TPE 2002/2003
VOITURE TÉLÉCOMMANDÉE

Sebastien Chapenoire
Nicolas Viennot

...:: INTRODUCTION ::...

Le TPE

Dans le cadre des Travaux Personnels Encadrés 2002/2003, qui ont débuté au début du mois d'octobre 2002 et ce sont achevés à la fin du mois de janvier 2003, nous avons réalisé une voiture télécommandée par ordinateur. Cette idée nous est venu au départ car nous sommes passionnés par la robotique, l'électronique, l'informatique et tous les domaines se rattachant autour de cela. Nous avons au cours de ces quatre mois conçu et créé en totalité un engin piloté par ordinateur.
Nous avons essayé le mieux possible de réaliser un véhicule voué à l'exploration, c'est pourquoi la voiture est équipée d'une transmission radio AM afin de pouvoir disposer d'un rayon de manœuvre suffisant. Elle est aussi dotée d'une vitesse relativement importante de 30 km/h, de suspensions dans l'optique d'explorer de multiples terrains et d'une bonne autonomie de fonctionnement. Un gros travail de recherche a été obligatoire afin d'acquérir les divers connaissances que nous avons utilisé pour mener le projet à son terme avant même de commencer la conception des différentes parties de la voiture. Cette dernière s'est étalée sur environ trois mois et il fut difficile de s'organiser afin d'optimiser le mieux possible le temps qui nous a été imparti. Nous avons perdu de précieuses heures dans beaucoup de domaines concernant la conception car il est difficile de cerner rapidement le travail à réaliser et les solutions techniques que nous avons utilisé, tant elles sont divers. La conception s'est scindée en trois partie : la partie mécanique, la partie électronique et enfin la programmation. Une fois cela suffisamment entamé, nous avons pu commencer la réalisation de la voiture. Cette partie comprend l'usinage des pièces mécaniques, la création des cartes électroniques et celle des programmes informatiques. La dernière étape fut l'imposante partie de tests afin de coordonner la mécanique, la transmission radio, l'électronique et la programmation.

...:: LE SOMMAIRE ::...

LA MÉCANIQUE

Sebastien Chapenoire

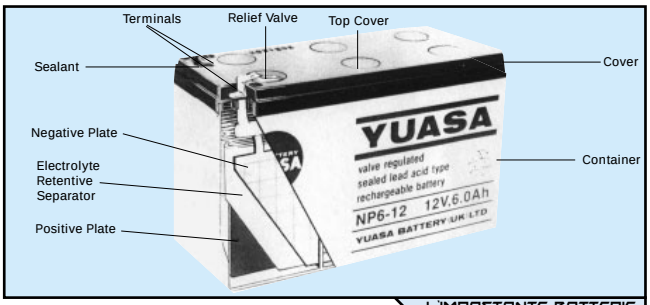
...:: L'APPROCHE THÉORIQUE ::...

Principe

Le premier travail auquel nous nous sommes attachés a été d'établir les différentes fonctions que devait remplir la voiture au niveau mécanique, dont voici la description.

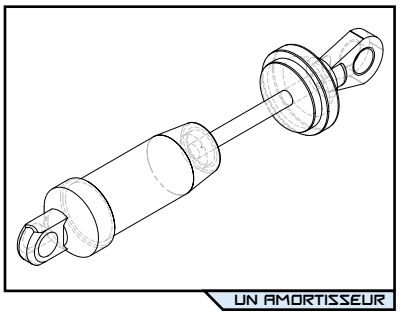
Nous avons prévu au départ que la voiture ait une vitesse maximale de 30 km/h. Nous avons donc essayé de nous en rapprocher le plus possible aux moyens des éléments déjà existant dans le commerce, c'est à dire les moteurs, les pignons et les roues dentées. Cette vitesse est à notre avis relativement acceptable car elle permet de nombreuses performances très correctes étant données les solutions techniques retenues. Nous pensons en effet qu'il est déjà compliqué de faire avancer très précisément un tel engin, c'est pourquoi nous avons trouvé ce compromis. Certes il existe beaucoup de voitures électriques modèle-réduit ayant des vitesses bien supérieures, mais il s'agit pour notre projet d'un prototype et nous préférons créer entièrement la voiture, plutôt que de récupérer des éléments tels qu'une plate-forme de modélisme, un châssis déjà fabriqué, et ainsi ne faire que modifier un engin déjà existant. Rentrons maintenant plus en détails dans les solutions techniques du projet. Nous avons opté pour une propulsion de la voiture et pour un contrôle de la direction par différentiel, c'est à dire par modification du rapport cyclique des deux moteurs à courant continu, ou plus simplement en faisant varier la vitesse d'une des roues par rapport à l'autre, forçant ainsi le véhicule à tourner. C'est le principe que l'on peut retrouver sur de nombreux engins de chantier à chenilles. Ainsi la voiture pourra pivoter sur elle-même autant en marche avant qu'en marche arrière en faisant tourner les roues arrières en sens contraire l'une par rapport à l'autre. Cette solution technique a été préférée au banal système de direction que l'on retrouve sur toutes les voitures, c'est à dire une crémaillère ainsi qu'un système de tringlerie, qui réduit le champ des manœuvres en raison du rayon de braquage. De plus ce type de direction requiert l'ajout d'un servo moteur, élément assez onéreux pour l'utilisation qui en est faite dans notre projet. Ayant choisit ce type de motorisation, il était donc obligatoire de penser à créer un système destiné à permettre la rotation des roues avant sans difficulté, puisqu'elles ne sont pas motorisées, ce qui sera décrit ultérieurement dans le dossier. Autre élément important à prendre en compte, le véhicule est entièrement suspendu au moyen de quatre amortisseurs à pression d'huile équipant chacun une roue de la voiture, et relié au châssis grâce à une pièce usinée, détaillée plus tard dans la partie mécanique. Cependant la partie moteur est dite non suspendue, c'est à dire qu'elle décrit les mêmes trajectoires que les amortisseurs, ce qui peut paraître contraire aux principes mécaniques qui conseillent d'avoir le moins possible de masse non suspendue dans un ouvrage tel que le nôtre. Nous aurions pu suivre ces conseils, mais il aurait fallu pour cela utiliser une transmission à cardan ce qui impose une conception un peu plus complexe et des pertes énergétiques par frottements supplémentaires. On peut considérer que dans notre cas, compte tenu du poids de la batterie, la masse non suspendue de la partie motrice est faible, donc relativement négligeable. Nous voulons aussi disposer d'une bonne autonomie de fonctionnement, c'est pourquoi la voiture est équipée d'une batterie au plomb gélifié d'un poids de 2,8 kg, ce qui représente une masse importante pour la taille du véhicule. C'est l'élément le plus volumineux, il a donc été préférable de la placer judicieusement, ce qui sera précisé plus loin dans le dossier. Les parties usinées de la voiture sont quasiment toutes réalisées en aluminium, mise à part la carrosserie qui sera faite en résine transparente et les axes qui sont en acier pour une meilleure solidité.

UN MOTEUR GZ-600 BB



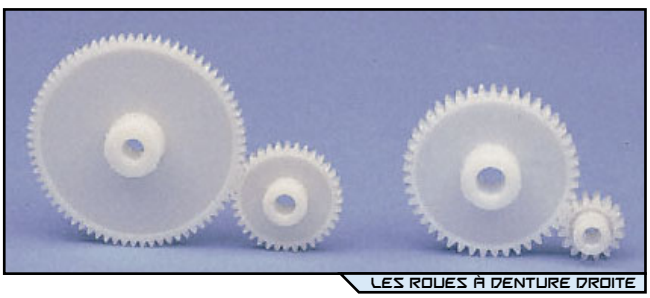
L'IMPOSTANTE BATTERIE

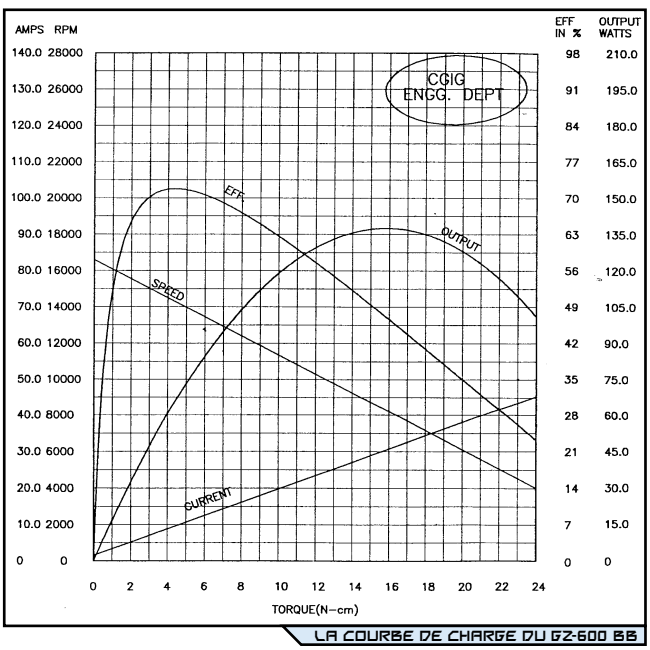

UNE CRÉMAILLIÈRE


UN AMORTISSEUR

Calculs du rapport de reduction

Les roues montées sur la partie moteur de la voiture ont un rayon de 37,5mm. On dispose de moteurs à courant continu ayant une vitesse maximale de rotation nominale de 13000 tr/min et un couple moteur de 4,25N.cm. La vitesse maximale à atteindre est de 30 km/h (= 8,33 m/s). Le périmètre de la roue est P = 235,61 mm. Donc sans réducteur, la vitesse maximale est V = 13000 x 235,61 / 60 = 51,04 m/s. Cette vitesse est bien évi-


LES ROUES À DENTURE DROITE



LA COURBE DE CHARGE DU GZ-600 BB

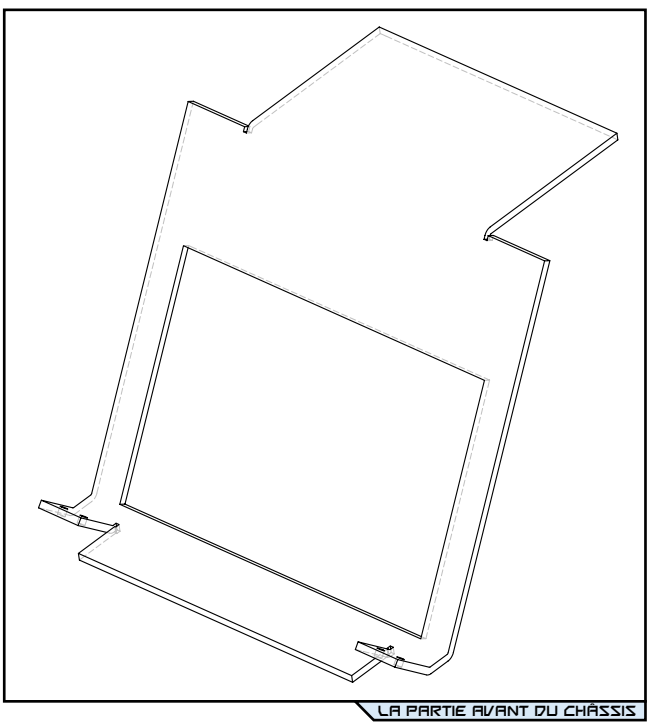
demment trop grande et le couple développé par les moteurs est bien trop faible, c'est la raison pour laquelle il faut un réducteur afin tout d'abord de réduire la vitesse et d'augmenter le couple. On peut calculer le rapport de réduction dont on a besoin : r = 51,04 / 8,33 = 6,12. Il faut donc un système réduisant la vitesse d'environ six fois. Pour cela, nous avons sélectionné deux roues dentées afin de réaliser ce réducteur : une roue de 12 dents et une roue de 72 dents. On peut vérifier la conformité de ce choix avec la formule : R = -1 x 12/72 = -1/6. Ce résultat signifie que la fréquence de rotation est divisée par six et que le sens de rotation à la sortie du réducteur est inversé par rapport à celui de l'axe du moteur. Au niveau de l'augmentation du couple, la raison (rapport de réduction) étant de -1/6, le couple devrait en théorie être multiplié par six. Cependant, en raison des pertes dues aux divers frottements , l'augmentation est en moyenne multipliée par 0,75. Donc k = 0,75 x 6 = 4,5. Le couple moteur nominal est de 4,25 N.cm, donc théoriquement, le couple exercé sur la roue est C = 4,25 x 4,5 = 19,2N.cm. Puis on peut calculer le couple afin de savoir si ce rapport correspond bien à notre demande. On fixe le temps d'accélération pour atteindre les 30 km/h en 5 secondes. Donc l'accélération est de ϕ = 8,33 / 5 = 1,66m/s². Grâce au théorème du centre d'inertie, on peut déterminer la force appliquée à la roue. Ainsi F = m x ϕ, m étant la masse du véhicule. Nous l'avons évalué à 3,5kg. Ainsi, F = 3,5 x 1,66 = 5,81N. Grâce à cette valeur, on peut désormais déterminer le couple exercé. C'est en réalité le moment de la force F, c'est pourquoi il faut utiliser le rayon de la roue. C = F x R = 5,81 x 0,0375 = 21N.cm, ce calcul est valable pour une roue, donc avec les deux roues, on obtient un couple de 40N.cm. Ainsi, on assure la possibilité de monter des pentes sans problèmes. De toutes les manières, les valeurs utilisées dans calculs précédents ne sont pas définitives car il va s'en suivre une grosse partie de tests qui fixera ces résultats définitivement. Sur la courbe caractéristique du moteur, on voit que le couple maximal fournit peut dépasser la valeur nominale donc les tests nous promettent de bien belles surprises.

...:: LA CONCEPTION ET LA RÉALISATION ::...

Toutes les pièces usinées du véhicule ont été au préalable conçues sur papier, puis elles ont été dessinées sur le logiciel SolidWorks 2001. Tous les dessins et les vues en perspective inclus dans ce dossier proviennent de ce logiciel.

Le chassis

Nous avons choisi après plusieurs semaines de réflexions de réaliser un châssis ressemblant à celui d'un véhicule deux roues, c'est à dire basé sur une forme triangulaire, pour plus de solidité. Les fixations qui engendrent des problèmes de solidité sont ainsi réduites ainsi que le nombre de pièces, contrairement à la solution envisagée ci-après, dans laquelle nous avions conçu un châssis très simple basé sur une plate-forme où venait se greffer une armature permettant la fixation d'autres éléments. Le châssis est réalisé entièrement en aluminium à partie de plaques de 3 mm d'épaisseur. Il se compose de différentes parties fixées entre elles au moyen d'un système vis/écrou indesserrables, dans l'optique de résister aux vibrations lors du déplacement de la voiture. Les écrous sont équipés pour cela d'un anneau de Nylon (Nylstop) qui permet d'empêcher le desserrage des pièces. On a préféré cette solution à la place de rivet pop car on aura certainement besoin de changer ou de modifier certaines pièces en vue de futures améliorations et donc de démonter le châssis. Les plaques ont été découpées, évidées à des endroits précis pour créer les logements de certains composants, puis pliées. Des perçages ont été réalisés afin de permettre la fixation des différents éléments qui viendront composer la voiture.



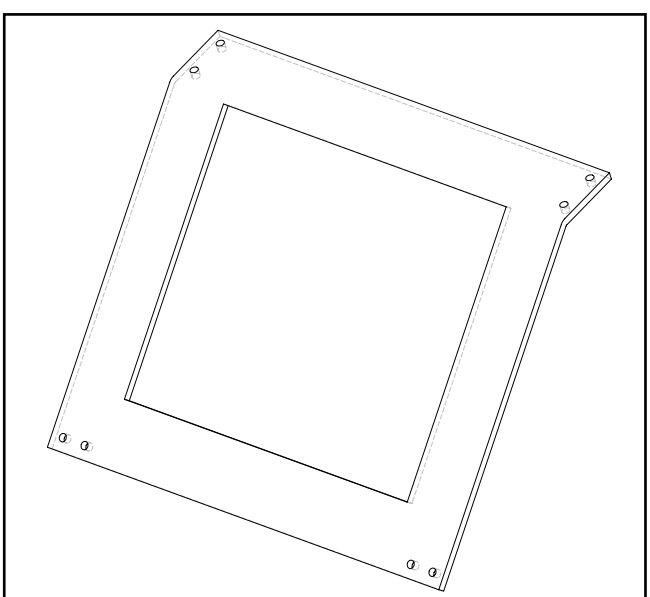
LA PARTIE AVANT DU CHÂSSIS

La partie avant du chassis

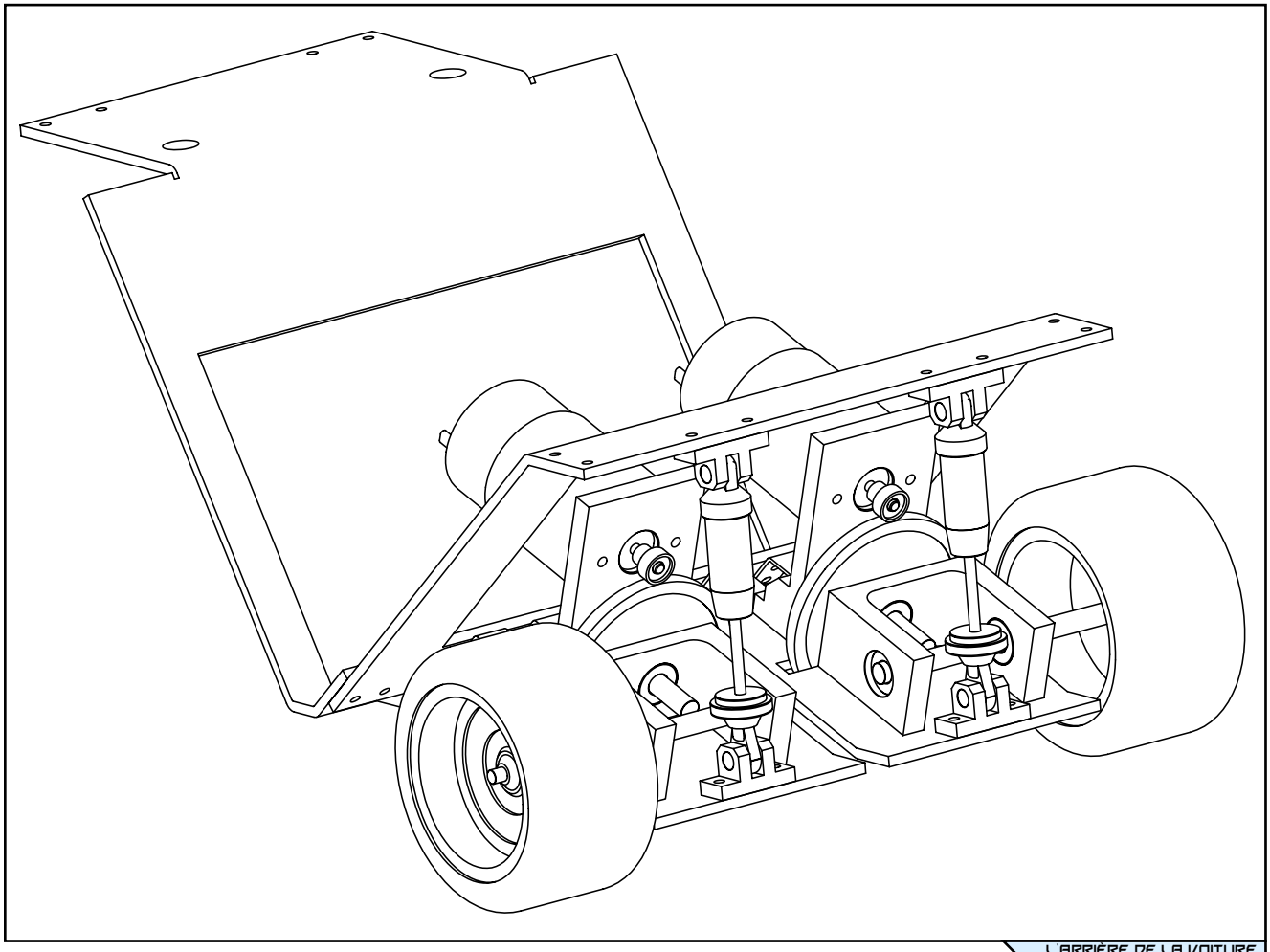
Elle est donc entièrement réalisée en aluminium, matériau léger, facile à travailler et relativement solide. Elle est fabriquée en plusieurs étapes. D'abord vient le découpage à la forme voulue, puis viennent les évidements pour la batterie et le pliage. Il y a trois traits de pliages. Les trous réalisés ont un diamètre de 3 mm. La batterie est soutenue dans son logement grâce à deux barres d'aluminium de 3 mm d'épaisseur et de 10 mm de largeur, coudées à 90°, et pliées à une des extrémités afin de permettre la fixation par deux plans parallèles. On peut remarquer que le logement de la batterie est situé le plus au centre et le plus bas possible afin d'abaisser au maximum le centre de gravité du véhicule dans l'optique de réduire le déséquilibre et l'inertie de la voiture lors de changements de direction.

La partie arriere du chassis

Cette partie vient se rattacher à la partie avant du châssis par l'intermédiaire de deux pattes de fixation situées à l'arrière de la partie avant. Un évidement est placé au centre de la pièce afin de permettre le positionnement des moteurs ainsi que leur déplacement dû au débattement des amortisseurs. La pièce a été ensuite pliée à une des extrémités et sur toute sa largeur afin de pouvoir positionner et fixer les attaches d'amortisseurs.



