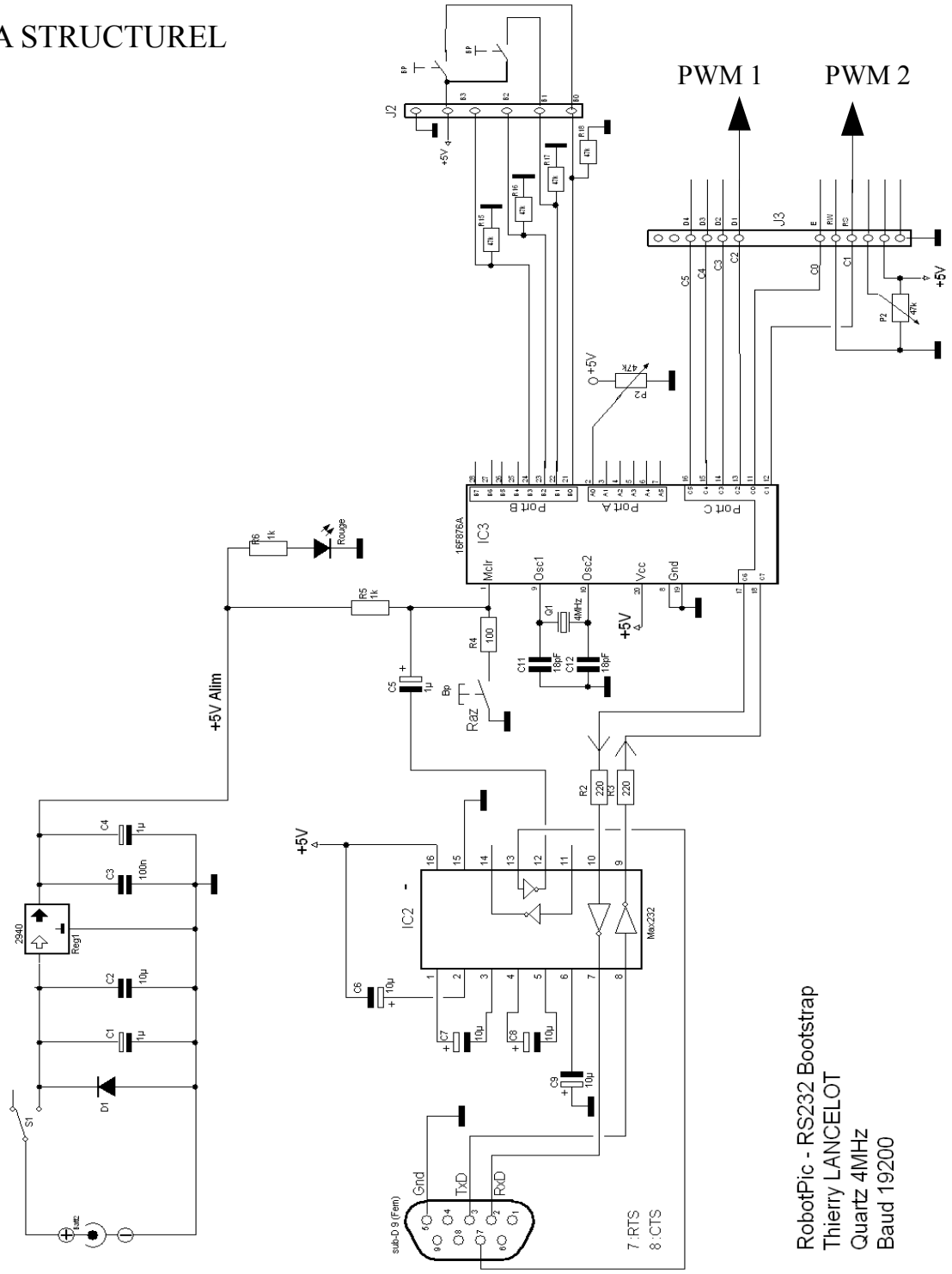
Objectif : Il s'agit de comprendre et d'utiliser le module CCP des pics pour fournir un signal PWM.

