au récepteur. L’interruption *isr\_Timer\_L* appellent la fonction *capture\_demi\_period* où les tableaux *demi\_period* et *is\_acquired* sont des tableaux globaux de 2 cases (LEFT = 0 et RIGHT = 1) :

Le tableau *previous\_data* est défini en statique pour être mémorisé entre chaque appel de *capture\_demi\_period.*

Nous avons maintenant par capteur une variable globale *demi\_period* contenant fois la période qui se met à jour automatiquement.

# Acquisition d’une composante RGB

À chaque changement de LED la dernière demi-période mesurée doit être enregistrée en tant que composante RGB ou référence. Pour minimiser l’effet des néons dont la luminosité varie à 50 Hz, notre capteur trouve 500 couleurs à la seconde. Nous changeons donc les LEDs toutes les 500 µs (2000 fois par seconde). Pour avoir une valeur de demi-période correcte, il faut au moins qu’il y ait un état complet en sortie du récepteur entre 2 changements de LED et ce le plus éloigné possible du régime transitoire de ce même récepteur.

Le tableau *is\_acquired* devient donc très utile pour savoir si le tableau *demi\_period* contient des valeurs utilisables.

La fonction *acquisition* enregistre la fréquence référence ou remplie une composantes RGB avec la fréquence en tenant compte du nombre d’acquisitions de période réalisées depuis son dernier appel, de la référence et des coefficients entre les couleurs. S’il n’y a pas eu au moins 2 acquisitions de période depuis son dernier appel, il prévient la fonction d’*isr\_LED* qu’une composante est fausse en plaçant *is\_acquired[LEFT* ou *RIGHT]* à 3.

La constante *MAX\_DEMI\_PERIOD* utilisée dans *acquisition* est égal à où *TIMER\_CLK* vaut 24000000 (24 MHz) et *MAX\_FREQ* vaut 1000000 (1 MHz : fréquence supérieure au maximum atteint par le récepteur de lumière). Elle permet de dynamiser la fréquence reçue sur 8 bits.

*RED*, *GREEN* et *BLUE* valent respectivement 0, 1 et 2.

# Étalonnage entre les composantes RGB

À cause de la différence de puissance entre les 3 LED et de l’atténuation du récepteur en fonction de la longueur d’onde de la lumière, on ne reçoit pas également les composantes RGB. Pour contrer ce phénomène on associe un coefficient à chaque composante. On voudrait que devant une surface blanche les trois composantes soit égales et qu’elles soient loin de la saturation (on prend 64 sur 256). Pour trouver ces coefficients, la fonction *etalonnage*, qui est appelée lorsqu’on appuis sur le bouton associé au capteur à étalonner, calcule ces trois coefficients.

# Conversion du RGB au HSL

Le système HSL (teinte, saturation, lumière en français) utilise un codage de couleur facilement compréhensible pour un humain. Dans le programme, il nous permet de placer des seuils pour reconnaitre le noir, le blanc, le gris et les différentes couleurs précisément. En plus de convertir, la fonction *convert\_RGB\_to\_HSL* renvoie un booléen informant si la couleur RGB correspond à la couleur de la ligne réglée par des seuils.

La structure *order* utilisée dans *convert\_RGB\_to\_HSL* donne les index triés du tableau *color* grâce à la fonction *sort\_compos*.

# Commande des moteurs

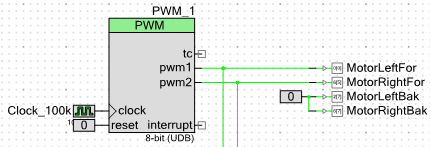
La fonction *commande\_moteur* doit faire suivre la ligne au robot à partir des 2 booléens données par la partie de captage. Il y a donc 4 cas possibles :

Lorsque le robot a :

* les 2 capteurs sur la ligne, il avance droit ;
* le capteur gauche seul sur la ligne, il tourne légèrement à gauche ;
* le capteur droit seul sur la ligne, il tourne légèrement à droite ;
* aucun capteur sur la ligne, il tourne autour de la roue qui était la plus lente (effet mémoire).

Les 2 constantes *V\_LENT* et *V\_RAPIDE* permettent de régler rapidement les vitesses possibles. Le coefficient *MOTOR\_COEFF\_L* permet de ralentir la roue gauche (roue la plus rapide d’après les tests) d’un pourcentage.

Les PWMs utilisés pour commander les moteurs sont créés par ce schéma :



Ils ont une pseudo-fréquence de fclock /28 = 391 Hz.

Les sorties *MotorLeftBak* et *MotorRightBak* sont misent à ‘0’ car le robot n’a pas besoin de reculer.

# Initialisation et contrôle du robot

L’initialisation consiste à amorcer les modules externes au processeur par les fonctions *\*\_Start* et à désactiver les interruptions (sauf celle d’*isr\_On\_Off*) pour ne pas que le robot démarre immédiatement. Ensuite, la fonction *main* scrute la variable globale *isFollowing* qui s’inverse à chaque appuis sur le bouton On/Off grâce à l’interruption de *isr\_On\_Off* pour activer ou désactiver les interruptions. Une bascule D permet d’éviter les rebonds du bouton On/Off qui donneraient un aspect aléatoire lors des appuis :