LA PARTIE ARRIÈRE DU CHÂSSIS



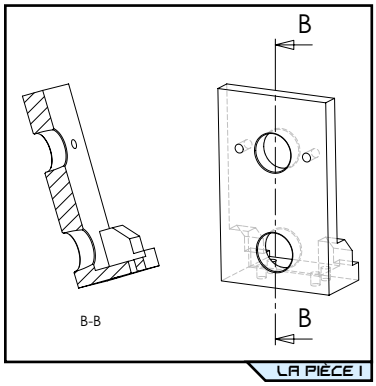
L'arrière de la voiture

La partie moteur

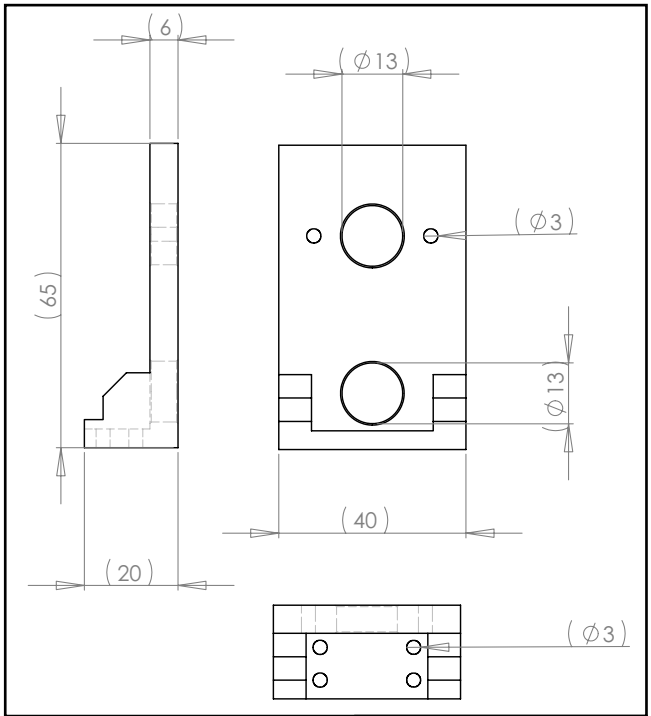
La partie moteur est composée de deux entités identiques. Elles sont chacune composées d'un moteur fonctionnant par différentiel l'un par rapport à l'autre fixé à des pièces entièrement similaires. Toutes les pièces de la partie moteur sont donc en double exemplaires dans la voiture. Il faut de plus accorder une précision plus particulière à tous ces éléments afin de respecter une parfaite symétrie au niveau de la partie motrice. En raison des performances que doit posséder la voiture, nous avons dû réduire la vitesse et augmenter le couple, et donc dû créer un réducteur (voir plus haut dans la partie calcul), ainsi qu'un renvoi d'angle afin de gagner de la place. C'est pourquoi il a fallu concevoir une première pièce pouvant permettre la fixation du moteur ainsi que celle du pignon de 12 dents et de la roue dentée de 72 dents.

Piece 1

Cette pièce est au départ un parallélépipède en aluminium, puis nous avons usiné la pièce afin de créer une forme en L, tout d'abord pour alléger la pièce, et ensuite pour permettre la mise en position du moteur grâce à un premier alésage ne demandant pas une grande précision contrairement au second destiné à contenir un roulement à billes. Voici donc quelques calculs permettant de concevoir la pièce. Nous disposons de roues dentées de module 0,8. Afin de déterminer l'entre-axe des deux engrenages, il faut donc calculer leur diamètre primitif respectif, car deux roues à denture droite utilisées dans n'importe quel système ont en permanence un point de contact qui est le point de tangence de leur diamètre primitif. On peut déterminer cela grâce à la formule suivante : d = m x Z, m étant le module et Z le nombre de dents. On obtient donc d(12) = 0,8 x 12 = 9,6 mm et d(72) = 0,8 x 72 = 57,6 mm. Pour connaître l'entre-axe, il suffit maintenant de faire l'opération suivante : D= [d(12) + d(72)] / 2 = [9,6 + 57,6] / 2 = 33,6 mm. Ayant ainsi déterminé l'entre-axe, il a fallut créer les deux alésages de 13 mm de diamètre ayant un entre-axe de 33,6 mm. En réalité l'alésage devant contenir le roulement, a un diamètre inférieur



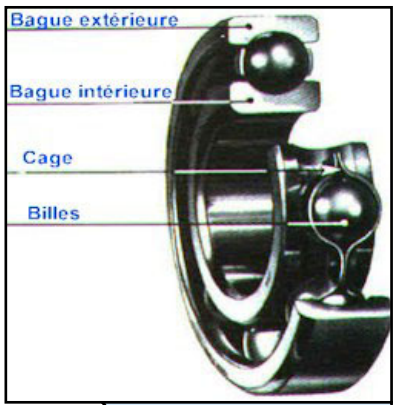
La pièce 1



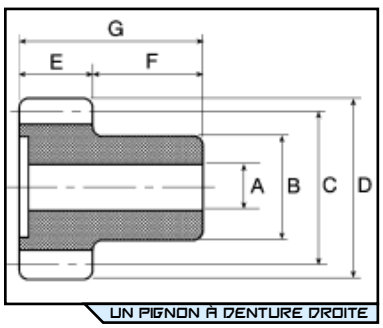
Les dimentions de la pièce 1

DES ROULEMENTS À BILLES

à celui du roulement qui est précisément de 12,98 mm. Cette opération fut réalisée en quatre étapes : un alésage avec un foret de 2 mm de diamètre. Un second avec un foret de 6 mm. Un troisième avec un foret de 12,8 mm. La finition a été réalisée à la main à l'aide d'une lime. Au final, l'alésage approche les 12,93 mm. La précision pour ce genre de manœuvre n'est pas rigoureuse car nous travaillons avec des éléments de petites tailles de l'ordre de 10 ou 20 mm et un outil tel qu'un foret a toujours une certaine flexibilité, cela créé donc une imprécision lors de l'usinage. L'opération suivante consiste en l'insertion du roulement à billes de diamètre extérieur de 13 mm en serrage. Le serrage le plus approprié pour ce type de roulement se situe entre 5 et 10 centièmes de millimètres. Le roulement est monté serré dans la pièce pour plus de facilité de montage des pièces entre elles. On aurait pu les laisser libre dans la pièce et réaliser le serrage grâce à de la colle araldite, dite colle Loctite, mais cette solution est moins dans l'esprit mécanique, c'est en quelque sorte une solution de secours. On a rajouté à la pièce quatre alésages de 3 mm de diamètre lui permettant d'être fixée plus tard sur un support d'assemblage réunissant toutes les pièces de la partie motrice.



L'INTERIEUR D'UN ROULEMENT

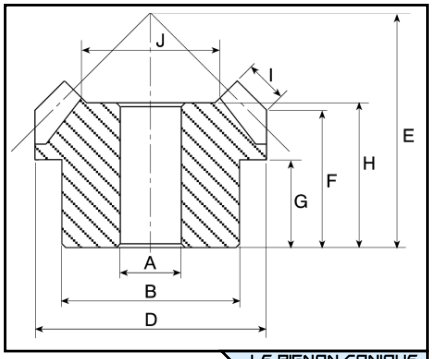


UN PIGNON À DENTURE DROITE

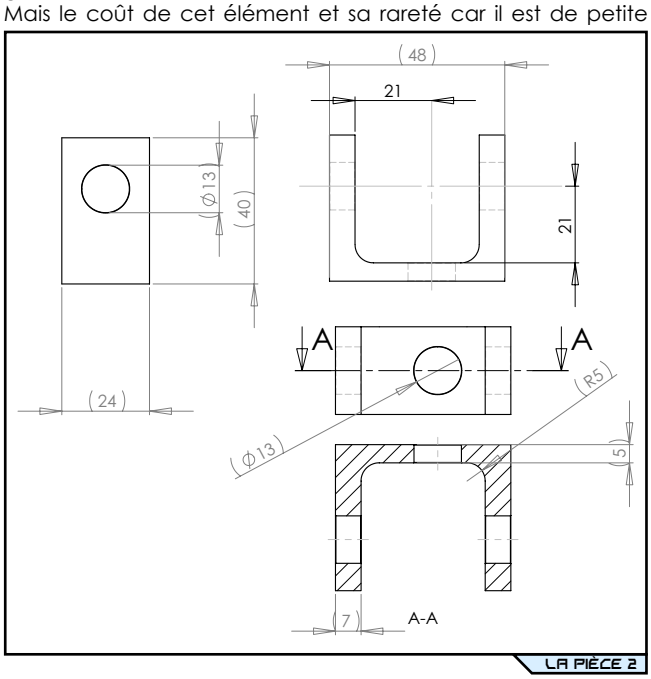
Piece 2

La seconde pièce qu'il a fallu réaliser devait contenir les pignons coniques du renvoi d'angle. Elle est aussi réalisée en double exemplaires. Les deux pièces sont équipées de roulements à billes afin de réduire les pertes mécaniques créées par les frottements des différents éléments de transmission. Le matériau utilisé pour la fabrication de la pièce est l'aluminium. Différentes opérations y ont été effectuées. Elle a tout d'abord été usinée en forme de U, puis elle a été alésée afin d'y intégrer trois roulements à billes identiques à celui inséré dans la pièce précédente. Nous avions, avant d'adopter ce schéma de pièce, fixé notre attention sur le même système mais composé de trois pièces identiques en forme de L, assemblées sur une plaque d'aluminium percée de plusieurs trous de fixation. Mais cette méthode s'est avérée moins performante car il fallait fixer les trois pièces très précisément et le nombre d'entités fabriquées passait à six alors que deux suffissent pour la solution définitive. Les roulements sont au nombre de trois : le premier est utilisé pour soutenir d'un coté, l'arbre provenant de la pièce contenant les deux roues à denture droite et de l'autre, le premier pignon conique du renvoi d'angle. Quant aux deux autres roulements, ils sont utilisés pour soutenir le second pignon conique de part et d'autre de la pièce par l'intermédiaire de l'axe relié à la roue arrière. Nous aurions pu réaliser ce système à l'aide d'un unique roulement, mais il aurait fallu pour cela un roulement à double rangées de billes pour un meilleur maintien dans l'axe de la roue. Mais le coût de cet élément et sa rareté car il est de petite taille, nous ont forcé à choisir une solution technique à deux roulements. La technique utilisée pour les différents alésages est la même que pour la pièce précédente, c'est à dire que nous nous sommes servi d'un foret de 12,8 mm. Cependant, il a fallu respecter avec grand soin les spécifications techniques de ces pignons. Elles sont décrites dans le document ci-joint où la principale grandeur à respecter est E = 21 mm, qui est la distance entre la base du pignon et le point de concours des droites confondues avec le diamètre primitif décrivant l'inclinaison de l'engrenage. Cette pièce a été usinée avec deux congés situés à la base de l'usinage intérieur et nous avons gardé une épaisseur de 7 mm sur deux des côtés afin de rigidifier cette partie, car elle subit de nombreux efforts en raison de la proximité de la roue et de la suspension.

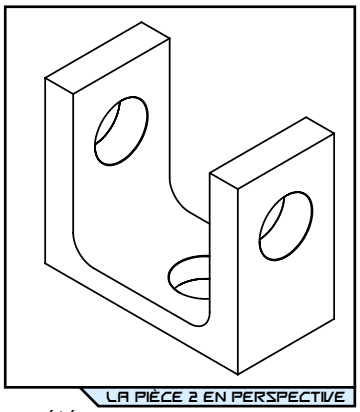

L'ENGRENAGE CONIQUE



LE PIGNON CONIQUE

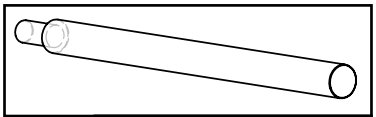


LA PIÈCE 2


LA PIÈCE 2 EN PERSPECTIVE

Les axes

Deux autres éléments viennent compléter les deux pièces décrites précédemment, ce sont deux arbres de 6 mm de diamètre. Le premier est destiné à relier la pièce supportant le moteur et les roues à denture droite, à l'autre pièce contenant les deux pignons coniques. Cet axe fait 37 mm de longueur. Il


L'AXE DE LA ROUE ARRIÈRE

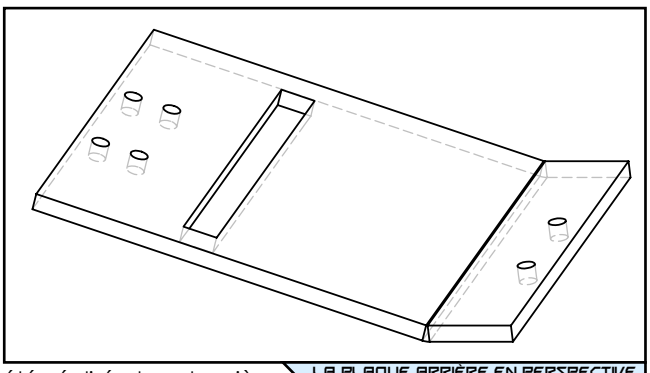

UNE ROUE

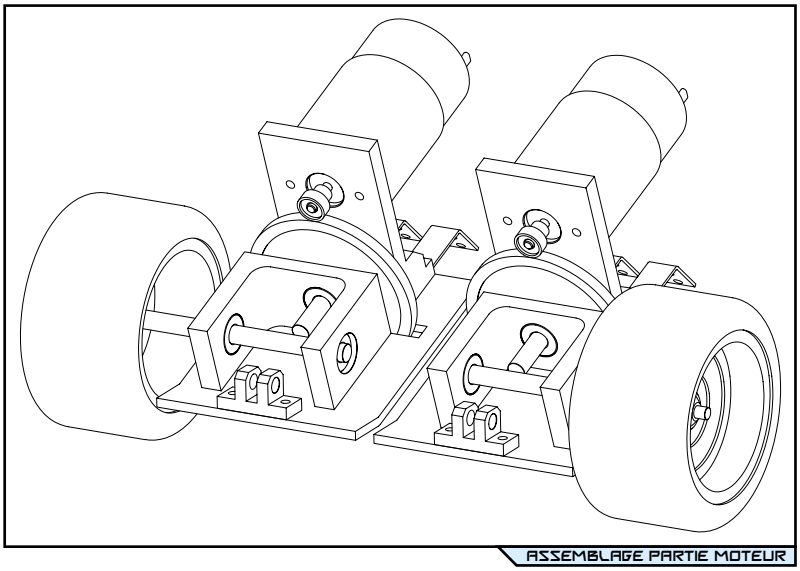
traverse l'engrenage de 72 dents, ainsi que deux roulements appartenant chacun à une pièce, puis il vient enfin s'emmancher sur le premier pignon conique. La mise en position des engrenages se fait grâce aux surfaces cylindriques et le maintien en position est réalisé au moyen de goupilles fendues traversant l'engrenage et l'axe. Un outil spécial a été fabriqué pour cette opération. Il permet d'aléser en même temps l'élément de transmission et l'axe ce qui améliore considérablement la précision de ce perçage. Quant à la roue dentée de 12 dents est maintenu en position en serrage car le diamètre de son alésage intérieur est de 3 mm et celui de l'arbre moteur est de 3,17 mm. Il y a donc 17 centièmes de serrage. Le second arbre est destiné à transmettre le mouvement de rotation du second pignon conique à la roue arrière. L'arbre traverse deux roulement à billes situés sur une même pièce ainsi que le deuxième pignon. Il fait 92 mm de longueur. Les moyens utilisés pour le maintien en position des éléments sont les mêmes que précédemment, c'est à dire la goupille fendue. La fixation de la roue sur l'axe est faite au moyen d'une tête de vis en H. En effet la jante dispose d'un logement hexagonal sur sa face intérieure, c'est pourquoi nous avons transformé une simple vis M8. Nous l'avons alésé d'un trou de 6 mm de diamètre puis nous avons surfacé les deux faces afin qu'il puisse se loger dans la jante sans difficultés. La pièce a été ensuite emmanchée en force sur l'arbre. En réalité le diamètre de l'arbre est légèrement inférieur à 6 mm, il fait précisément 5,98 mm, il a donc fallu réaliser un alésage s'approchant de 5,93mm pour la tête de vis M8. La dernière opération consista à fileter une extrémité de cet arbre sur 9 mm de longueur afin que la jante puisse être immobilisée entre la pièce hexagonale et l'écrou venant sur le filetage. L'écrou de serrage est un écrou indesserrable Nylstop de dimension M4.

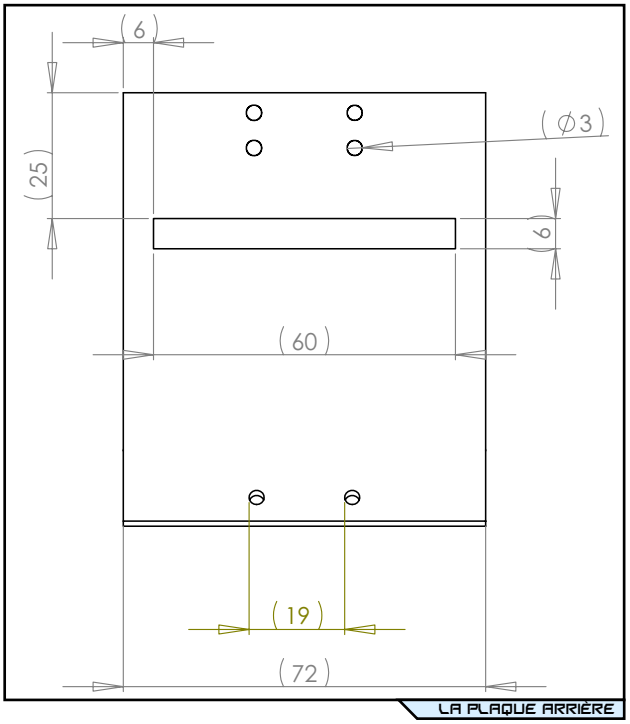

UN ECROU NYLSTOP

La plaque arriere

Dans le but d'assembler tous ces éléments, nous avons conçu autre une pièce très simple. Il s'agit d'une tôle de 3 mm d'épaisseur pliée à une extrémité et percée à plusieurs endroits. La raison du pliage sera détaillée plus tard lorsque l'assemblage du châssis et de la partie moteur sera détaillée. La plaque est dotée de six trous de 3 mm de diamètre. Quatre d'entre eux sont réservés pour la fixation du support de moteur et des engrenages à denture droite, et les deux autres servent au placement de la fixation des amortisseurs. Un évidement a été réalisé dans la pièce pour créer un logement à la roue de 72 dents afin d'éviter une collision avec la plaque compte tenu de sa grande taille. Cet élément étant la base de la partie motrice, la plaque est reliée au châssis, d'un coté par la suspension, et de l'autre par deux pattes jouant le rôle d'une liaison pivot. Ces deux dernières sont faites en acier ressort, matériau similaire aux lames de scie à métaux. Il est suffisamment solide et flexible pour supporter une trajectoire comme celle des suspensions. Une fois ces pièces conçues, il s'agit de les assembler les unes aux autres. La fixation est réalisée avec des vis M3 et des écrous Nylstop. Le couple moteur est donc transmis de pièce en pièce : en voici l'énumération. L'arbre moteur, la roue de 12 dents, la roue de 72 dents, la goupille 1, l'arbre de 37 mm de longueur, la goupille 2, le premier pignon conique, le second pignon conique, la goupille 3, l'axe de 92 mm de longueur, la tête de vis H, la jante et le pneu.


LA PLAQUE ARRIÈRE EN PERSPECTIVE

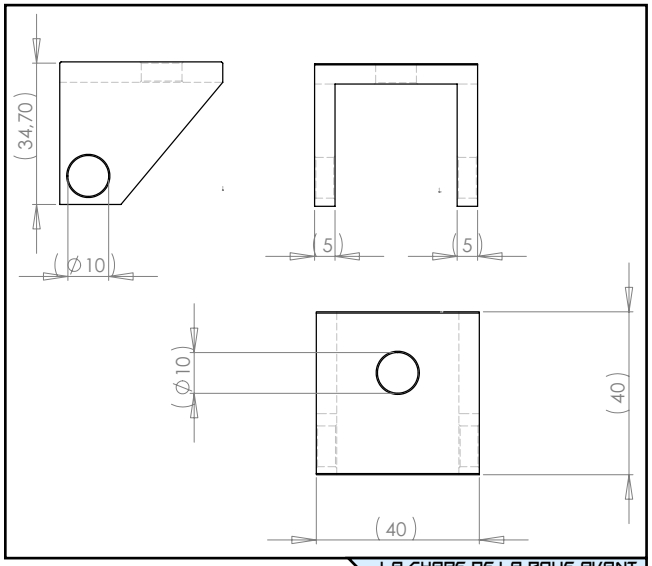

ASSEMBLAGE PARTIE MOTEUR



LA PLAQUE ARRIÈRE

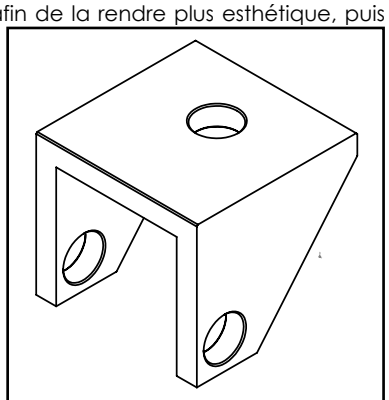
La partie avant

La partie avant du véhicule est composée des roues et des suspensions. Le système que nous avons conçu permet aux roues avant de pivoter sur elles-mêmes comme des roues de chariot. Cette solution est de rigueur dans le projet car la direction est réalisée par différentiel, et si on ne laissait aucune liberté de mouvement aux roues avant, nous serions confrontés à des frottements et, à plus long terme, à des problèmes mécaniques tels que la détérioration de certaines pièces à cause d'axes faussés. Dans ce but, nous avons



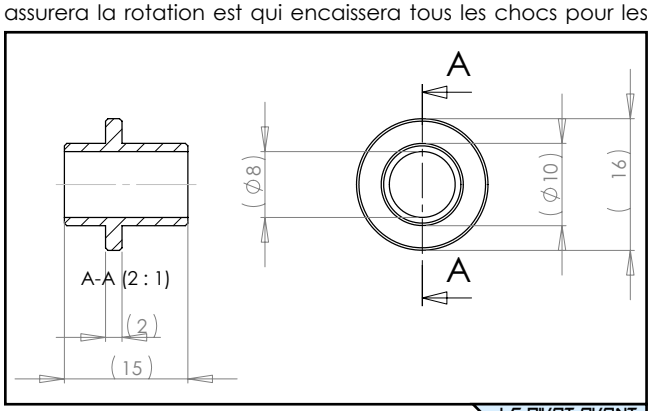
LA CHAPE DE LA ROUE AVANT

du créer une pièce contenant la roue et pouvant pivoter. C'est une sorte de chape qui contient la roue avant et son axe. Elle est aussi en aluminium et nous l'avons usiné en forme de U afin de créer un logement pour la roue et pour un capteur de vitesse. Nous avons ensuite enlevé une partie de la pièce afin de la rendre plus esthétique, puis nous avons créé trois alésages : deux d'entre eux sont destinés à contenir des roulements pour l'axe de la roue, ils ont un diamètre de 9,98 mm. Le dernier alésage sert à faire pivoter la roue. La pièce est en effet assemblé avec un autre élément décrit ci-après. Le trou fait 9,95 mm de diamètre.