1 - LA CIBLE

Les programmes sont réalisés sur la platine BootRS232 **sans affichage LCD**, en effet, la sortie du signal PWM s'effectue sur le port C2. Cette cible est constituée par un microcontroleur 16F876A avec un oscillateur interne à 4MHz. Pour fonctionner en mode bootstrap, le pic devra être programmé une première fois avec un bootloader (16F876A_4M.hex), le test des programmes s'effectuera avec Tinybootloader dont l'appel est possible dans la version 2.07 de Logipic.

L'ensemble des explications ne s'attardera pas sur la prise en main de Logipic considérant que vous connaissez suffisamment ce formidable logiciel.

SCHEMA STRUCTUREL



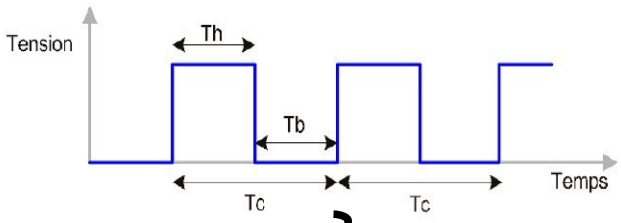
2 - RAPPEL SUR UN SIGNAL PWM

Le PWM, Pulse With Modulation ou MLI, Modulation de largeur d'impulsion est un signal de fréquence fixe dont le rapport cyclique peut être modifié par logiciel. Ce signal PWM utilisera une pin du Pic configurée en sortie.

F = 1/Tc

Tc = Th + Tb

Raport cyclique : Rc = Th / Tc en %



Un signal de Rc à 0 % est un signal constamment à l'état bas.
Un signal de Rc à 100 % est un signal constamment à l'état haut. } Ce sont des signaux non périodiques
Le rapport cyclique d'un signal PWM doit donc être supérieur à 0 % et inférieur à 100 %

3 - CALCUL DE Tc

Les pics travaillent avec 2 informations pour créer notre signal :
Tc donc F qui sera fixe,
Th donc Rc qui sera variable.

L'information Tc est gérée à partir du timer2. Il faut donc configurer le prédiviseur et PR2 (le postdiviseur n'intervient pas).
La mise en œuvre du timer2 s'effectue grâce au registre **T2CON et au registre PR2**

T2CON :

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Non utilisé	**TOUTPS3**	**TOUTPS2**	**TOUTPS1**	**TOUTPS0**	**TMR2ON**	**T2CKPS1**	**T2CKPS0**
0	**/5 Calcul du postdiviseur**				**1**	**/16 Calcul du prédiviseur: 0 1**	

TOUTPS3 :
TOUTPS2 : } choix du postdiviseur (tous à 0)
TOUTPS1 :
TOUTPS0 :

Après un reset le registre **T2CON** se positionne à 00000000

TMR2ON : mise en route du timer2. 0 : arrêt. 1 : marche.
T2CKPS1 :
T2CKPS0 : } choix du prédiviseur