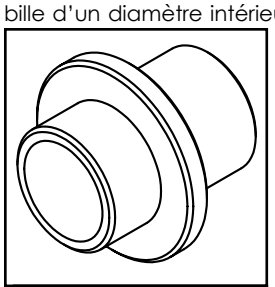

LA CHAPE EN PERSPECTIVE

Le pivot

L'élément destiné à être assemblé à la pièce précédente est un pion cylindrique. Il est muni d'un épaulement dans l'optique de le renforcer car c'est lui qui assurera la rotation est qui encaissera tous les chocs pour les transmettre ensuite aux suspensions. Une extrémité de la pièce est emmanchée dans la chape de la roue, tandis que l'autre est montée en force dans un roulement à bille d'un diamètre intérieur de 10 mm. Ce pion est creux car il doit laisser passer les fils de connexion des différents capteurs. La roue est fixée à la chape au moyen d'un arbre de 3 mm de diamètre traversant la roue ainsi qu'un petit disque muni de fente régulièrement espacées jouant le rôle, avec une fourche optoélectronique, de capteur de vitesse.

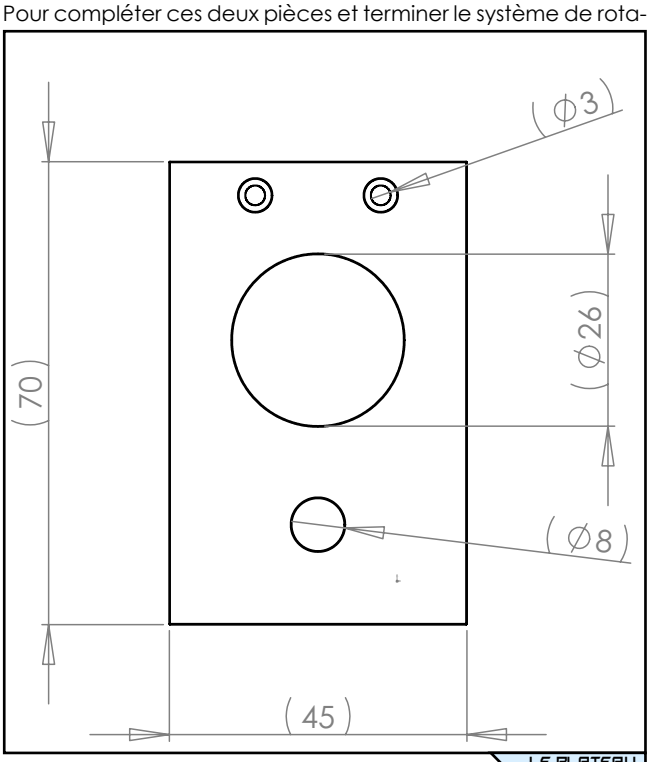


LE PIVOT AVANT

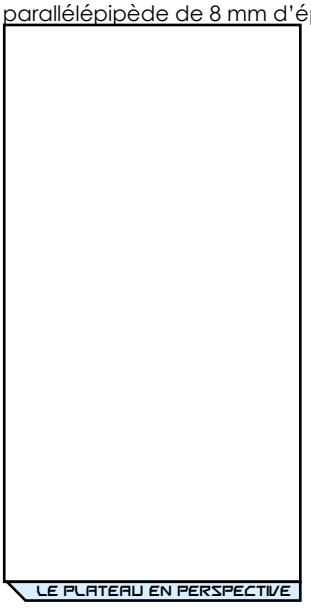

LE PIVOT EN PERSPECTIVE

Le plateau

Pour compléter ces deux pièces et terminer le système de rotation, nous avons créer un plateau pouvant contenir un roulement à bille de 25,98 mm de diamètre. C'est un parallélépipède de 8 mm d'épaisseur alésé d'un trou de 25,94 mm de diamètre. Cette opération est assez compliqué et fastidieuse car il est difficile de trouver un outil tel qu'un foret ou un fraise de 25,9 mm de diamètre. Il faut pour cela usiner la pièce sur un tour en la montant au préalable sur un plateau, ce qui est très long car il faut centrer parfaitement la pièce afin de respecter les bonnes dimensions. Ainsi insérer dans le plateau, le pion cylindrique et donc la chape sont rentrer en serrage dans le roulement. Le plateau est aussi percer de trois trous, deux de 3 mm de diamètre dans l'optique de fixer une attache d'amortisseur, et un de 8 mm de diamètre pour permettre l'emmanche d'un cylindre de guidage.

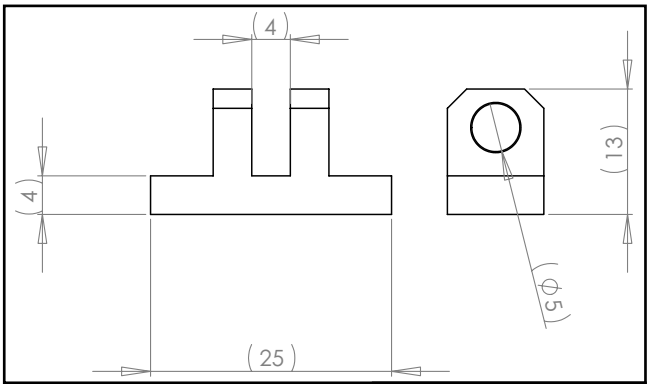


LE PLATEAU


LE PLATEAU EN PERSPECTIVE

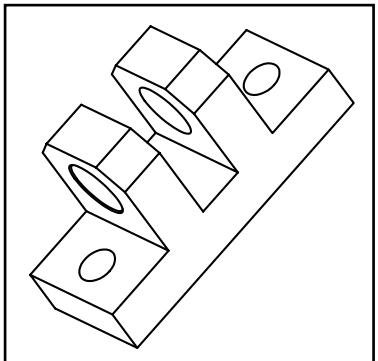
Attache d'amortisseur

Un autre élément a été spécialement conçu pour rattacher les suspensions au châssis et au reste de la voiture. C'est une fixation en aluminium réalisée à partir d'une barre que l'on a usiné puis tronçonnée pour créer huit pièces similaires. Elle est



L'ATTACHE D'AMORTISSEUR

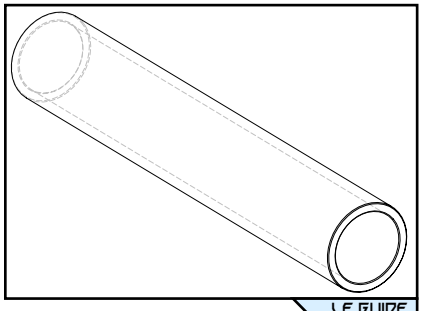
principalement composée d'une gorge destiné à la tête de l'amortisseur et d'un alésage de 5 mm de diamètre. Deux trous de 3 mm de diamètre ont été percés pour permettre la fixation. Le maintien en position de la suspension et de la fixation est réalisé au moyen d'un petit axe. Le blocage en translation de l'axe est effectué d'un coté grâce à la tête de l'axe et de l'autre grâce à une goupille épingle fabriquée avec de la corde à piano. Cette pièce joue le rôle de liaison pivot car le débattement de l'amortisseur créé une légère rotation.



L'ATTACHE EN PERSPECTIVE

Le guide en translation

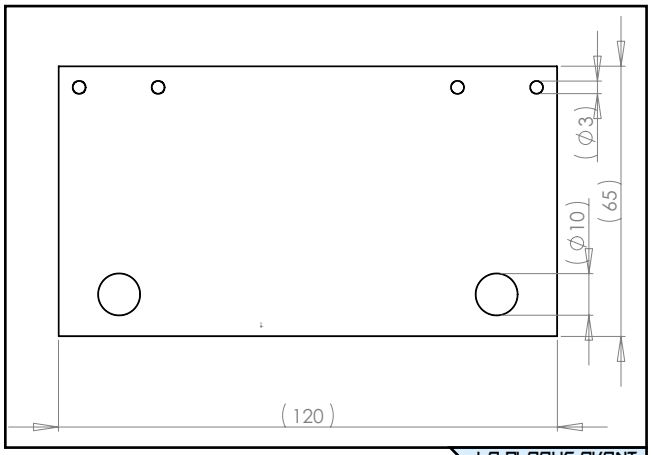
La suspension ne jouant que le rôle d'une liaison pivot glissant, il a fallu supprimer un degrés de liberté : la rotation autour de la fourche de l'amortisseur. Pour cela nous avons créé un guide composer d'un cylindre d'aluminium de 8 mm de diamètre et d'un tube en cuivre de 10 mm de diamètre extérieur et de 8 mm de diamètre intérieur. Le but étant aussi de renforcer cette partie du montage. Le cylindre peut donc coulisser librement à l'intérieur du tube. Nous avons choisis du cuivre car d'après une règle simple, deux métaux de couleurs différentes sont moins exposées aux frottements que deux métaux en contact de même aspect.



LE GUIDE

La plaque avant

Le guide est d'un coté assemblé au plateau contenant le roulement et de l'autre à une plaque en aluminium de 5 mm d'épaisseur accueillant l'autre partie du guide ainsi qu'une autre attache d'amortisseur. Les trous de la fixation de la suspension et celui du tube sont en double sur cette pièce car elle permet d'assembler la roue avant droite et la roue avant gauche. Cette plaque est ensuite assembler à la partie avant du châssis.



LA PLAQUE AVANT

L'ÉLECTRONIQUE

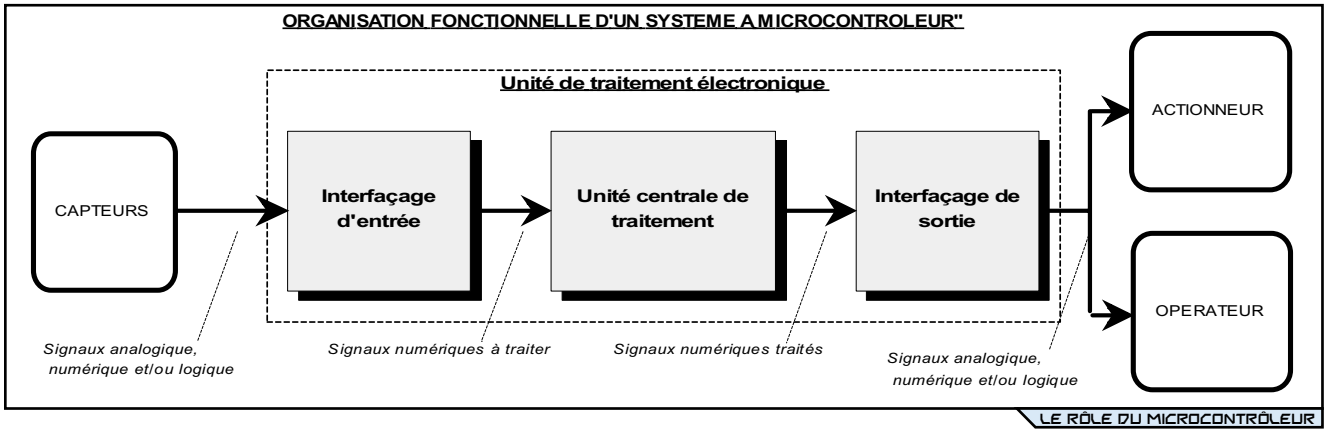
Nicolas Viennot

...:: INTRODUCTION ::...

La base du montage

Nous avons tout d'abord fait fausse route en essayant de traiter le problème de la commande des moteurs, de la transmission radio, analogiquement. Puis nous avons découvert la puissance de l'électronique numérique (autre que de quelques portes NAND en série). En effet, l'utilisation de microcontrôleurs est une obligation pour les projets de robotiques, domotiques etc... Le circuit électronique devient simple, et ainsi, la facture s'allège. On arrive à une miniaturisation très poussée. Le circuit est beaucoup plus fiable car les caracéristiques des composants analogiques changent au cours du temps et de la température. En bref, l'électronique numérique commence véritablement à remplacer tous les circuits analogiques (chaine, radio, etc...). Ainsi nous avons du faire face au choix des microcontrôleurs, à leur programmation, à la réalisation des schémas et à la réalisation des circuits imprimés.

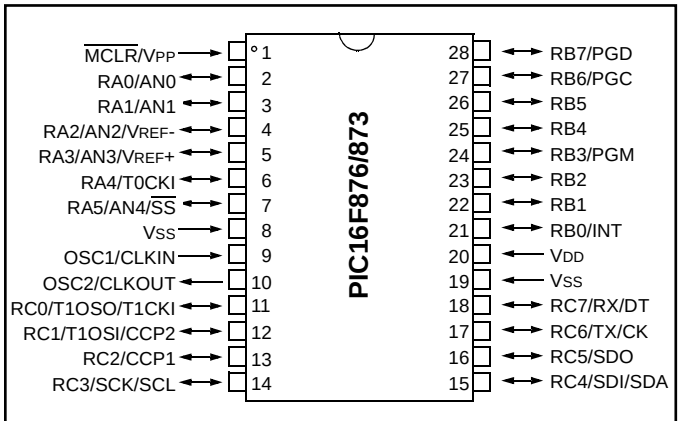
...:: LE MICROCONTRÔLEUR ::...



LE RÔLE DU MICROCONTRÔLEUR

Le choix du microcontroleur

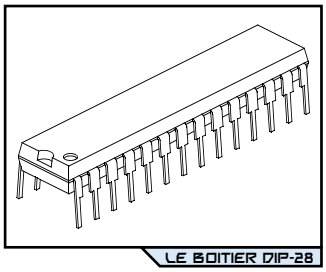
On distingue 2 familles de microcontôleurs : Les CISC (Complex Instruction Set Computer), le CPU (Central Processing Unit) possède un grand nombre d'instructions, chacune d'elles s'exécute en plusieurs périodes d'horloges. Et les RISC (Reduced Instruction Set Computer), le CPU possède un nombre réduit d'instructions ainsi la vitesse d'exécution est bien plus rapide qu'un CISC, en effet le décodage d'une instruction se fait plus rapidement puisque cette dernière est codée sur un faible nombre de bits. On peut choisir entre les DSPs, les classiques 68HC11 , et les PICs. Les DSP sont des microcontrôleurs très spécifiques dans le traitement du signal. Ce n'est pas un choix adapté à notre projet. Les 68HC11 de Motorola appartiennent à la famille CISC, ils sont puissants mais lents. Ce microcontrôleur n'est pas adapté puisque nous n'avons pas réellement besoin de sa capacité de calcul. Les PIC de Microchip (Programmable Intelligent Controller) appartiennent à la famille RISC, ils sont très performants. Le PIC est donc un microcontôleur adapté à notre projet.



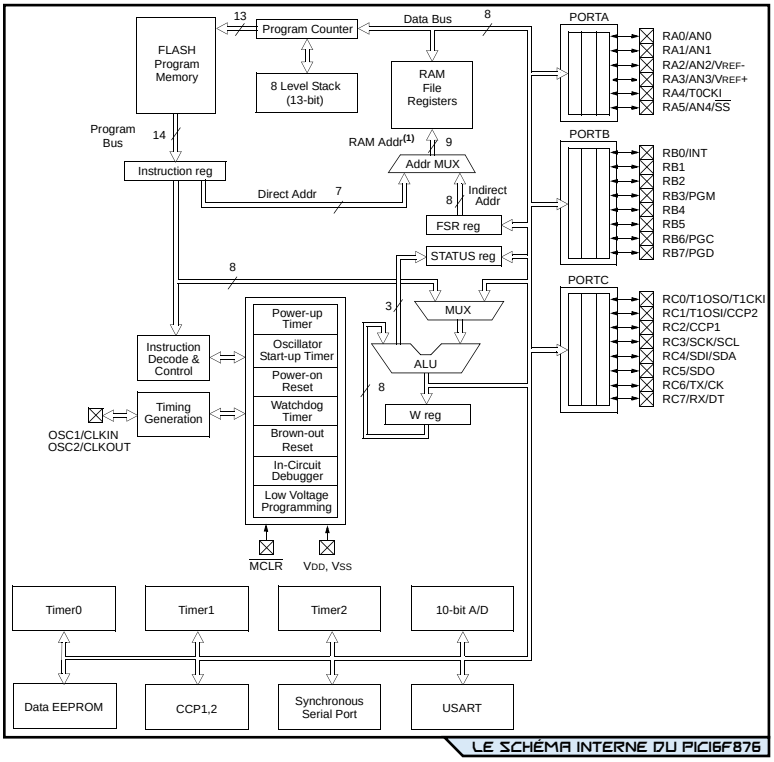
LE BROCHAGE DU PIC16F876

Le PIC16F876

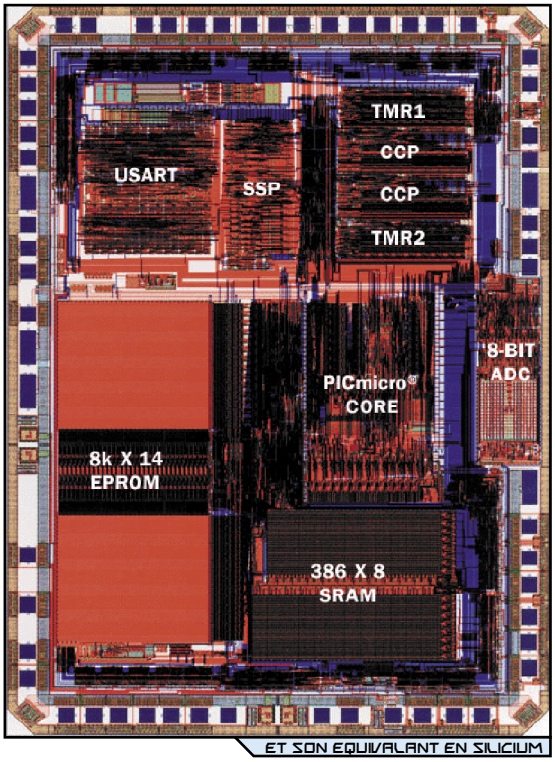
On utilisera le PIC16F876 de la gamme Mid-Range (8 bits). Il comporte un jeu de 35 instructions, et possède une vitesse d'exécution de 200ns par instruction pour une horloge fonctionnant à 20MHz. Les autres fonctionnalitées seront traitées dans la partie programmation. Ce microcontrôleur est flashable, c'est à dire qu'on peut le programmer autant de fois que l'on veut, et non comme les OTP (One Time Programmable) que l'on utilise uniquement dans la fabrication de masse, ce qui sera très utile lors du prototypage. Il possède 8Ko de ROM (le programme ne pourra donc pas éxeder 8192 instructions), 368 octets de RAM et 256 octets de EEPROM (non utilisés pour notre application). Autres avantages des PICs : ils sont peu onéreux, très robustes, et bien sûr très fiables, en bref c'est un outil très performant dans l'électronique numérique et les systèmes embarqués. Le PIC16F876-04/SP coûte 7.90€ à l'unité chez Farnell (www.farnell.com). Il est disponible en boitier. On le choisit en boitier DIP-28 et on prend la version 4MHz même si on l'utilise à 20MHz (les PICs sont overclockables même si la documentation technique en prétend le contraire c'est à dire que l'on peut 'booster' la vitesse d'exécution du microcontrôleur). Cette version supporte des températures allant de 0° à 75°, si on préfère avoir une plus grande marge, on choisira la version I (Industrial) voir E (Extended) mais nous choisirons la version commerciale. Leur environement de travail pour la programmation est gratuit et convivial. Ce type de microcontrôleur est extremement utilisé dans toute sortes d'applications.



LE BOITIER DIP-28



LE SCHÉMA INTERNE DU PIC16F876

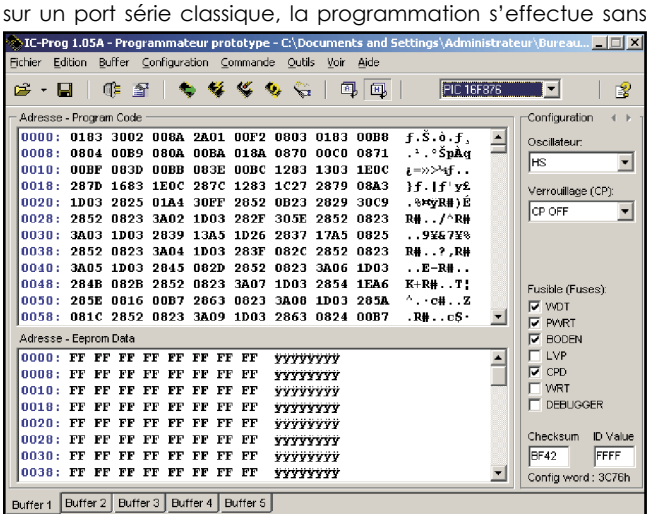


ET SON EQUIVALANT EN SILICIUM

...:: LE PROGRAMMATEUR ::...