bit1 : T2CKPS1	**bit0 : T2CKPS0**	/ Prédiviseur
0	0	1
0	1	4
1	0 ou 1	16

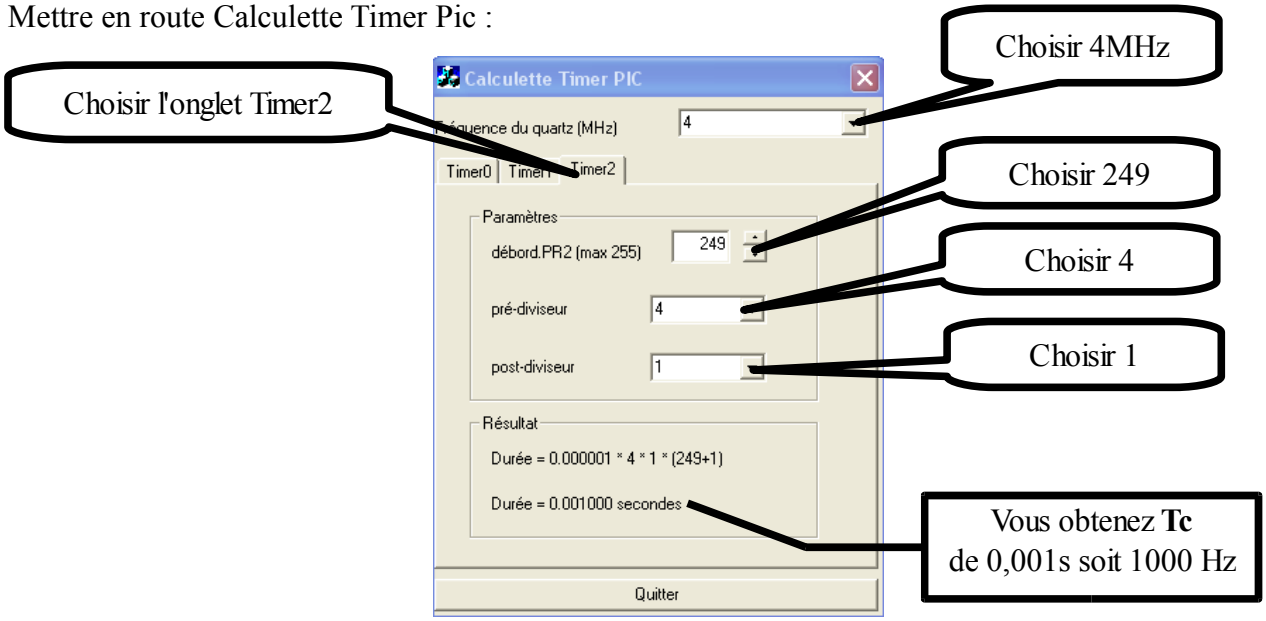
Le calcul Tc s'effectue de la façon suivante :
Sachant que le quartz utilisé est de 4 Mhz, nous pouvons en déduire que Tosc de 1/ 4E6 = 2,5E-7

$$Tc = (PR2 + 1) \times 4 \times Tosc \times prédiviseur$$

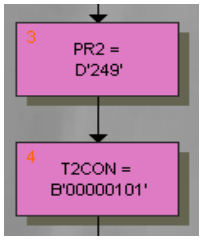
ou alors

$$PR2 = (Tc / (4 \times Tosc \times prédiviseur)) - 1$$

Dans ce 1° tutoriel, commençons avec une fréquence de 1000Hz soit Tc = 0,001s
Mettre en route Calculette Timer Pic :



Nous avons toutes les informations pour commencer l'écriture sur Logipic et fixer Tc à 0,001s :



4 - MISE EN OEUVRE DU MODULE CCPx

Le 16F876A dispose de 2 modules CCP : CCP1 et CCP2 qui sont quasiment identiques (le module CCP2 permet en plus de démarrer automatiquement la conversion A/D mais je n'en parlerai pas...si vous arrivez jusque là, vous pouvez aussi étudier les documents de Bigonoff ou de Microchip).

CCP1 est disponible sur le PortC2 (pin 13). CCP2 est disponible sur le PortC1 (pin 12).

Ces deux modules sont gérés par leur registre respectif : CCP1CON et CCP2CON.

CCP1CON :

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Non utilisé	**Non utilisé**	**CCP1X**	**CCP1Y**	**CCP1M3**	**CCP1M2**	**CCP1M1**	**CCP1M0**
0	**0**	**Complément bits**		**Choix du mode de fonctionnement : 1100 pour PWM**			

CCP2CON :

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
Non utilisé	**Non utilisé**	**CCP2X**	**CCP2Y**	**CCP2M3**	**CCP2M2**	**CCP2M1**	**CCP2M0**
0	**0**	**Complément bits**		**Choix du mode de fonctionnement : 1100 pour PWM**			

Le bit5 et bit4 permettent d'ajouter 2 « décimales binaires » pour augmenter la précision finale.
Avec un prédiviseur de 1 , la précision est de Tosc,
Avec un prédiviseur de 4 , la précision est de 4 x Tosc,
Avec un prédiviseur de 16 , la précision est de 16 x Tosc,

Sans oublier la mise en action du port C en sortie en fonction des modules CCPM
TRISC,2 à 0 pour le module CCP1 ou TRISC,1 à 0 pour le module CCP2.

5 - CALCUL DE Th

En positionnant le registre CCPR1L, (ou CCPR2L), nous pourrons déterminer Th donc le rapport cyclique Rc du signal. Rappel : Rc = Th / Tc

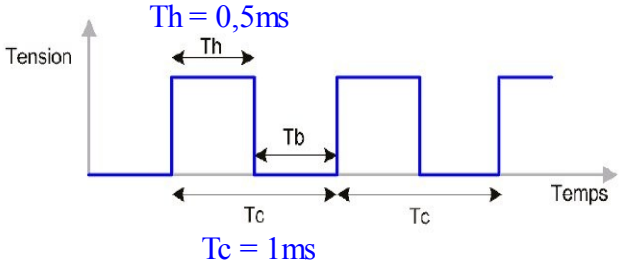
On calcule la valeur de comparaison DCB :

DCB = Th / (prédiviseur x Tosc)

ou alors

Th = DCB x Tosc x prédiviseur

Commençons avec un rapport cyclique de 50% .
Notre Tc est de 1 ms (voir paragraphe 3)
Il nous faut donc un Th de 0,5 ms
Soit :
DCB = 0,5E-3 / 4 x 2,5E-7
DCB = 500
codé en binaire sur 10 bits cela donne :
500(d) = 0 1 1 1 1 1 0 1 0 0 (b)



CCPR1L

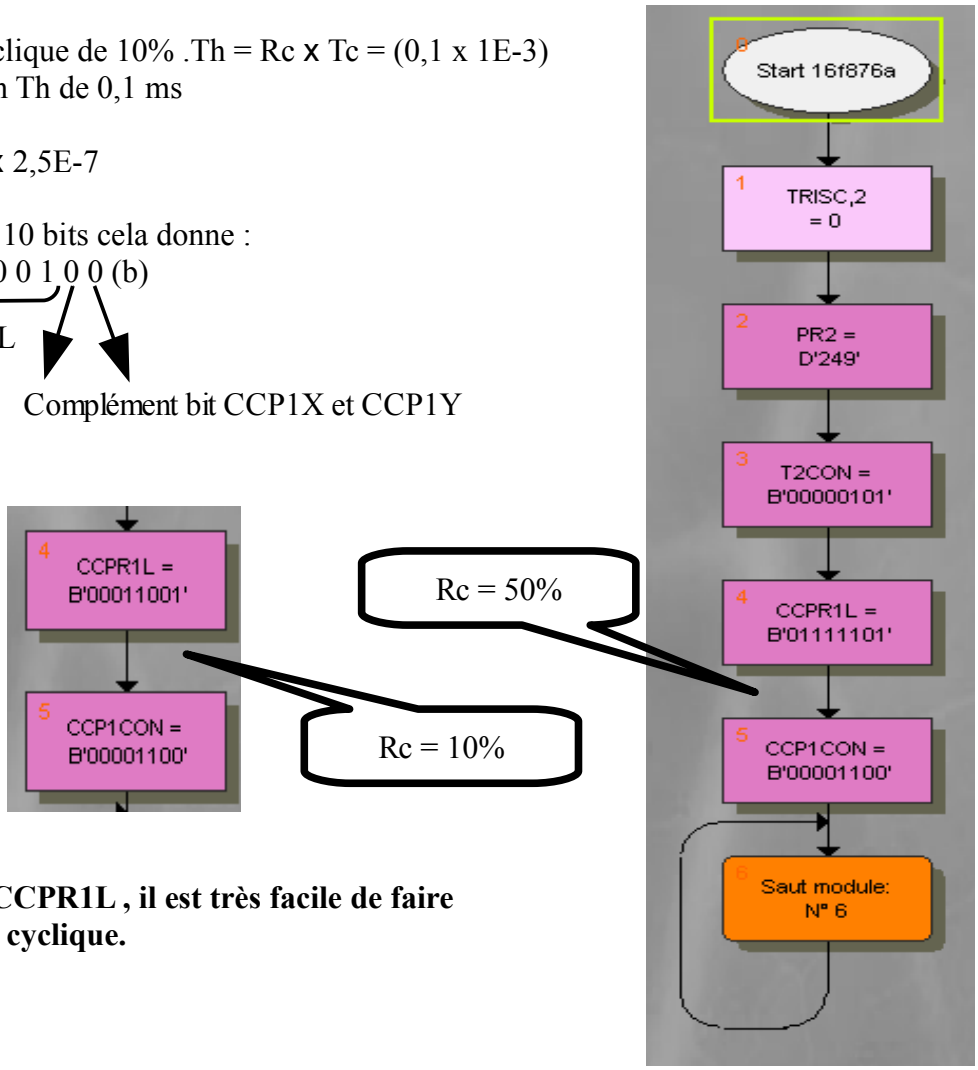
Complément bit CCP1X et CCP1Y du registre CCP1CON