Le programmateur standard

Pour implanter un programme dans la mémoire flash, il faut un programmateur, le transfère des données se fait de manière serielle synchrone sur les pins RB6 (Horloge) et RB7 (Donnée) du microcontrôleur. Le programmateur proposé par électronique pratique (magazine n°270) fonctionne sur un port série classique. Pour piloter le programmateur de PIC, on utlise IC-Prog (www.ic-prog.com), qui est un logiciel gratuit de très bonne qualitée. Il faut le configurer en programmateur JDM, sur un port série classique, la programmation s'effectue sans

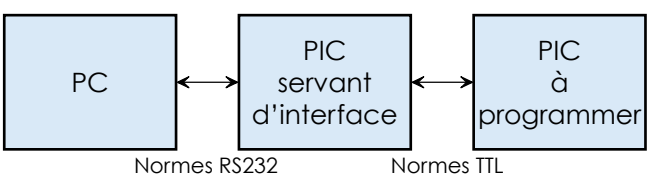


L'INTERFACE CONVIVIALE DE ICPROG

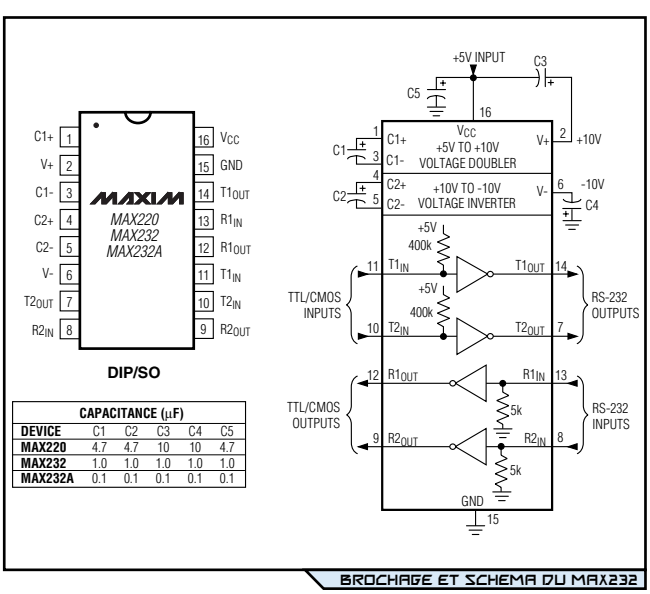
problèmes. Or mon ordinateur portable ne possède pas de port série, donc j'ai du faire l'achat d'un convertisseur USB/Série. Avec ce nouveau port, la programmation ne marche pas (il s'agit en fait d'un bug du microcontrôleur), en effet la vitesse de commutation des pins RTS et DTS étant trop lente, le microcontrôleur décroche. C'est ainsi que l'utilisation d'un programmateur de PICs utilisant les pins Tx et Rx du port série s'avère être obligatoire. En effet, les pins Rx et Tx sont spécifiques pour l'émission de donnée.

Le programmateur alternatif

Pour utiliser les pins Rx et Tx, il faut intercaler un microcontrôleur qui va se charger de convertir les données du PC pour les envoyer au PIC à programmer. On va utiliser pour cela un autre PIC16F876 même si un PIC16F84A moins cher aurait suffit.



Le port série du PC obéit aux normes RS232, le niveau logique 0 correspond à une tension comprise entre +3V et +25V (habituellement 12V) et le niveau logique 1 correspond à une tension comprise entre -3V et -25V. Pour convertir le RS232 en TTL (logique 0V 5V) on utilise le MAX232.



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

BROCHAGE ET SCHEMA DU MAX232

Pour programmer les PICs, il faut également une tension de 13V sur la pin MCLR, or on veut que l'alimentation se fasse du port USB (5V - 500mA) pour une portabilité maximale. Donc il faut créer la tension de programmation artificiellement, j'utilise pour cela un TL497A. En prenant les désignations du schéma de Texas Instrument, V_I = 5V, R_{CL} = 10Ω, L = 100µH, C_T = 100pF, C_O = 47µF, R_1 est une résistance variable de 100K. Il faut régler R1 pour obtenir V_O = 13.3V.

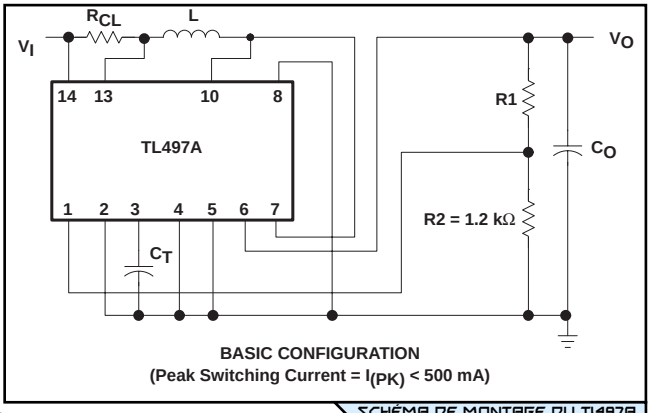
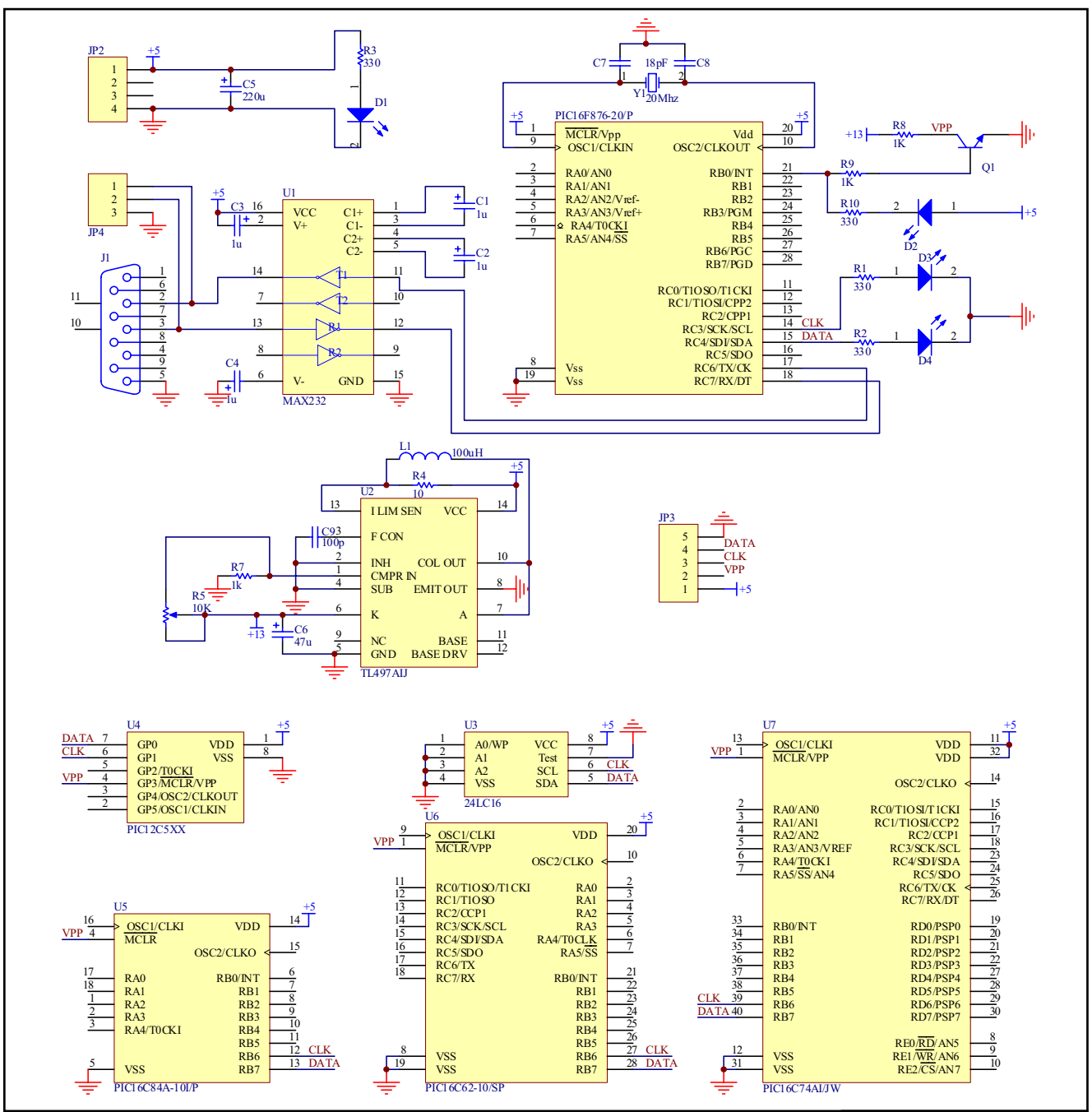


SCHÉMA DE MONTAGE DU TL497A

Explications du schema

Le programmateur peut programmer des PICs à 8, 16, 28, 40 broches, ainsi que des EEPROMs 24LCxx. JP3 est un connecteur pour une éventuelle carte fille (si on veut programmer des smartcard, etc...). JP4 est un connecteur pour le port série en support 3 broches, ce qui est très pratique lors du prototypage puisque je branche alternativement le port série au programmateur et à la platine d'expérimentation. JP2 est le connecteur pour l'alimentation (USB). Y1 est le quartz qui va cadanser le PIC à 20MHz avec ses condensateurs de 18pF (il faut en choisir entre 15pF et 30pF), on aurait pu utiliser un résonateur qui inclue les 3 en 1. Q1 sert à piloter le 13V par une fonction inverseuse: si RB0 = 0V, Q1 est bloqué, donc VPP = 13V (R8 consomme un peu, c'est pour cela que l'on règle le TL497A à 13.3V) ; si RB0 = 5V, Q1 est passant, donc V_{CE} = 0V et donc VPP = 0V. D1, D2, D3, D4 sont les leds de visualisation d'état, leur résistances séries sont de 330Ω, c'est suffisant pour la luminosité. De plus chaque pin du PIC ne doit pas débiter plus de 20mA, la règle est respectée Attention tout de même avec la résistance de reglage du 13V, je dis ceci en connaissance de causes : par inadvertance, le PIC a reçu 60V par la pin MCLR, ce qui l'a immédiatement détruit.

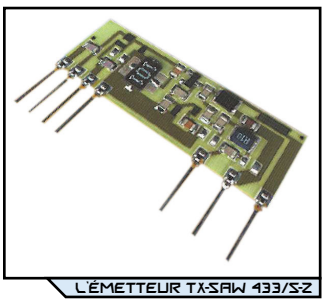


SCHEMA DU PROGRAMMATEUR

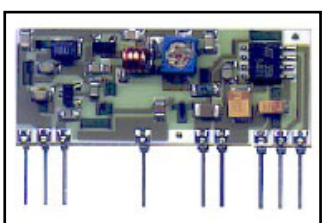
...:: LA TÉLÉCOMMANDE ::...

Description de la transmission

Pour acheminer les données vers la voiture, on utilise les modules radio hybrides Aurel (l'émetteur: TX-SAW 433/s-Z et le récepteur: RF 290 A-5S). La transmission doit se faire dans les 2 sens, pour permettre des retours d'informations tels que la vitesse. Ce sont des modules radio qui fonctionnent en AM (Amplitude Modulation) à la fréquence de 433.92MHz, on les utilise à 4800bps (ce qui est beaucoup pour ces modules mais presque trop faible pour notre application que nous verrons en détails dans la partie programmation). On pourra utiliser le transceiver (module d'emission et réception) XTR-869 fonctionnant en FM (Frequence Modulation) à 869.95 MHz (bande moins polluée que la 433 puisqu'elle n'a été ouverte que récement en Europe), on pourra l'utiliser à 115200bps ce qui donne de plus grandes possibilités pour la gestion en temps-réel de la voiture. La portée de la transmission n'a pas encore été mesurée (elle depend énormement de l'accordage de l'antenne, etc...) mais elle devrait être comprise à priori entre 50m et 100m. Puisque l'émeteur et le récepteur fonctionnent sur la même fréquence, on ne peut les utiliser en même temps car sinon il se brouilleraient entre eux. On parle alors de transmission half-duplex, c'est à dire que les 2 interlocuteurs peuvent parler mais pas en même temps. L'utilisation de 2 antennes séparées est obligatoire si l'on ne veut pas utiliser de relais spécifique (ensuite il faut piloter le relais par le port série etc...) sinon la puissance serait diffusée et donc la portée serait réduite.

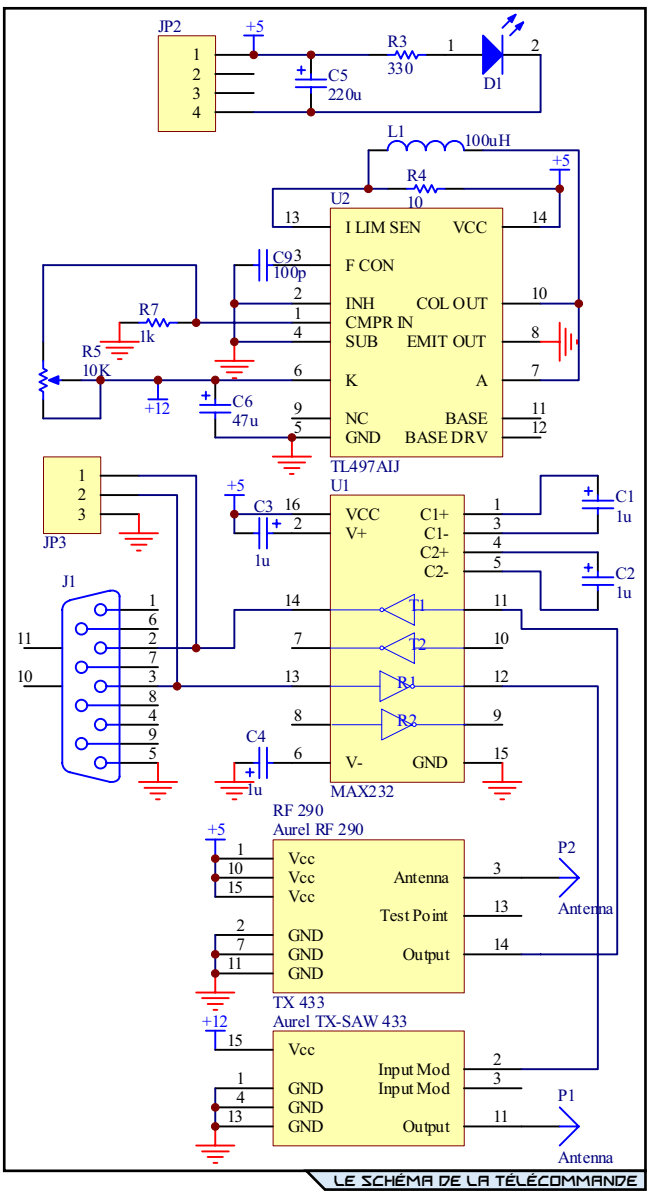


L'ÉMETTEUR TX-SAW 433/S-Z



LE RÉCEPETEUR RF 290 A-5S

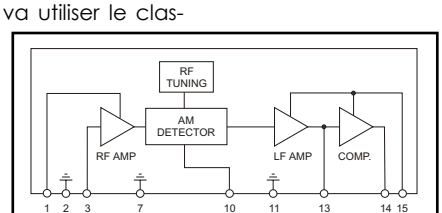
LE TRANSEIVER XTR-869



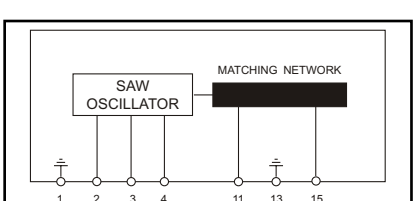
LE SCHÉMA DE LA TÉLÉCOMMANDE

Description des composants

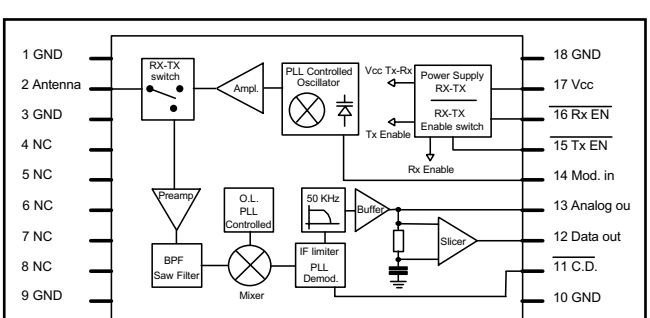
Pour convertir les signaux du port série, on va utiliser le classique MAX232, les modules radios Aurel TX-SAW 433/s-Z et RF 290 A-5S. L'émetteur est alimenté par une tension comprise entre +5V et +12V, on va donc utiliser le TL497A puisque la télécommande tirera son alimentation à partir du port USB pour une portabilité maximale. Le schéma fonctionne très similairement au programmateur de PIC, le réglage de la tension de sortie du TL497A doit par contre se faire à 12V (pas plus !). On aurait pu mettre des condensateurs de découplage sur les modules radios (très sensibles) de l'ordre de 100nF pour absorber les pics d'intensité des circuits integrés mais la version finalisée de la télécommande sera avec le transeiver XTR-869, qui nécessite énormement de précaution comme l'électricité statique de l'antenne qui doit être évacuée par une bobine vers la masse au lieu de pénetrer dans le module. Il y a encore bien d'autres astuces que je ne vérrais que expérimentalement avec un analyseur de spectres et autres. Le débuggage des circuit HF est très difficile puisque les facteurs de fonctionnement sont très variés comme la proximité des composants, ou encore les masses environnantes (la batterie au plomb va perturber énormement la transmission voire la bloquer totalement).



LE SCHÉMA INTERNE DU RF 290 A-5S



LE SCHÉMA INTERNE DU TX-SAW 433



LE SCHÉMA INTERNE DU XTR-869

...:: LA VOITURE ::...

Description de l'environnement

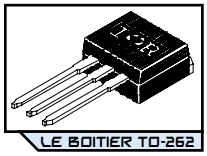
La partie électronique de la voiture se compose de deux parties, une logique et une puissance. Ces deux parties sont distinctes, c'est à dire que l'on sépare les deux cartes électroniques, chacune d'elles possède sa propre alimentation pour éviter les éventuelles perturbations des moteurs. La partie logique s'alimente avec deux piles de 9 volts et la partie puissance avec la batterie au plomb de 12V. La voiture est équipée de 2 moteurs GZ-600 BB à courant continu de tension nominale 9.7V. Les capteurs de vitesse sont des fourches HoneyWell (optocoupeur avec un disque à fentes entre l'émetteur infra-rouge et le phototransistor).

UN MOTEUR GZ-600 BB

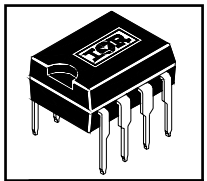
Etude de la partie puissance

La courbe de charge des moteurs est établie sur une tension de 9.6V donc n'est pas vraiment adapté à notre application puisqu'on les sur-alimente à 12V. La courbe indique que si on désir un couple important, l'intensité absorbée peut aller jusqu'à 40A par moteur, donc 80A maximum pour les 2 (ce que la batterie peut fournir) mais à priori, le point de fonctionnement d'un moteur devrait se situer au rendement maximal, et donc la consommation nominale est en dessous de 10A. L'étude de la consommation est déterminante pour le choix de nos composants. L'inversion du sens de rotation des moteurs se fait traditionnellement avec un pont en H (montage à 4 transistors) mais nous allons la faire avec des relais 2RT (relais à 2 contacts inverseurs). On a choisi les relais Finder 44.62 qui supportent 10A sans problèmes, pour les piloter il faut mettre un transistor qui pilote la bobine car elle consomme 100mA et le PIC ne peut en délivrer que 20. La variation de vitesse pour un moteur à courant continu se fait par modulation de largeur d'impultions (PMW) que l'on verra plus en détails dans la partie programmation. Le PIC16F876 possède 2 modules PWM et donc on ne se servira pas de contrôleurs externes. Au début, Le pilotage des moteurs se faisait par des transistors de puissance darlington NPN, mais avec leur Vce qui avoisinent 1.5V, la perte est $V_{ce} * I_n$ = 1.5 * 8 = 12W ce qui est la puissance de mon fer à souder. Ces transistors ont été évidement détruits lors de l'experimentation de la carte puissance. C'est pourquoi on utilise des transistors MOS qui génerent moins de pertes que les bi-polaires. On a choisit les IRL1004L de International Rectifier à canal N. Ils sont de la 5ème génération de transistors HEXFETs avec une résistance $R_{ds(on)}$ = 0.0065Ω et une intensité maximale en charge de 110A, ce qui donne une dissipation nominale en charge de $R_{ds(on)} * I_n^2$ = 0.0065 * 64 = 0.5W (c'est bien mieux). Ces transistors sont vendus dans un boitier TO-262. Pour piloter ces transistors, on utilisera une tension de 12V au lieu de 5V (tension délivrée par le PIC) car d'après la courbe ci-dessous le courant passe mieux avec une tension V_{GS} élevée donc il y aura moins de pertes. Il faut donc utiliser un driver MOS, j'ai choisis le IR4427 dans son boitier DIP-8, il peut piloter 2 transisors MOS. Pour freiner, on utilisera des résistances de 0.5Ω, mais on testera si l'utilisation d'un simple fil fera l'affaire, en effet, le freinage s'effectue en cours-circuitant le moteur, celui-ci fonctionnera comme une géneratrice et convertira toute l'énergie d'inertie en chaleur. Pour piloter le freinage, on utilisera également un relais sans oublier les diodes de roue libres qui protègent les transistors.

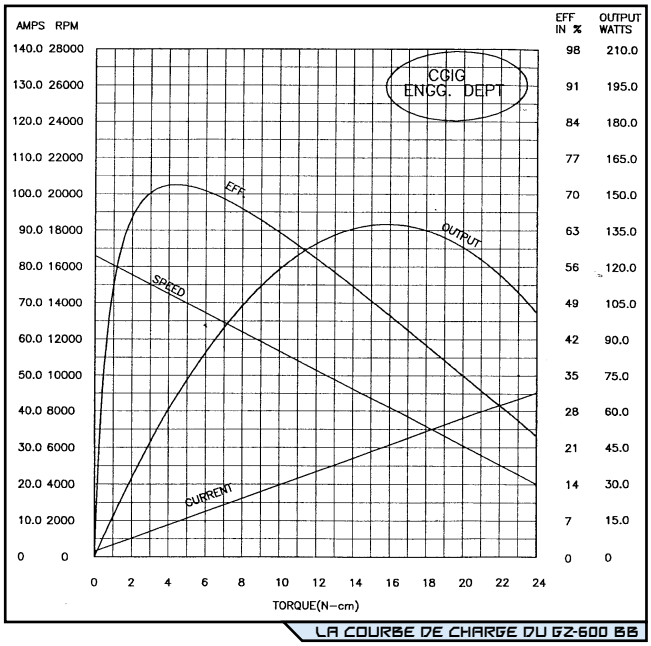
UN FINDER 44.62



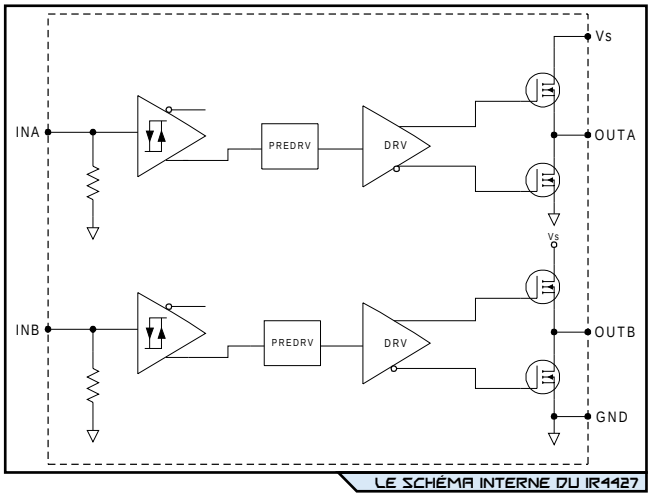
LE BOITIER TO-262



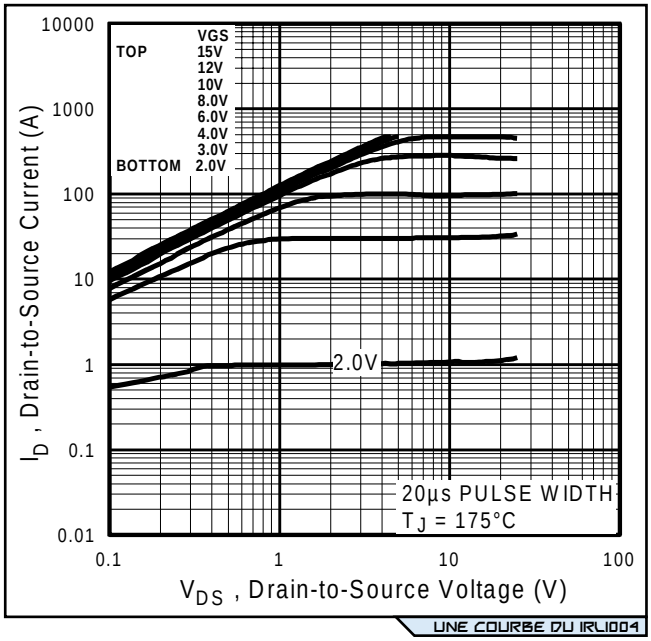
LE BOITIER DIP-8



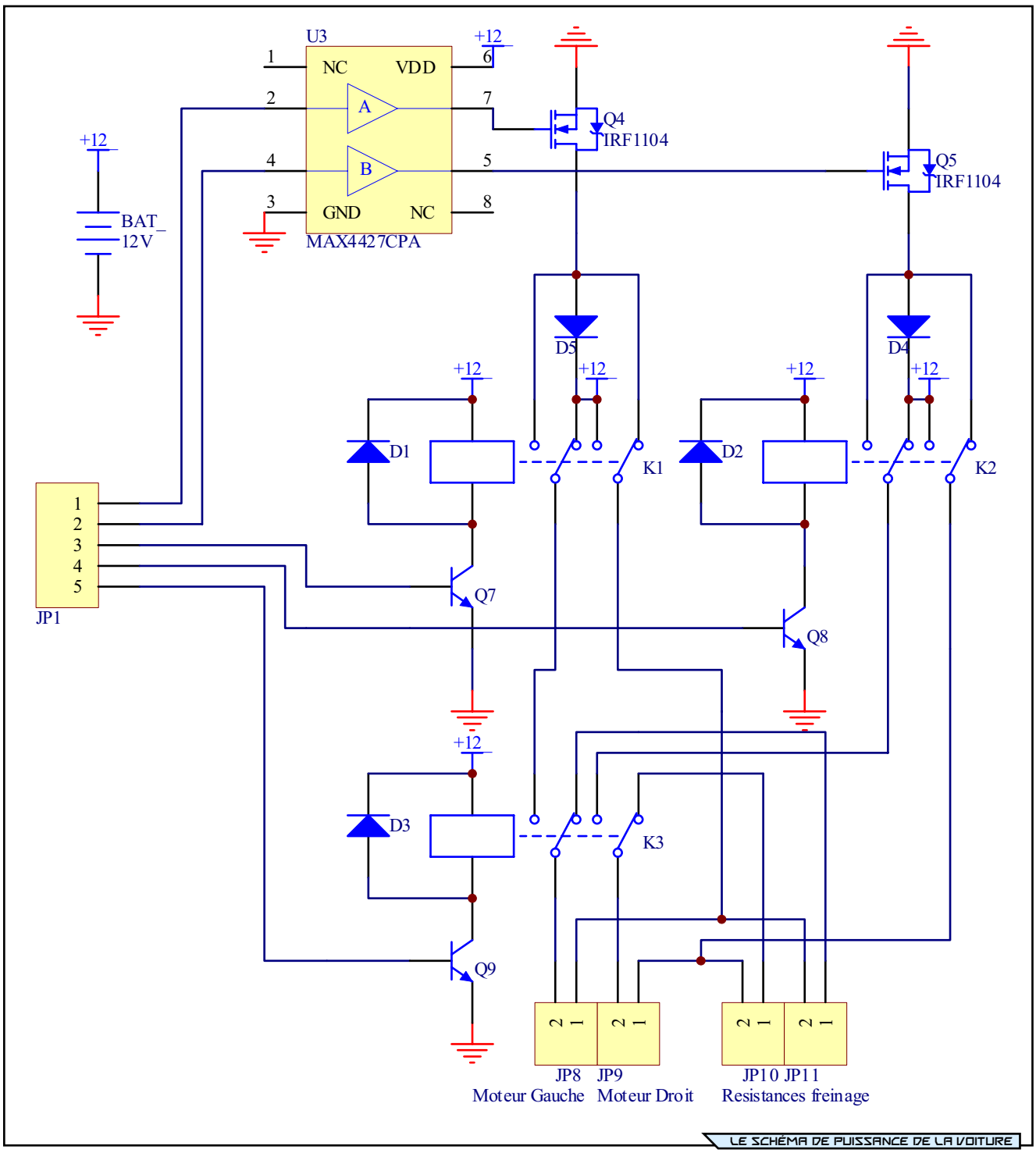
LA COURBE DE CHARGE DU GZ-600 BB



LE SCHÉMA INTERNE DU IR4427



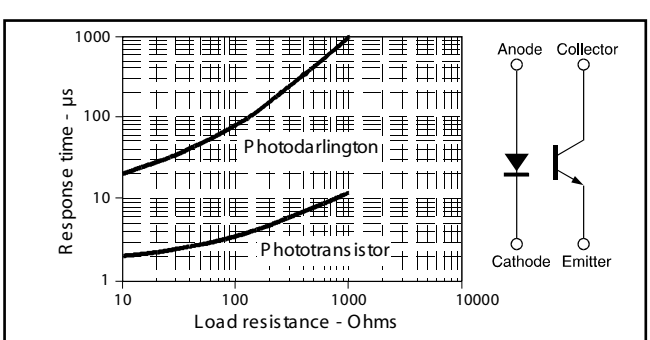
UNE COURBE DU IRL1004



LE SCHÉMA DE PUISSANCE DE LA VOITURE

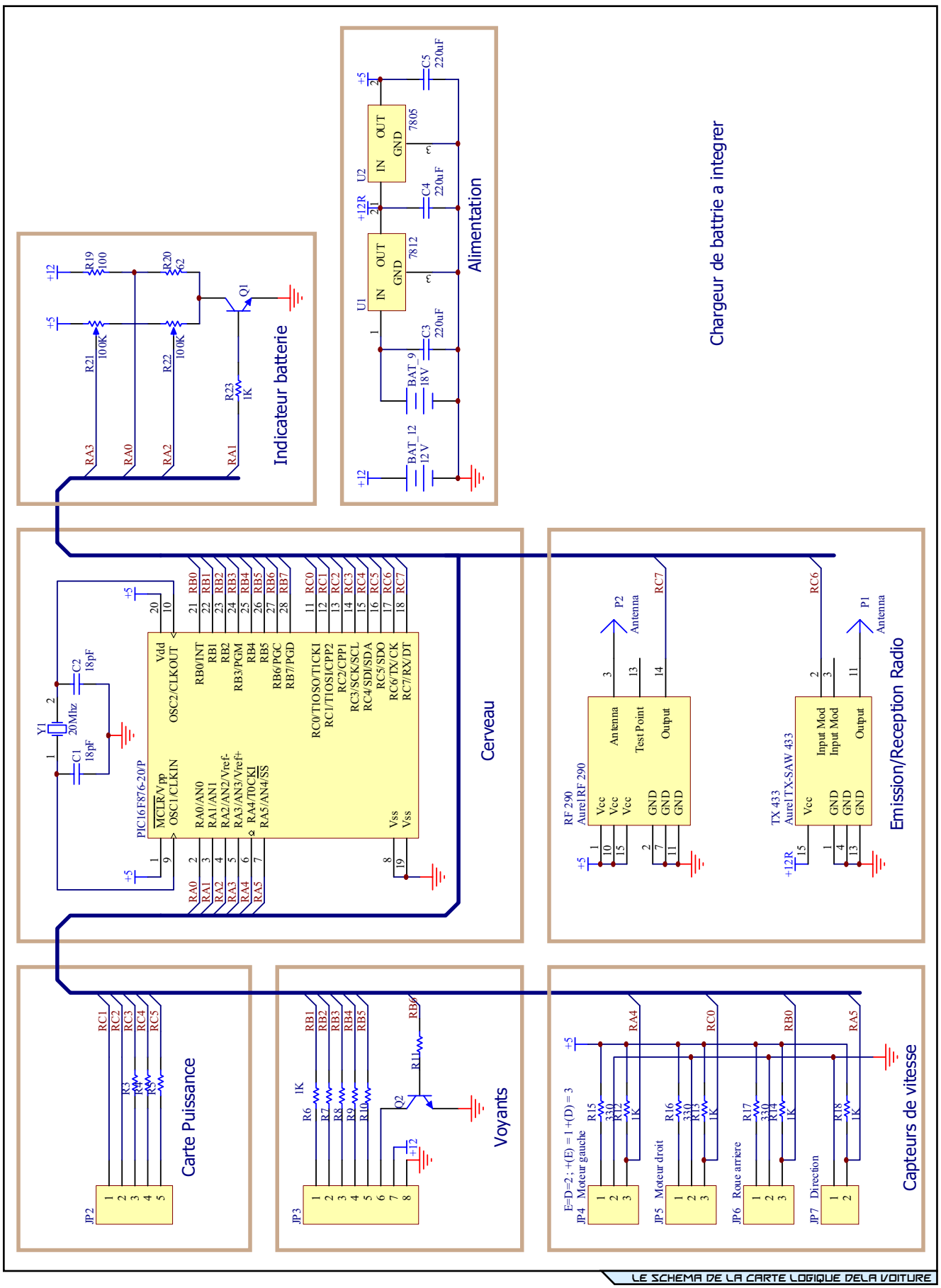
Etude de la partie logique

L'alimentation de la partie logique se fait en 18V, donc il faut employer des étages de régulation pour obtenir le 12V pour les modules Aurel, et le 5V pour le PIC. On utilise les régulateurs 7812 et 7805 avec des condensateurs de 220µF pour stabiliser la tension. On aurait pu mettre des condensateurs de découplagessur les modules Aurel et sur le PIC de 100nF mais le schéma n'est pas une version finalisée (il manque le chargeur de batterie). Pour charger une batterie, il faut l'alimenter avec un courant continu de 10% de sa capacitée, notre batterie faisant 7Ah, il faut lui injecter 700mA pendant une 12ène d'heures, mais attention à ne pas trop depasser la durée, sinon la batterie risquerait d'exploser. Pour faire un générateur d'intensité constante, on utilise un transistor NPN en régime d'amplification, en lui fixant un courant dans sa base et en utilisant une diode (une LED par exemple) avec une résistance variable en série, et donc le courant du collecteur sera constant (et réglable avec la résistance). Puisqu'il fonctionne en amplificateur, la puissance dissipée est alors importante donc il faut munir le transistor d'un radiateur. On pourra également utiliser un circuit intégré qui s'occupera intégralement de la charge. Les ré-



LA FOURCHE PHOTOTRANSISTOR

cepteurs infra-rouge ont un temps de réponse qui dépend de leur résistances de tirage. On a choisi des 1KΩ pour obtenir un temps de réaction avoisinant les 10µs, ce qui semble suffisant. La vitesse de la voiture sera fonction de la fréquence du signal reçu par le PIC (nous aborderons les calculs relatifs aux capteurs de vitesses dans la partie programmation). Le PIC intègre un convertisseur analogique/numérique, on s'en servira pour réaliser un testeur de batterie. Sur le schéma, R19 et R20 consituent un diviseur de tension de rapport 1/4 basé sur la tension de la batterie. R21 et R22 sont des résistances variables pour régler les tensions de réference à partir du 5V fixe. Si la mesure se trouve erronée, on ajoutera une faible résistance pour forcer la

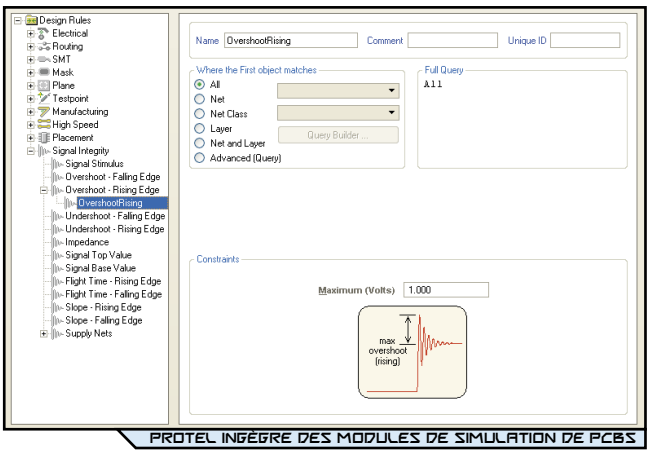


LE SCHEMA DE LA CARTE LOGIQUE DELA VOITURE

...:: LES CIRCUITS ÉLECTRONIQUES ::...

Fonctionnement de Protel

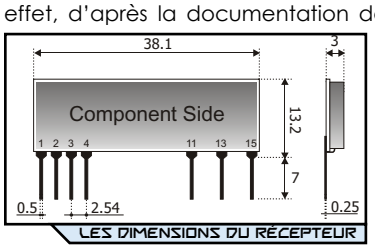
Pour concevoir les PCBs (Printed Circuit Board), on utilise Protel de la companie Altium (même logiciel que pour la capture des schémas). Il intègre un système très pratique de synchronisation des PCBs et des schémas. Il intègre de complexes modules de simulation pour les circuits haute-fréquence mais on ne s'en servira pas dans notre application. Il faut préciser que Protel est un logiciel pouvant router des circuit bien plus compliqués que le notre, en effet il peut router des cartes mères, gérer un circuit comportant plus de 32 couches, et en faire l'auto-routage 'intelligent', qui nous le verrons, est totalement inefficace pour notre circuit simple couche. Son environnement de travail est très convivial, l'interface est soignée. Le seul défaut de ce logiciel est le prix (8000$).



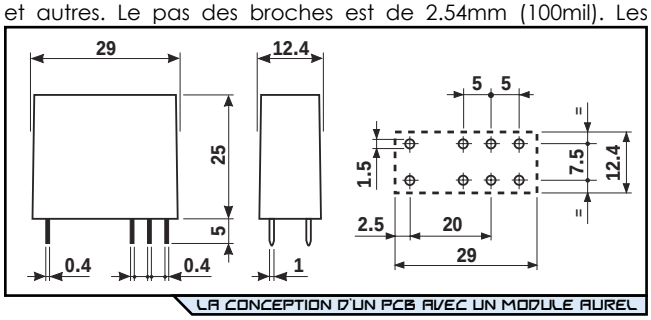
PROTEL INTÈGRE DES MODULES DE SIMULATION DE PCBS

Les modules Aurel sur un PCB

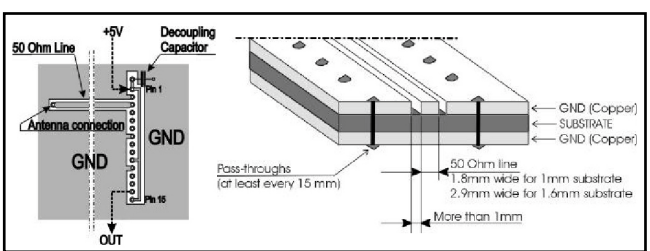
Le PCB supportant un module radio doit être très soigné. En effet, d'après la documentation de Aurel, il faut établir des plans de masse sous l'ensemble du module. Si le circuit était double face, il aurai fallut mettre un plan de masse sur la deuxième couche. Il faut également les garder à quelques centimètres des autres composants et le plus loin de tout circuit haute fréquence comme quartz et autres. Le pas des broches est de 2.54mm (100mil). Les dimensions des modules ont été pris en compte lors de la réalisation des librairies, de même pour les relais Finder.



LES DIMENSIONS DU RÉCEPTEUR



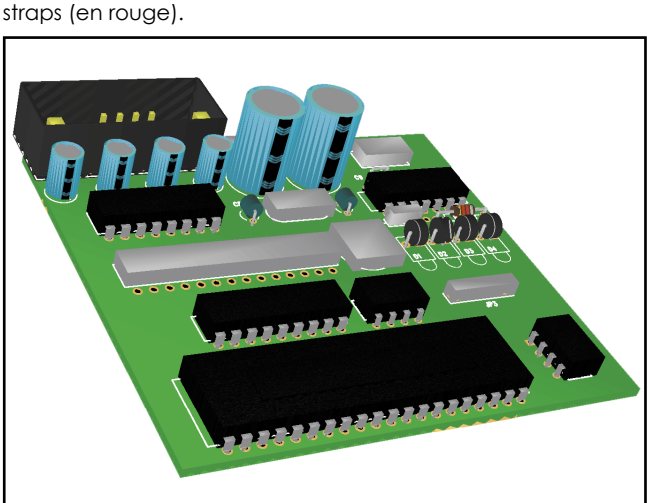
LA CONCEPTION D'UN PCB AVEC UN MODULE AUREL

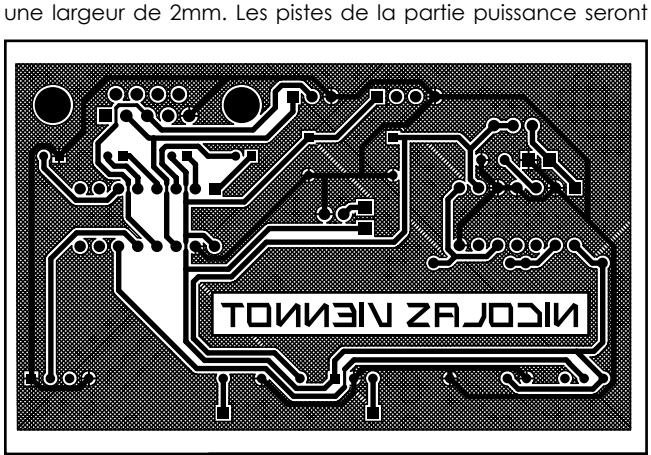


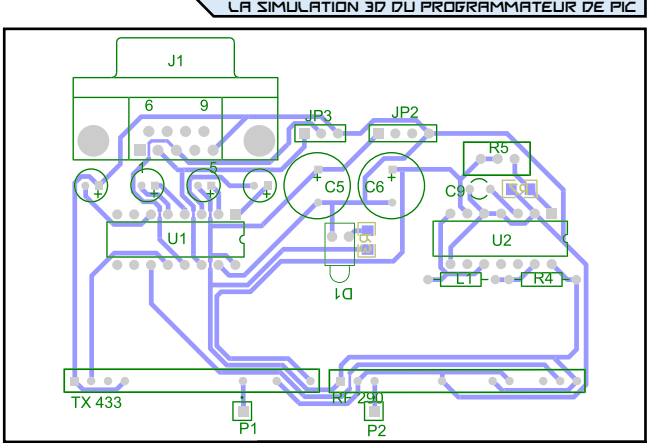
LA CONCEPTION D'UN PCB AVEC UN MODULE AUREL

Le routage

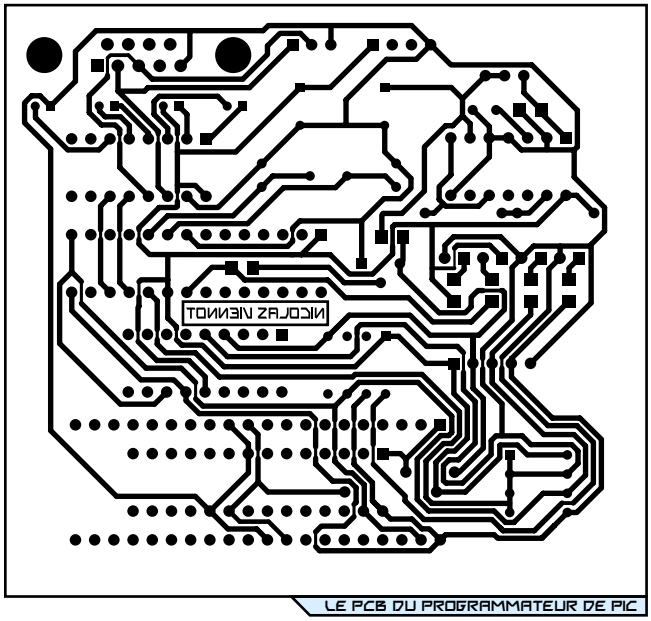
L'auto-routeur de Protel (nommé Situs) utilise des algorythmes très complexes mais surtout utile dans un PCB multi-couche comme le lissage de pistes, la gestion des vias (trou inter-couche), et bien d'autres encore. Pour un circuit comme le notre, il n'arrive même pas a router les 70% des pistes. Il a donc fallut faire un routage manuel sur feuille pour ensuite le porter dans le logiciel. Pour la partie logique des PCBs, on utilise une largeur de piste de 0.7mm et pour la partie puissance, on utilise une largeur de 2mm. Les pistes de la partie puissance seront étamée pour avoir une hauteur de 1mm afin d'apporter une résistance minimale pour ne pas échauffer le circuit lors de passage important d'intensité. L'espacement des pistes est de 0.3mm. Les PCBs ci-dessous sont à l'échelle 1:1. Pour le routage du programmateur de PIC, j'ai été forcé de faire deux petits straps (en rouge).