Nous avons maintenant toutes les informations pour écrire l'obtention d'un signal de 1KHz avec un rapport cyclique de 50% :

Pour un rapport cyclique de 10% .Th = Rc x Tc = (0,1 x 1E-3)
Il nous faut donc un Th de 0,1 ms
Soit :
DCB = 0,1E-3 / 4 x 2,5E-7
DCB = 100
codé en binaire sur 10 bits cela donne :
100(d) = 0 0 0 1 1 0 0 1 0 0 (b)

CCPR1L

Complément bit CCP1X et CCP1Y

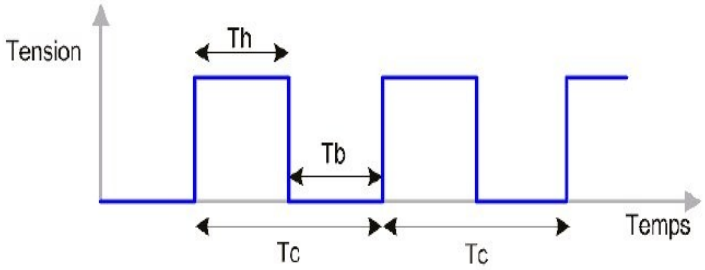


En incrémentant CCPR1L , il est très facile de faire évoluer le rapport cyclique.

6 - COMMANDE D'UN SERVOMOTEUR

La commande d'un servomoteur nécessite la fourniture d'impulsions sur la broche de commande du servomoteur. L'angle de rotation du servomoteur dépend de la l'impulsion Th avec une répétition Tc inférieure à 20ms.

Une impulsion de 1 ms génère un angle de -90°
Une impulsion de 1,5 ms génère un angle de 0°
Une impulsion de 2 ms génère un angle de +90°



Th de 1 ms à 2 ms

Tc < 20 ms

6.1 - Calcul de Tc

Avec un quartz de 4MHz, il est nécessaire d'utiliser les valeurs maximum de PR2 et du prédiviseur :
Avec PR2 = 255 (valeur maxi) et un prédiviseur de 16 (valeur maxi), nous obtenons Tc :

$$Tc = (PR2 + 1) \times 4 \times Tosc \times prédiviseur$$

Soit Tc = (255 + 1) x 4 x 2,5E-7 x 16 Tc = 4,096 ms

Cette valeur est « limite » par rapport au 20ms mais fonctionne correctement. Au besoin , il est possible de diminuer la fréquence du quartz...

6.2 - Calcul de Th

La valeur Th devra varier de Thmini de 1 ms à Thmaxi à 2 ms.

$$DCB = Th / (prédiviseur \times Tosc)$$

DCBmini = 1E-3 / (16 x 2,5E-7) DCBmini = 250
DCBmaxi = 2E-3 / (16 x 2,5E-7) DCBmaxi = 500

Remarque : Thcentre = 1,5 ms DCBcentre = 375

Une bonne solution serait d'avoir 250 pas pour la rotation du servomoteur. La précision sera ainsi de 250 pas sur 180° de rotation soit 0,72° par pas : valeur largement suffisante.

Une variable (appelons la rotation) codée sur 8 bits varie de 0 à 255, en répartissant les 5 valeurs de part et d'autres de DCB, il est possible de faire varier DCB de 248 à 503 (il est possible aussi de faire 247 à 502...ou 250 à 505...etc...). Valeurs compatibles avec les servomoteurs.