LA SIMULATION 3D DU PROGRAMMATEUR DE PIC

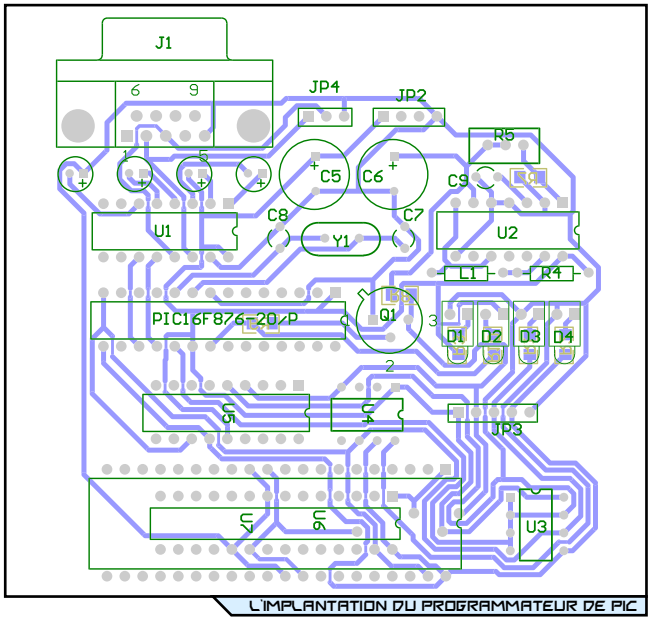

LE PCB DE LA TÉLÉCOMMANDE À L'ÉCHELLE 1:1



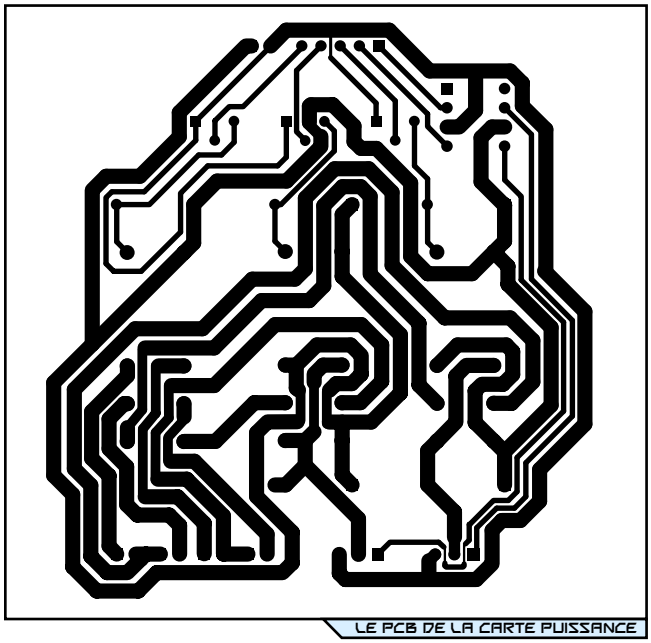
L'IMPLANTATION DE LA TÉLÉCOMMANDE



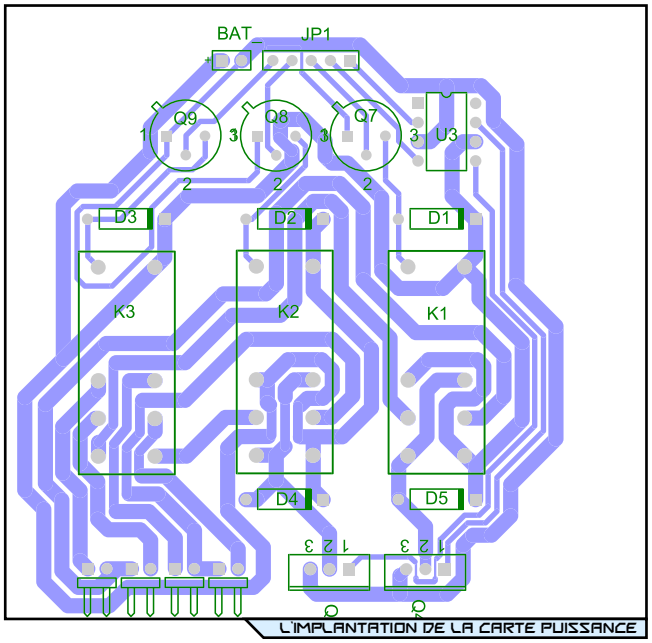
LE PCB DU PROGRAMMATEUR DE PIC



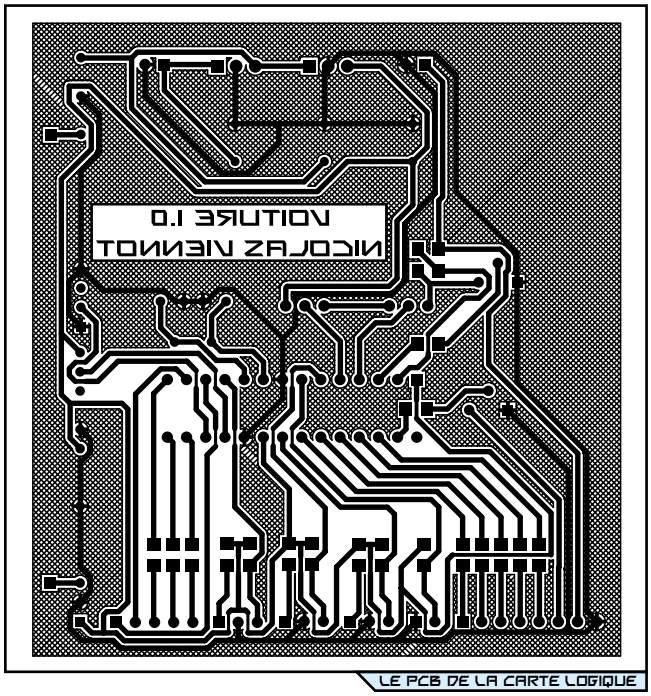
L'IMPLANTATION DU PROGRAMMATEUR DE PIC



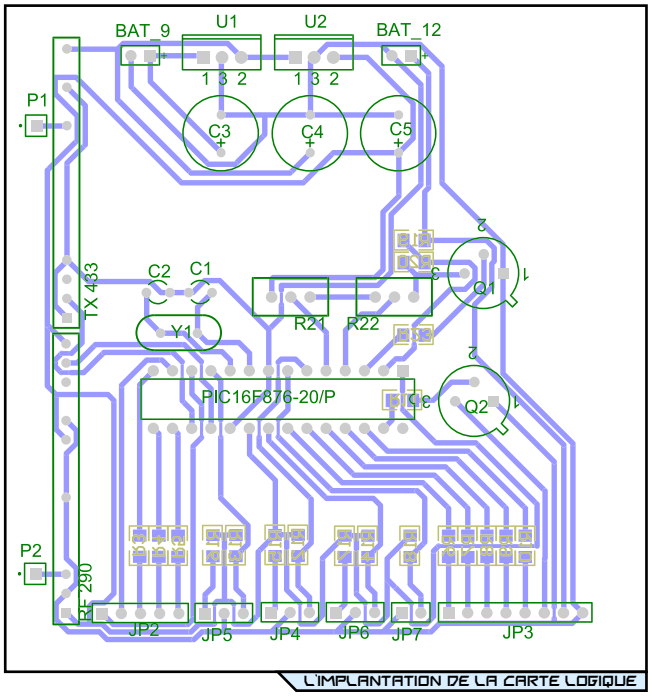
LE PCB DE LA CARTE PUISSANCE



L'IMPLANTATION DE LA CARTE PUISSANCE



LE PCB DE LA CARTE LOGIQUE

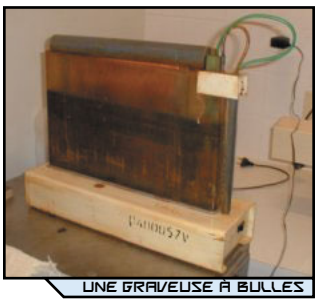


L'IMPLANTATION DE LA CARTE LOGIQUE

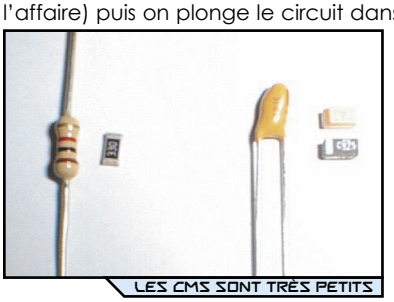
La realisation

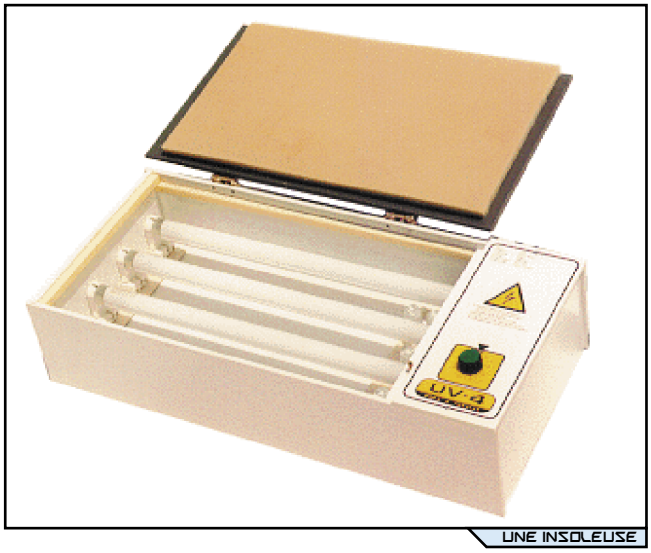
Pour réaliser un circuit imprimé, on passe généralement par 8 étapes. Tout d'abord, on imprime le typon sur un transparent (faire attention à l'échelle), puis on insole la plaquette cuivrée présensibilisée positive avec une insoleuse (on peut aussi le faire avec des lampes UV qui sont bien moins chères. On peut également transformer un vieux scanner en insoleuse avec des néon UV) et attention au sens du typon. Après l'insolation, il faut enlever le présensibilisé qui a été exposé aux UVs, pour cela on utilise du revelateur qui n'est rien d'autre que de la soude. On peut également utiliser du DESTOP (déboucheur pour les canalisations) qui une fois mélangé avec l'eau chauffera. Une fois le cuivre à enlever mis à nu, on procède à la gravue, traditionnellement fait avec du très salissant perchlorure de fer chaud. Ce procédé dure environ 5 minutes. On peut également utiliser de l'acide chloridrique et de l'eau oxygénée à 130 volumes. Les proportions sont : 1/3 d'eau, 1/3 d'acide et 1/3 d'eau oxygénée (à la louche). Une fois le mélange fait, il faut vite mettre la plaquette dans le bain, des bulles s'en échappent et environ 15 secondes plus tard, la gravure est terminée et c'est propre. Le mélange n'est plus utilisable et doit être jetté contrairement au perchlorure de fer qui peut servir pour plusieurs gravures. Bien sûr, il faut prendre un maximum de précautions avec ce genre de produits. Ensuite, il faut enlever le présensibilisé, pour cela on utilise de l'acétone ou de l'alcool à bruler. Ensuite, vient le perçage. Lorsque l'on pèrce avec des forêts HSS sur de l'époxy, ceux-ci s'usent très vite (environ 50 trous) et finissent par provoquer des cratères. Les forêts en carbure de tungstène sont adaptés et quasiment inusables mais ils sont très cassants (rien qu'en les faisant tomber par terre, ils cassent) et nécéssitent une vitesse de rotation de 30 000 tr/min. Il faut également faire des pastilles pleines (sans trous de centrage). Ensuite, pour avoir un circuit protégé contre les oxydations et facile à souder, on étame le circuit. Pour cela on passe du flux (ou du mirror fait l'affaire) puis on plonge le circuit dans un bain d'étain à froid. On peut d'ailleurs faire cela en appliquant de la pâte d'étain avec un pinceau en cuivre très chaud. Ou encore avec un fer (méthode employée avec la carte puissance : on a déposé très soigneusement une épaisseur de 1 à 2 mm d'étain sur les pistes ainsi la section est d'au moins 2mm²). Ensuite on vérifie le circuit à l'ohmmètre à la recherche de court-circuit et de micro coupures. Puis on soude les composants. Un petit fer à souder est obligatoire pour les composants CMS (Composants Montés en Surface).

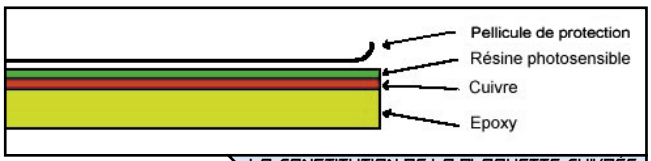

UN SCANNER RELOOKÉ EN INSOLEUSE


UNE GRAVEUSE À BULLES


UNE GRAVEUSE À JETS


LES CMS SONT TRÈS PETITS


UNE INSOLEUSE



LA CONSTITUTION DE LA PLAQUETTE CUIVRÉE

Pour une propreté optimal, on peut enlever la résine déposée lors de la soudure avec une brosse à dent et de l'acétone. Ensuite vient la phase du test (moment critique de l'opération) ...

Les nappes d'interconnection

Pour relier les cartes entre elles, il faut utiliser des nappes. Dans notre application, vu que nous n'avons pas beaucoup d'interconnections, on fait une nappe avec du fil et des connecteurs tulipes (avec un pas classique de 2.54mm). On mettra des gaines thermo-réctractables pour proteger les soudures. Pour les cables de puissances, on prendra des cables de 2.5mm² de section, ce qui est légerement sur-dimentioné, mais il vaut mieux être le plus prudent possible avec les circuits de puissance.

UNE PERÇEUSE


ET SES FORÊTS

LA PROGRAMMATION

Nicolas Viennot

...:: INTRODUCTION ::...

Presentation des projets

Nous avons 4 projets de programmation pour notre application. Deux pour le programmateur de PIC et deux pour la voiture. Pour le programmateur de PIC, il faut programmer le PIC qui va interfaçer les données (vu dans la partie électronique) et bien sûr le logiciel qui va envoyer les données vers ce PIC. Pour la voiture, il va falloir coder le PIC de la voiture et le logiciel qui va servir de télécommande. Les programmes vont être réalisé en assembleur et en C pour les PICs, et en C++ pour les logiciels. Nous verrons rapidement le fonctionnement du PIC16F876 qui est un microcontrôleur 8 bits.

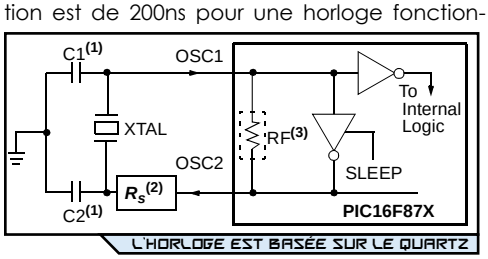
...:: LE PIC16F876 ::...

Les ports

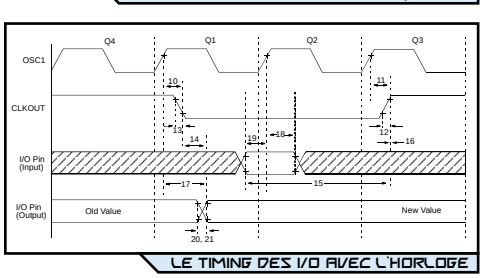
On dispose de 20 entrées/sorties décomposées en 3 ports distincts : le port A, B et C. Le port A possède 5 I/O, ce port peut être utilisé en port numérique ou analogique (convertion analogique/numérique) ou les 2. Les entrées sont du type TTL. Attention à la pin RA4 puisqu'elle est à collecteur ouvert, donc il faut en prendre compte si on l'utilise comme sortie. Le port B est classique avec 7 I/O et des sources d'interruptions que nous verrons ultérieurement. Les entrées sont du type TTL, on peut également les paramétrer avec des résistances de tirage en interne. Le port C possède 7 I/O, il est principalement utilisé pour les différents modules que nous verrons ultérieurement. Les entrées sont du type trigger de schmitt.

L'horloge

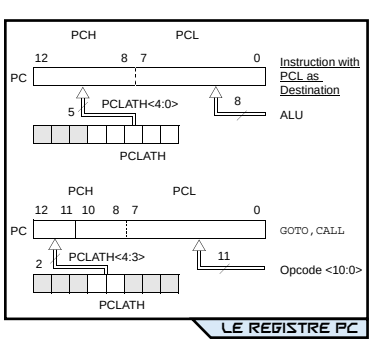
Comme nous l'avons vu dans la partie électronique, le temps d'éxecution d'une instruction est de 200ns pour une horloge fonctionnant à 20MHz tout simplement parceque l'éxecution d'une instruction se fait en 4 étapes. Généralement : en premier temps le décodage de l'instruction, ensuite la lecture de l'argument, puis le traitement des données , et enfin l'écriture des données. Ainsi 1 cycle d'instruction = 4 cycle d'horloge. Les seule instructions qui consomment 2 cycles d'instruction sont celles qui écrivent dans le PC.



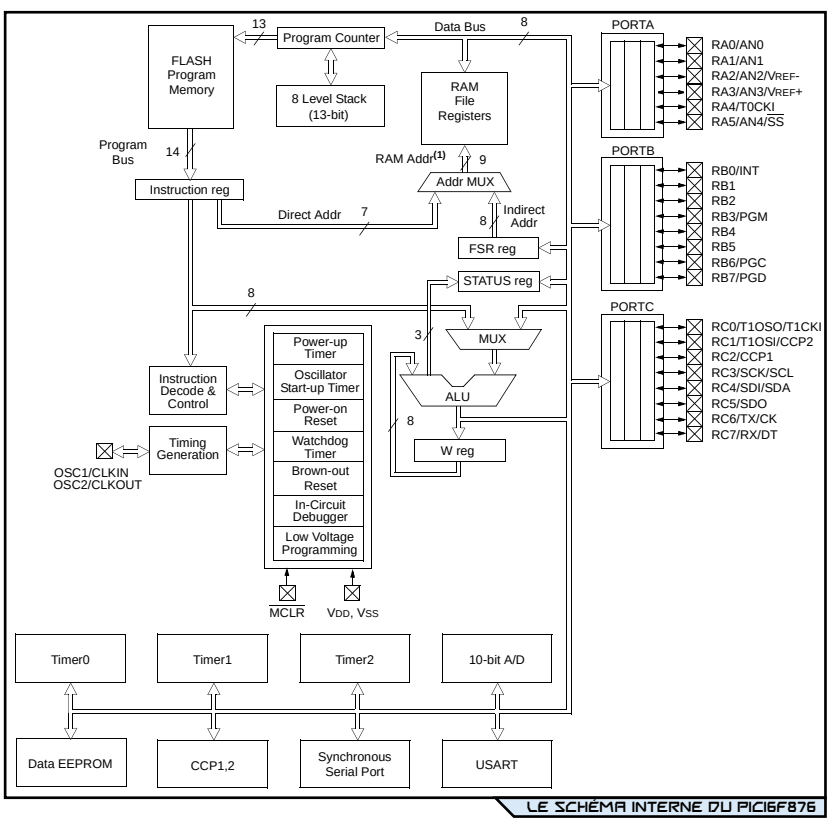
L'HORLOGE EST BASÉE SUR LE QUARTZ



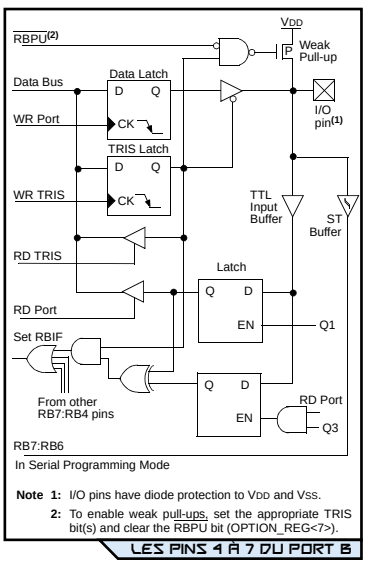
LE TIMING DES I/O AVEC L'HORLOGE



LE REGISTRE PC



LE SCHÉMA INTERNE DU PIC16F876

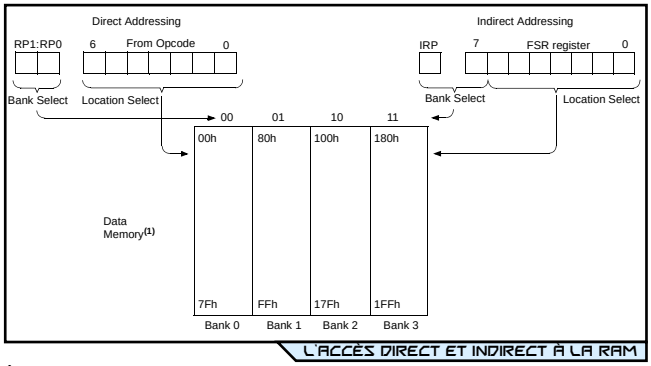


Note 1: I/O pins have diode protection to VDD and VSS.
2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).
LES PINS 4 À 7 DU PORT B

Les registres fondamentaux

Le PIC ne possède qu'un seul registre accumulateur de 8 bits (un pentium en possède quatres de 32bits chacun), c'est le registre qui va nous permettre de réaliser toute sortes de calculs. C'est un registre tampon, il s'appelle W. Le PC (Program Counter) est un ensemble de registres qui stoque l'addresse de la prochaine instruction à éxecuter. En effet, le processeur possède un sequenceur qui va executer une liste d'instructions organisées qui forment le programme. Le PC est de 13 bits car la ROM fait 8Ko. Il est composé du PCL (PC Low

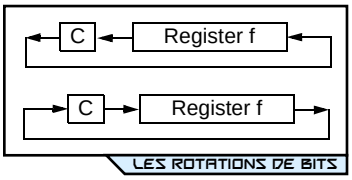
- 8bits) et du PCLATH (PC LATch High) car la ROM est composée de 8Ko d'instructions. Le registre STATUS est un registre qui rassemble des bits d'état du PIC. Il indique l'état d'une opération (débordement dans le bit de CARRY par exemple) ou encore compose les bits de sélection de banque de RAM. Les registres FSR et INDR forment un pointeur pour accéder à la mémoire indirectement. On charge l'addresse que l'on veut dans FSR et l'accède par INDR.



L'ACCÈS DIRECT ET INDIRECT À LA RAM

Les instructions

Il y a 35 instructions en tout. Il y a les opérations logiques : les OU, ET, XOR, NOT, IOR. Ainsi que les opérations arithmétiques comme l'addition et la soustraction mais de multiplication ni de division. Une routine multiplicative de deux nombres de 8 bits donnant un résultat sur 16 bits prend environ 70 instructions (avec des rotations de bits, addition, saut conditionnel, etc...). Il existe bien sûr des processeur éxecutant une multiplication en une seule instruction MAC (Multiply and ACcumulate). Il y a également les rotations de bit vers la gauche et vers la droite à travers le bit de retenue (CARRY du registre STATUS), ainsi que les opérations d'incrémentation et de décrémentation avec ou sans test de remise à 0. Les instructions de branchement (toutes celles qui modifient le PC) qui consomment 2 cycles d'instructions sont celles comme goto, return, call, jump. Il y a aussi les instructions pour positionner les bits comme bsf, les tests logiques etc... De nombreuses instructions prennent 2 opérandes, ce sont souvent un octet (addresse, nombre quelconque) et une destination (W, f).



LES ROTATIONS DE BITS

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		3
LITERAL AND CONTROL OPERATIONS						
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk	Z	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

LA LISTE DES 35 INSTRUCTIONS

La RAM

Abréviation pour Random Access Memory, c'est tout simplement une zone de mémoire que le programme va accéder pour stocker et lire des données relative à son fonctionnement. Cette mémoire est rapide d'accès. Au démarrage du PIC, elle contient des données aléatoires, ces données sont altérées si la tension d'alimentation baisse en dessous des 1.5V. Le PIC contient 368 octets de RAM et vu qu'il fonctionne en 8 bits, on va devoir y accéder par 3 différentes zones, dites banques. On va choisir une banque par les bits RP1:RP0. Il y a une zone commune aux 3 banques, c'est de 0x70 à àx7H. Cette zone de 16 octets est nécéssaire pour les interruptions que nous verrons ultérieurement, et elle est utile pour stocker les données très fréquement utilisées comme les états du programme.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

L'ACCES AUX BANQUES

	File Address		File Address		File Address		File Address
Indirect addr.(*)	00h	Indirect addr.(*)	80h	Indirect addr.(*)	100h	Indirect addr.(*)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD(1)	08h	TRISD(1)	88h		108h		188h
PORTE(1)	09h	TRISE(1)	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved(2)	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved(2)	18Fh
T1CON	10h		90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h	General Purpose Register 16 Bytes	117h	General Purpose Register 16 Bytes	197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h
	7Fh		EFh		16Fh		1EFh
accesses 70h-7Fh		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
Bank 0		Bank 1	FFh	Bank 2	17Fh	Bank 3	1FFh

Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on the PIC16F876.
2: These registers are reserved, maintain these registers clear.

LA RAM ET LES REGISTRES SPÉCIFIQUES

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR
Bank 0										
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000
01h	TMR0	Timer0 Module Register								xxxx xxxx
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000
0Bh(3)	INTCON	GIE	P EIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000
0Dh	PIR2	—	(5)	—	EEIF	BCLIF	—	—	CCP2IF	-r-0 0--0
0Eh	TMR1L	Holding register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx
0Fh	TMR1H	Holding register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON	--00 0000
11h	TMR2	Timer2 Module Register								0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000
15h	CCPR1L	Capture/Compare/PWM Register1 (LSB)								xxxx xxxx
16h	CCPR1H	Capture/Compare/PWM Register1 (MSB)								xxxx xxxx
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x
19h	TXREG	USART Transmit Data Register								0000 0000
1Ah	RCREG	USART Receive Data Register								0000 0000
1Bh	CCPR2L	Capture/Compare/PWM Register2 (LSB)								xxxx xxxx
1Ch	CCPR2H	Capture/Compare/PWM Register2 (MSB)								xxxx xxxx
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0
Bank 1										
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111
86h	TRISB	PORTB Data Direction Register								1111 1111
87h	TRISC	PORTC Data Direction Register								1111 1111
88h(4)	TRISD	PORTD Data Direction Register								1111 1111
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000
8Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000
8Dh	PIE2	—	(5)	—	EEIE	BCLIE	—	—	CCP2IE	-r-0 0--0
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq
8Fh	—	Unimplemented								—
90h	—	Unimplemented								—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000
92h	PR2	Timer2 Period Register								1111 1111
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000
95h	—	Unimplemented								—
96h	—	Unimplemented								—
97h	—	Unimplemented								—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000
9Ah	—	Unimplemented								—
9Bh	—	Unimplemented								—
9Ch	—	Unimplemented								—
9Dh	—	Unimplemented								—
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	0--- 0000

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR
Bank 2										
100h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000
101h	TMR0	Timer0 Module Register								xxxx xxxx
102h(3)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000
103h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
104h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx
105h	—	Unimplemented								—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx
107h	—	Unimplemented								—
108h	—	Unimplemented								—
109h	—	Unimplemented								—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000
10Bh(3)	INTCON	GIE	P EIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						xxxx xxxx
10Fh	EEADRH	—	—	—	EEPROM Address Register High Byte					xxxx xxxx
Bank 3										
180h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111
182h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000
183h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx
184h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx
185h	—	Unimplemented								—
186h	TRISB	PORTB Data Direction Register								1111 1111
187h	—	Unimplemented								—
188h	—	Unimplemented								—
189h	—	Unimplemented								—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000
18Bh(3)	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000
18Dh	EECON2	EEPROM Control Register2 (not a physical register)								---- ----
18Eh	—	Reserved maintain clear								0000 0000
18Fh	—	Reserved maintain clear								0000 0000

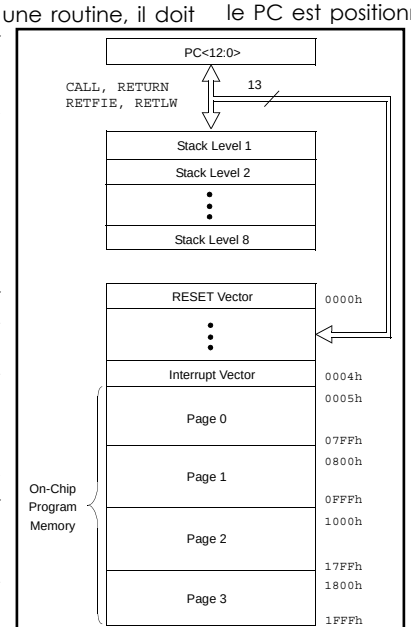
Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8> whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873/876 devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD, and TRISE are not physically implemented on PIC16F873/876 devices; read as '0'.
5: PIR2<6> and PIE2<6> are reserved on these devices; always maintain these bits clear.

LES REGISTRES SPÉCIFIQUES (SUITE)

La pile d'appels

Lorsque le programme veut faire appel à une routine, il doit éxecuter un 'call', On peut décomposer un call en quatres temps : l'addresse du PC est enregistrée dans la pile, puis on éxecute la routine, et ensuite par un return, on sort l'addresse de la pile pour la mettre dans le PC, et ainsi revenir dans le programme. On l'appelle pile car contrairement à un FIFO (que l'on verra plus tard), lorsque l'on ajoute des objets sur la pile, ces derniers se dépilent exactement comme une pile de livres. Le dernier livre empilé sera le premier livre dépilé. La pile du PIC comporte 8 emplacements, c'est à dire que l'on peut faire 8 appels de routines successives (une routine qui appelle une sous-routine etc...).



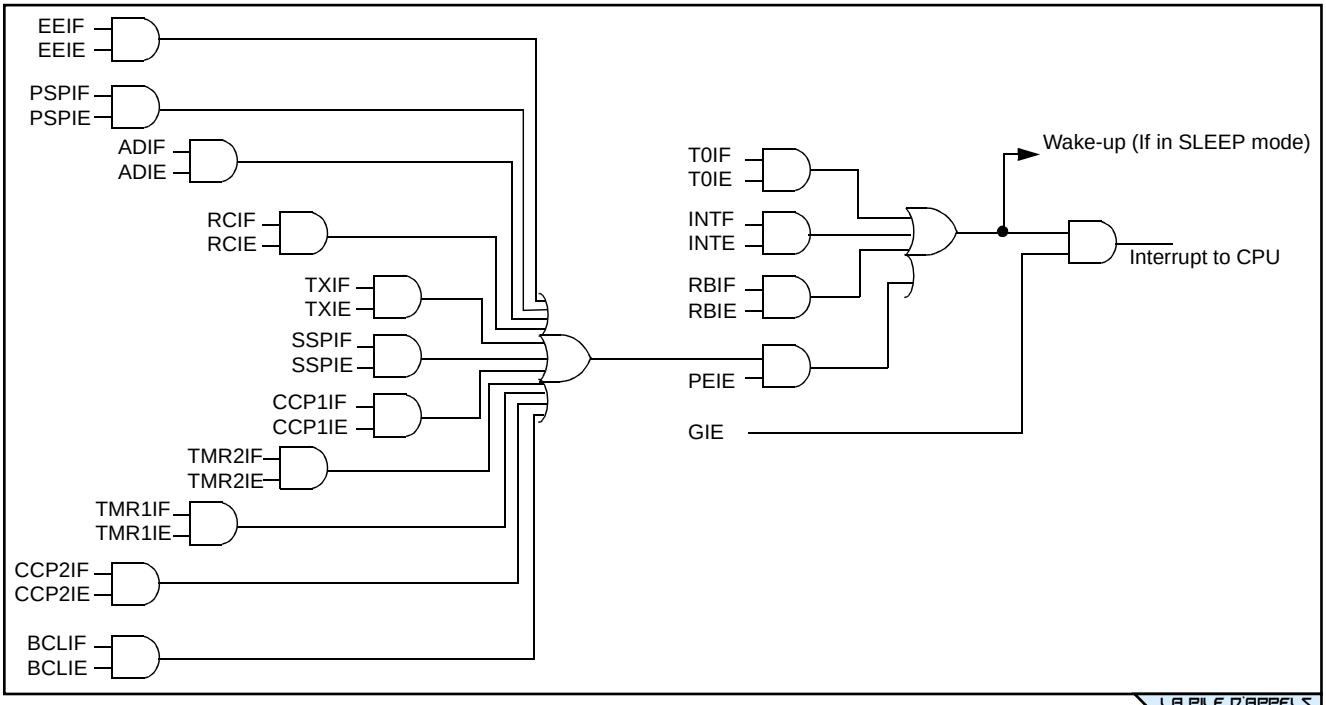
LA PILE D'APPELS

Les interruptions

Ce sont des 'évenements' qui interronpent directement le programme principal pour éxecuter des routines spécifiques. C'est une rupture de séquence asynchrone. En fait, c'est un appel et donc consomme une addresse dans la pile, ainsi on arrive à une limite de 7 'call' dans la routine d'interruption. Dès qu'une interruption survient, le PC est positionné au vecteur d'interruption (0x4) et le flag GIE est effacé pour etre repositioné par un 'retfie'. Il y a 14 sources d'interruptions differentes. Celles que nous utiliserons seront INT (changement d'état sur RB0, uniquement sur front montant ou descendant), les trois timers (cas d'un débordement), les deux évenements des modules USART (pour la transmission sérielle asynchrone), la conversion analogique/numérique. Chaque interruption peut être activée/désactivée à tout moment.

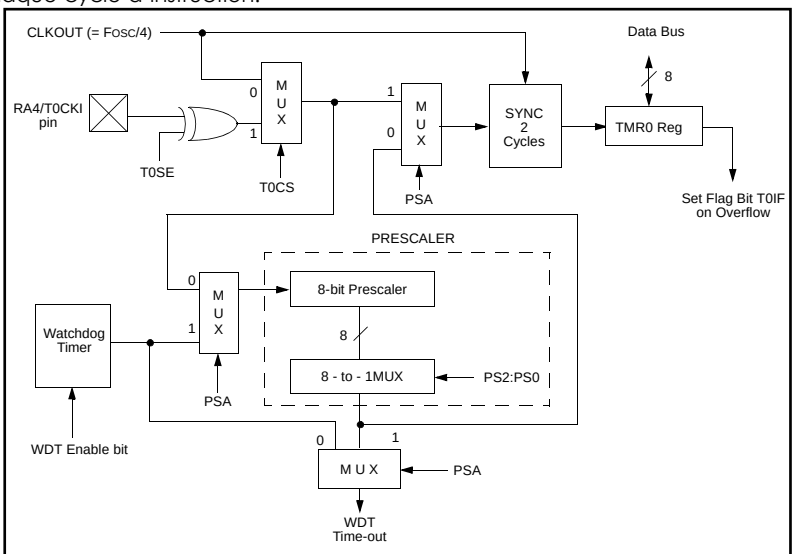
Les modules

Le PIC comporte trois timers. Le Timer0 (8 bits), le Timer1(16bits) et le Timer2 (8bits). Le timer0 et le timer1 peuvent faire office de compteur hardware pour compter des fronts montant ou descendants de la pin RA4 et RC0. La fréquence maximale de ces signaux ne doit pas excéder 10MHz (ce qui présente une marge certaine). Le Timer 1 et 2 sont utilisés pour les modules CPPs, l'interruption du Timer 2 est déclanchée quand ce dernier est égal à PR2 + 1. Les



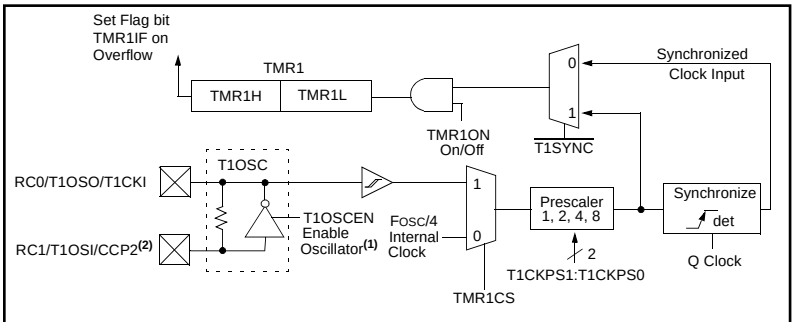
LA PILE D'APPELS

timers utilisés en mode timers sont incrémenté à chaque cycle d'instruction. On peut assigner aux timer des prédiviseurs, ce qui revient à augmanter leur base de temps. Si le Timer 0 à un prédiviseur de 8 alors tout les 8 cycles d'instruction, il sera incrémenté. Seul le Timer 2 est équipé d'un postdiviseur. Dans le PIC, il y a un système d'anti-plantage : le watchdog (ou chien de garde), le programme ne peut que le remettre à zero mais ne peut pas le lire. Dans le programme principal, on va éxecuter 'clrwdt' (clear watchdog) comme dans les longues boucles ou encore au début d'une routine, pour être sûr que le watchdog ne débordera pas, sinon le PIC va reseter (presque comme une remise sous tension). Si le PIC se promenne n'importe où dans la ROM et n'execute plus les 'clrwdt' alors il reset automatiquement, et attention à ne pas en mettre dans la routine d'interruption, sinon le watchdog est quand même remis à zero même en cas de plantage. Sans assigner de prédiviseur, il déborde au bout de 18ms (environ puisque ce dernier varie en fonction de la température du PIC). On peut lui assigner un prédiviseur (partagé avec celui du timer0) pour lui faire atteindre un temps maximum de 800ms environ. Le PIC comporte également



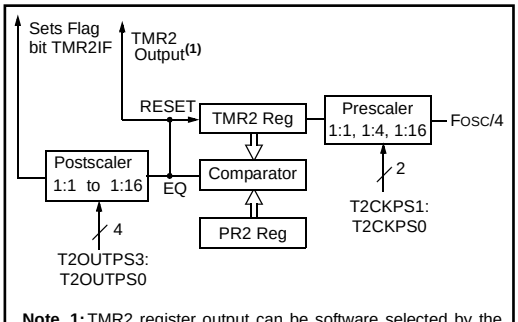
Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

LE TIMER 0 ET LE WATCHDOG



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

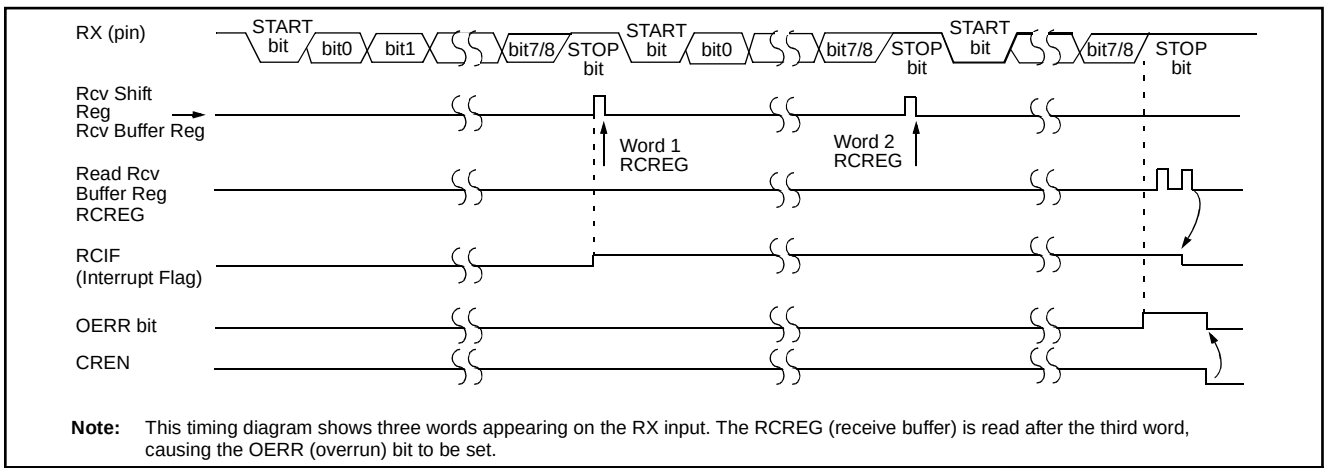
LE TIMER 1



Note 1: TMR2 register output can be software selected by the SSP module as a baud clock.

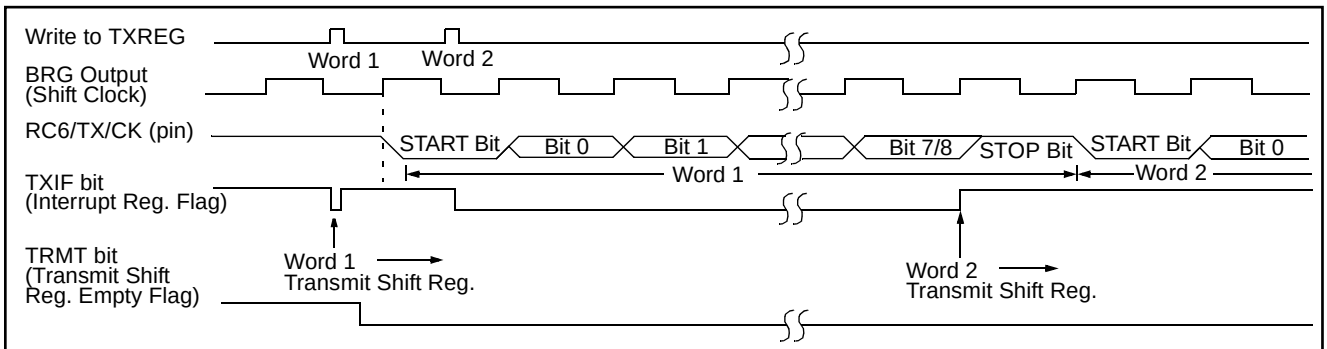
LE TIMER 2

des modules de communication sériel. On utilisera le module de communication USART en asynchrone (addressable Universal Synchronous Asynchronous Receiver Transmitter). Une liaison asynchrone est une communication où les interlocuteurs se sont mis d'accord sur la vitesse de transmission et donc ne nécéssite pas de ligne d'horloge. Tandis qu'une communication synchrone nécéssite une ligne d'horloge, ainsi à chaque frond déscendant ou montant, une lecture ou une écriture sur les lignes de données va opérer. On en verra en détails pour la transmission radio ou lors de la programmation de PIC. La vitesse de transmission minimal est de 1220bps et la vitesse maximale est de 1.25Mbps pour un quartz fonctionnant à 20MHz. Pour ne pas que l'un des interlocuteurs désynchronise, la tolérence maximale est de 5%, il vaut mieux rester en dessous des 3% voire 2. Avec un quartz à 20MHz, on a une erreur d'environ 1% avec les vitesse de communication classique (4800, 9600, etc...). Pour avoir une erreur nulle (à la précision du quartz) il faut utiliser des quartzs multiples de 3.3864MHz, ce qui peut être nécessaire dans certaines applications.