Dans ce cas là, la valeur Th varie de :

$$Th = DCB \times Tosc \times prédiviseur$$

Thmini = 248 x 2,5E-7 x 16 Thmini = 0,992 ms
Thmaxi = 503 x 2,5E-7 x 16 Thmaxi = 2,012 ms

Ceci étant la variable rotation doit évoluer entre 0 et 255 et DCB entre 248 et 503. Le problème qui se pointe est la nécessité de travailler sur au moins 9 bits, en effet, impossible de coder une valeur supérieure à 255 avec 8 bits...

6.3 - Transformation de la variable rotation

Pour résumé :
- la variable rotation évolue de 0 à 255,
- DCB doit évoluer de 248 à 503,

A priori, c'est simple : il suffit d'additionner 248 à rotation. DCB = rotation + 248(d)

Prenons des exemples :

Valeur de rotation = 0

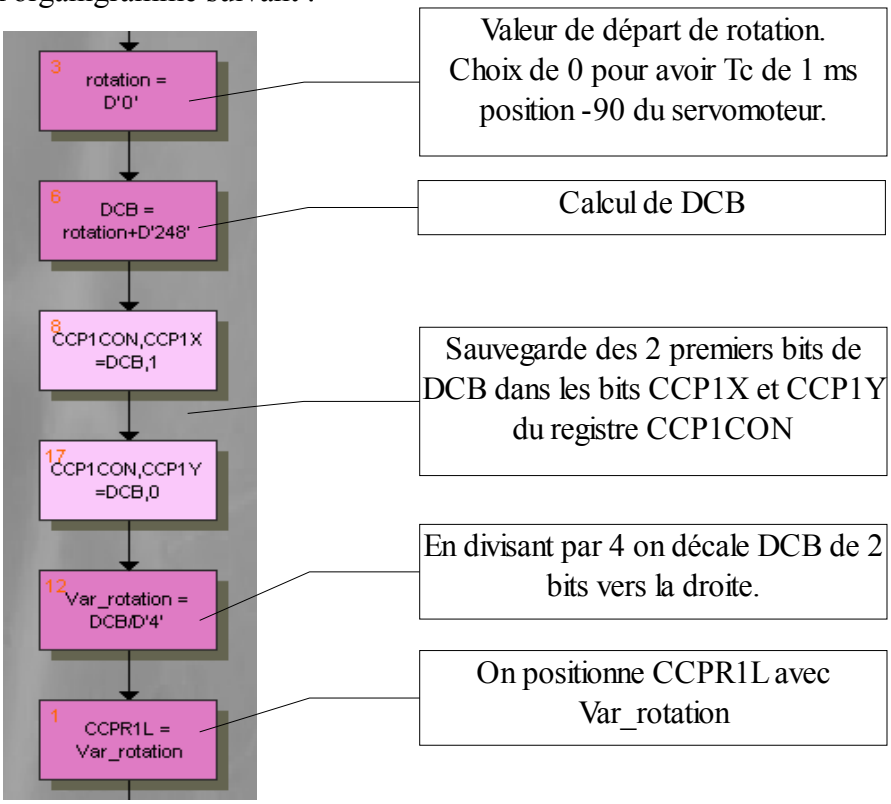
rotation = 0 → DCB = rotation + 248(d) → DCB = 248(d) ou 11111000(b)
codé en binaire sur 10 bits cela donne :
DCB = 248(d) = 0 0 1 1 1 1 1 0 0 0 (b) Var_rotation = 00111110(b) ou 62(d)

CCPR1L

Complément bit CCP1X et CCP1Y

Pour résumé, les 2 premiers bits de DCB permettent de compléter CCP1X et CCP1Y du registre CCP1CON. Pour pouvoir compléter CCPR1L, il faut décaler de 2 bits DCB.

Pour décaler 1 bit, il suffit de diviser par 2 et pour décaler de 2 bits de diviser par 4. Nous retrouvons donc l'organigramme suivant :



Valeur de rotation = 8

rotation = 8 → DCB = rotation + 248(d) → DCB = 256(d) ou 100000000(b)
codé en binaire sur 10 bits cela donne :
DCB = 256(d) = 0 1 0 0 0 0 0 0 0 0 (b) RRF de DCB = 10000000(b)
DCB/2 = 01000000(b) ou 64(d)

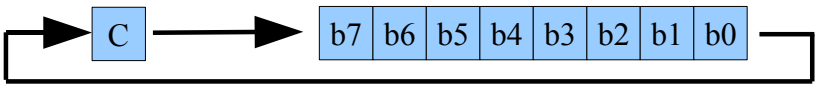
CCPR1L

Complément bit CCP1X et CCP1Y

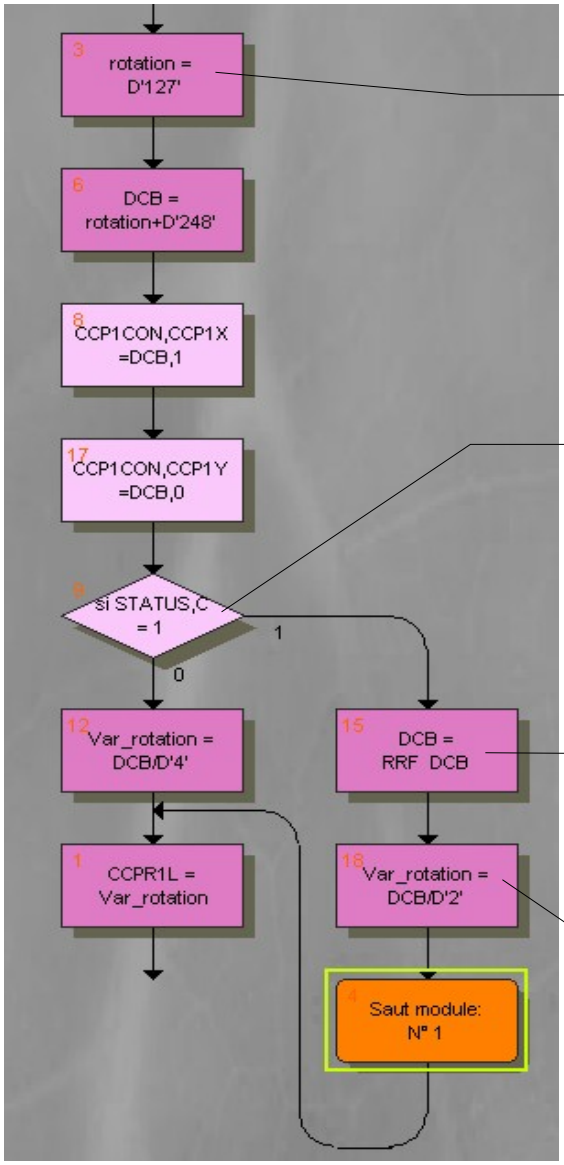
Dans ce cas là, DCB se positionne à 00000000, les 2 premiers bits de DCB permettent de compléter CCP1X et CCP1Y du registre CCP1CON.

Pour récupérer le 9° bit, il faut tester la retenue (carry du registre STATUS), ensuite, la commande RRF (Rotate Right through Carry) permet d'effectuer une rotation à droite en intégrant la retenue et enfin on termine avec une rotation de 1 bit vers la droite en effectuant une division par 2.

Remarque : dans un premier temps, j'ai essayé de travailler avec des RRF, cependant cette solution s'est avérer incorrecte en effet, la rotation positionne le bit0 en carry...etc...



L'organigramme suivant permet de prendre en compte toutes les valeurs possible de la variable rotation et de mettre le servomoteur en position centrale :



Valeur de départ de rotation.
Choix de 127 pour avoir Tc de 1,5 ms position 0 du servomoteur.

Test permettant de vérifier s'il y a une retenue.

Si c'est la cas, la retenue est positionnée en bit 7 de DCB

En divisant par 2 on décale DCB de 1 bit vers la droite.

L'organigramme de l'ensemble de commande d'un servomoteur se retrouve dans le projet servo.prj

L'évolution de la variable rotation s'effectue avec B2 (incrémentation) et B3 (décrémentation).

Cependant, en effectuant une conversion A/D, il est très facile de commander la position du servomoteur :