Note: This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.

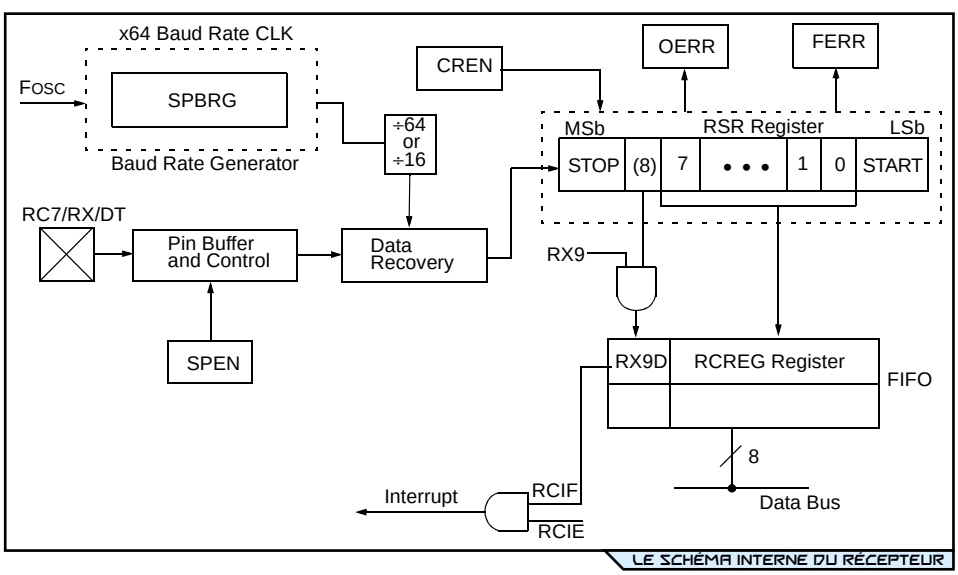
CHRONOGRAMME DE RÉCEPTION



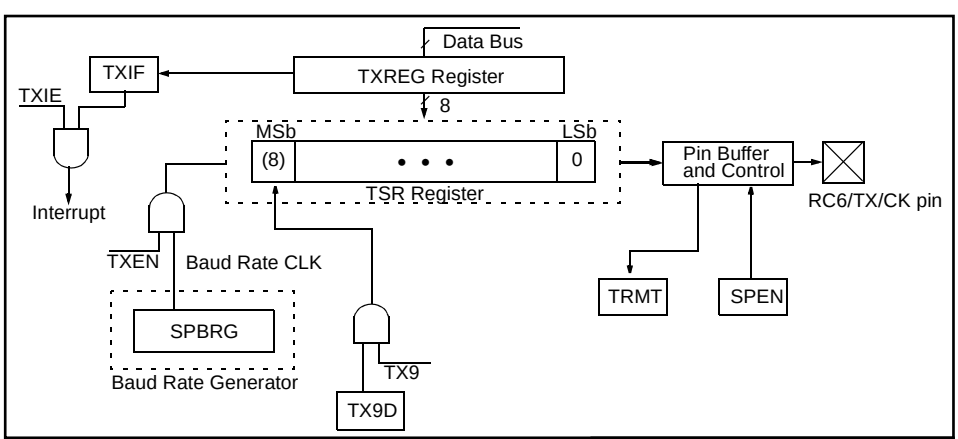
Note: This timing diagram shows two consecutive transmissions.

CHRONOGRAMME D'ÉMISSION

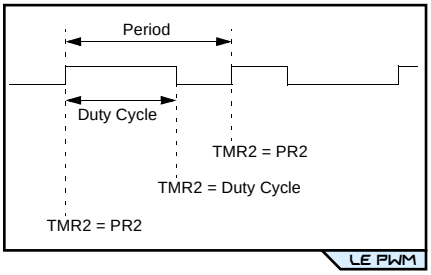
Ces modules fonctionne suivant la norme NRZ (non return to zero), c'est à dire un bit de start, 8 ou 9 bits de données, et un bit de start. On peut émuler le bit de paritée par le bit d'addressage (bit 9) logicielement (non supporté en hardware). La lecture d'un bit se fait en fesant trois lectures et en prennant la valeur la plus significative. Le module de réception intègre un FIFO (First In, First Out) de deux emlacements. Elle fonctionne comme une file d'attente dans un magazin : le premier arrivé est le premier traité, et le dernier attend jusqu'à ce qu'on le traite (d'où le fonctionement inverse de la pile). Ainsi le PIC peut avoir reçu deux octets et être en train de réceptionner un troisième avant qu'on les traite. Si le FIFO est plein et qu'un nouvel octet vient d'être reçu, alors le module passe en état d'overflow (bit OERR) et là, il faut couper la réception et la rélancer. Le module gère également les erreurs de frames (si le bit de stop = 0). On pilote les modules à travers RCSTA, TXSTA, SPBRG. Nous verrons ces modules en détails lors des communications dans notre application. Le PIC intègre également deux modules CPPs qui peut être utilisé en trois modes distincts : Capture, Compare, PWM. Nous allons utiliser le mode PWM (Pulse Width Modulation) qui signifie modulation de largeur d'impulsion. On peut paramétrer le PWM par sa fréquence et son



LE SCHÉMA INTERNE DU RÉCEPTEUR

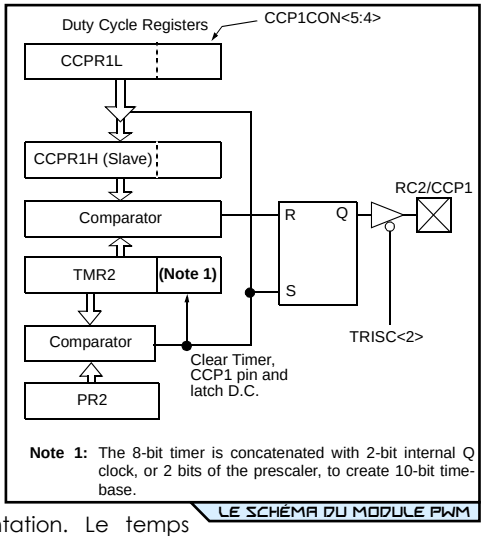


LE SCHÉMA INTERNE DE L'ÉMETTEUR

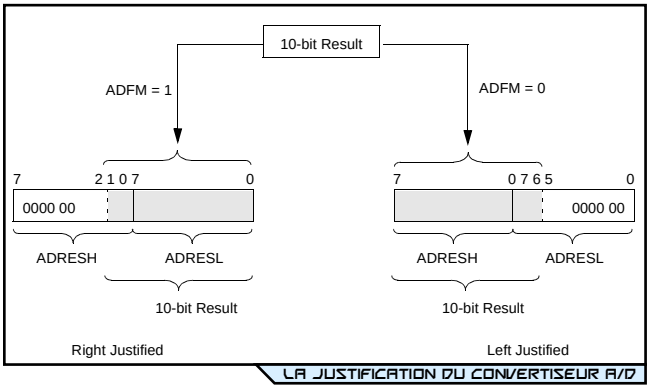


LE PWM

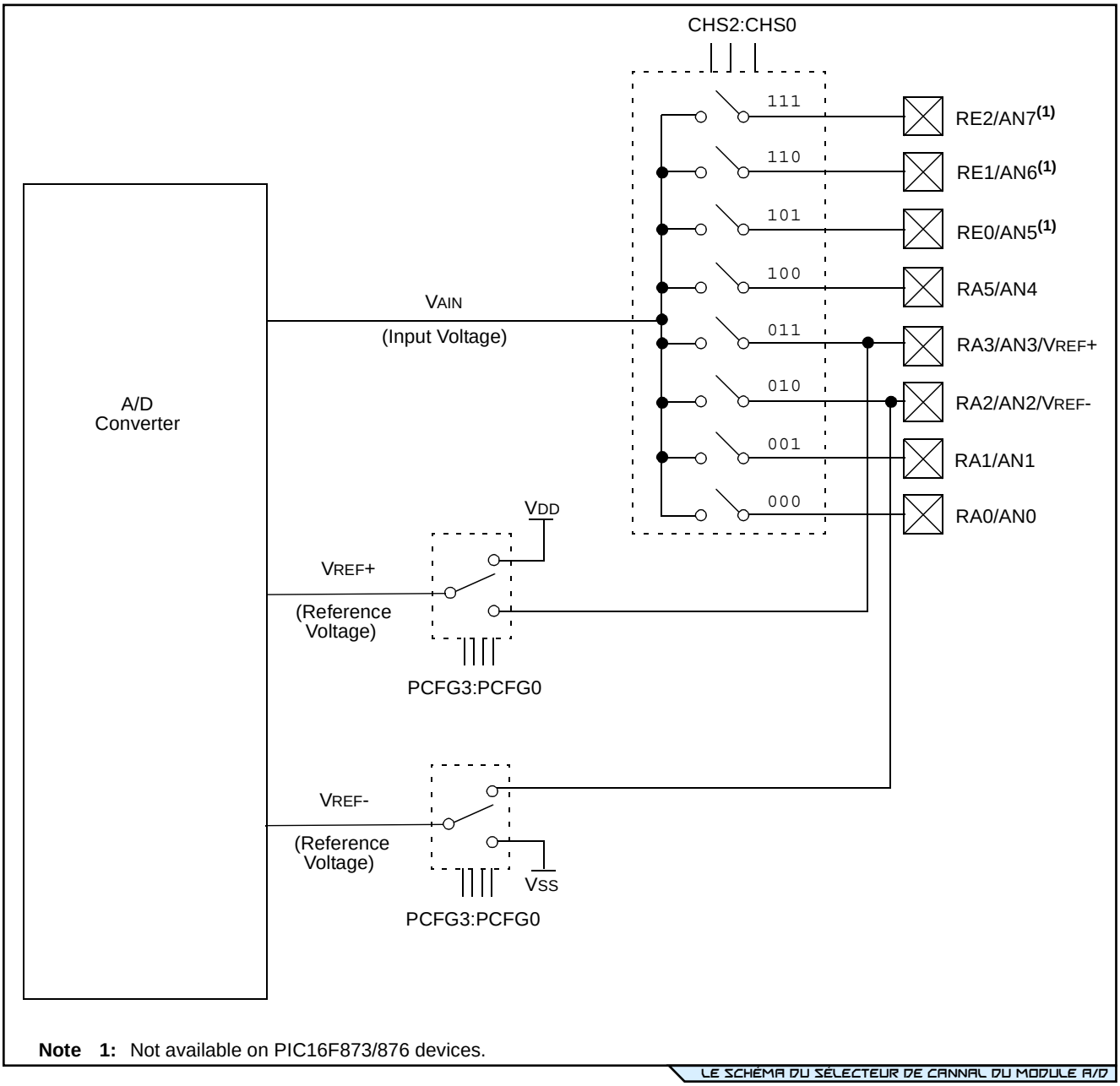
rapport cyclique. On arrive à atteindre une résolution maximale de 10bits (1024 valeurs possible) pour une fréquence de 20kHz et une résolution de 5.5bits (45 valeurs possibles) pour une fréquence de 200kHz. le PWM utilise le Timer 2 et son registre PR2. Le postscaler du Timer 2 n'influe en rien sur le PWM. Le PIC intègre également un convertisseur A/D (Analog to Digital) qui permet de lire une donnée analogique. Cette conversion s'effetue sur une seule pin du port A à la fois. On peut spécifier des tensions de référence autres que la tension d'alimentation. Le temps d'acquisition dépend de la tension d'alimentation du PIC, de sa température, et de la précision voulue. Dans le pire des cas, l'acquisition dure 20µs pour une précision optimale de 10bit. Le résultat peut être justifié à gauche ou à droite.Le PIC comporte d'autre modules : un module Compare, Capture, MSSP (Master Synchronous Serial Port) qui sert surtout pour le bus I²C.



LE SCHÉMA DU MODULE PWM



LA JUSTIFICATION DU CONVERTISEUR A/D



LE SCHÉMA DU SÉLECTEUR DE CANNAL DU MODULE A/D

...:: LE PROGRAMMATEUR DE PIC ::...

Introduction

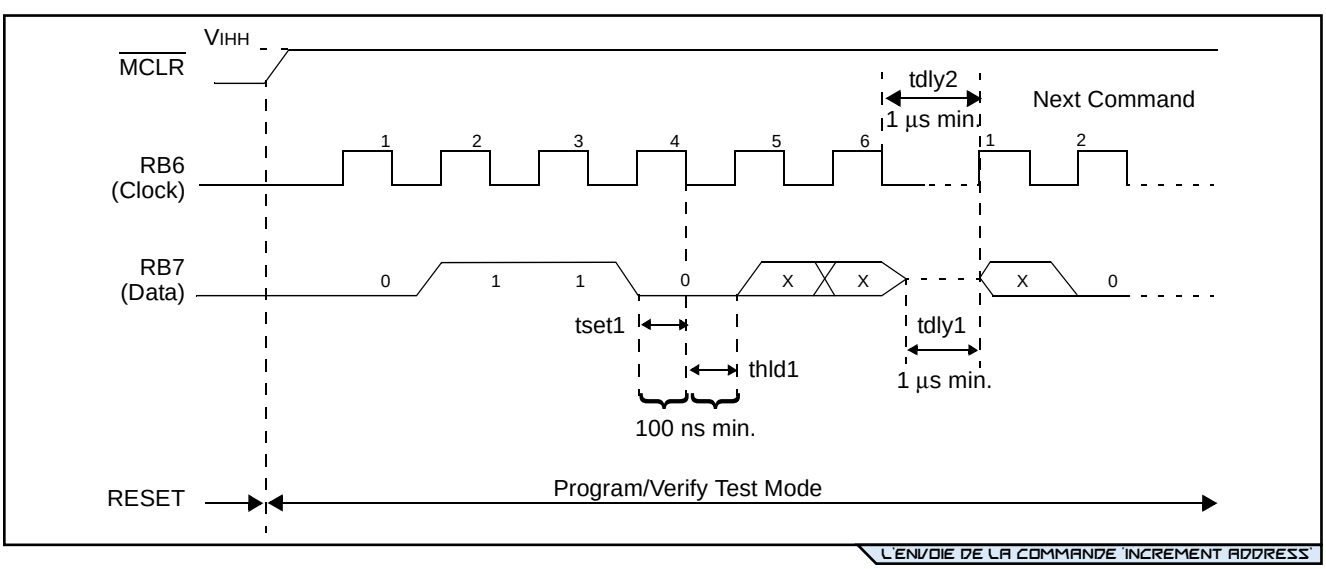
Notre programmateur de PIC doit fonctionner avec les pins Tx et Rx de port série qui est une transmission asynchrone et retransmettre un signal synchrone vers le PIC à programmer. On aurai pu utiliser un PIC moins performants que le 16F876 mais on va l'utiliser pour ses modules USART. La programmation du côté PIC va se faire en assembleur, ainsi on pourra se familiariser avec le PIC. Le logiciel côté PC va être en C++, langage orienté object très performent, nous aborderont rapidement la synchronisation entre les threads (définies ultiérieurement).

La programmation des PICs

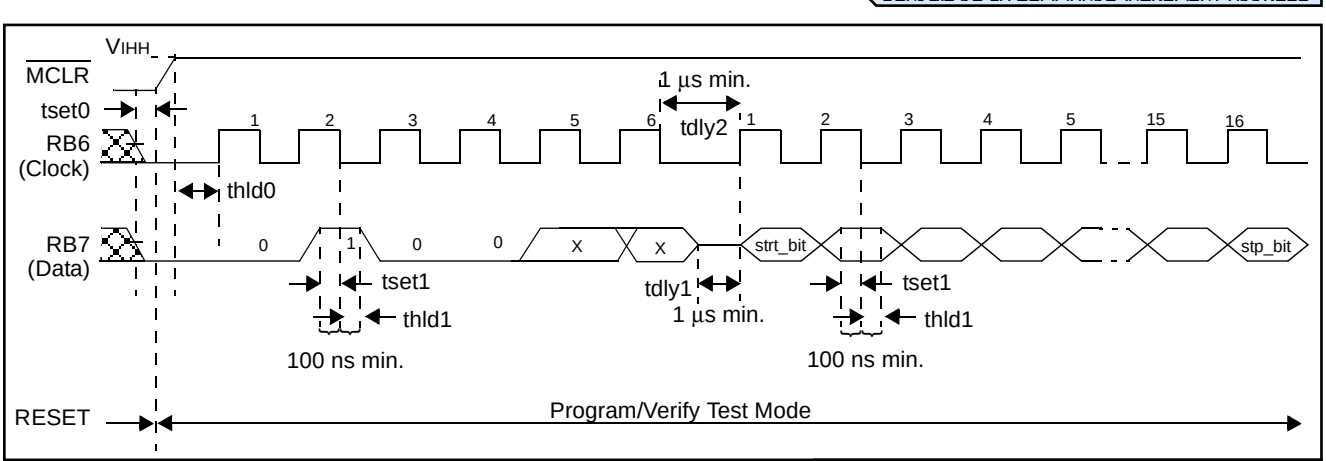
Pour programmer un PIC, on doit lui fournir les données de façon sérielle synchrone. c'est à dire avec une ligne de donnée et une ligne d'horloge (correspondant aux pins RB7 et RB6). La transmission se fait en half-duplex (les deux interlocuteurs ne parlent pas en même temps). On peut envoyer 10 commandes différentes pour le PIC16F876. On n'abordera pas du tout la programmation de l'EEPROM (donnée équivalente a la RAM sauf qu'elle est statique, elle ne s'efface pas). La programmation de la ROM (mémoire du programme) s'effectue

Command	Mapping (MSB ... LSB)						Data	Voltage Range
Load Configuration	X	X	0	0	0	0	0, data (14), 0	2.2V - 5.5V
Load Data for Program Memory	X	X	0	0	1	0	0, data (14), 0	2.2V - 5.5V
Read Data from Program Memory	X	X	0	1	0	0	0, data (14), 0	2.2V - 5.5V
Increment Address	X	X	0	1	1	0		2.2V - 5.5V
Begin Erase Programming Cycle	0	0	1	0	0	0		2.2V - 5.5V
Begin Programming Only Cycle	0	1	1	0	0	0		4.5V - 5.5V
Load Data for Data Memory	X	X	0	0	1	1	0, data (14), 0	2.2V - 5.5V
Read Data from Data Memory	X	X	0	1	0	1	0, data (14), 0	2.2V - 5.5V
Bulk Erase Setup1	0	0	0	0	0	1		4.5V - 5.5V
Bulk Erase Setup2	0	0	0	1	1	1		4.5V - 5.5V

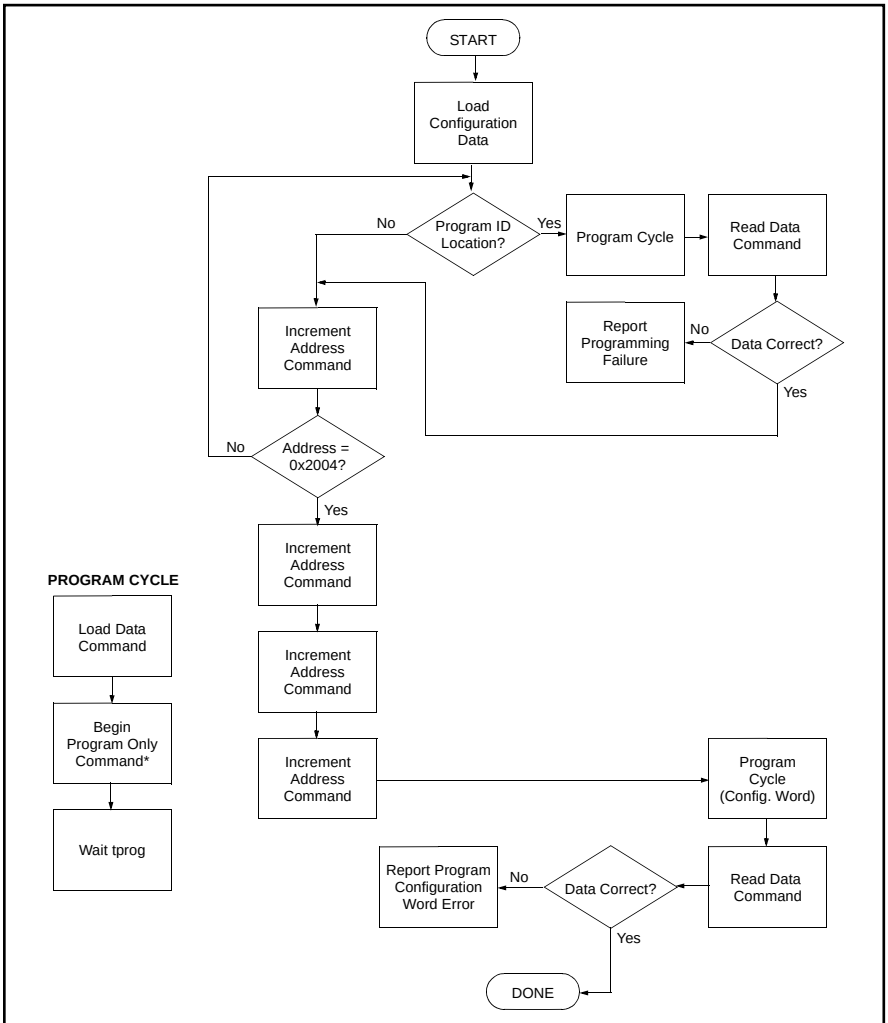
LA LISTE DES COMMANDES



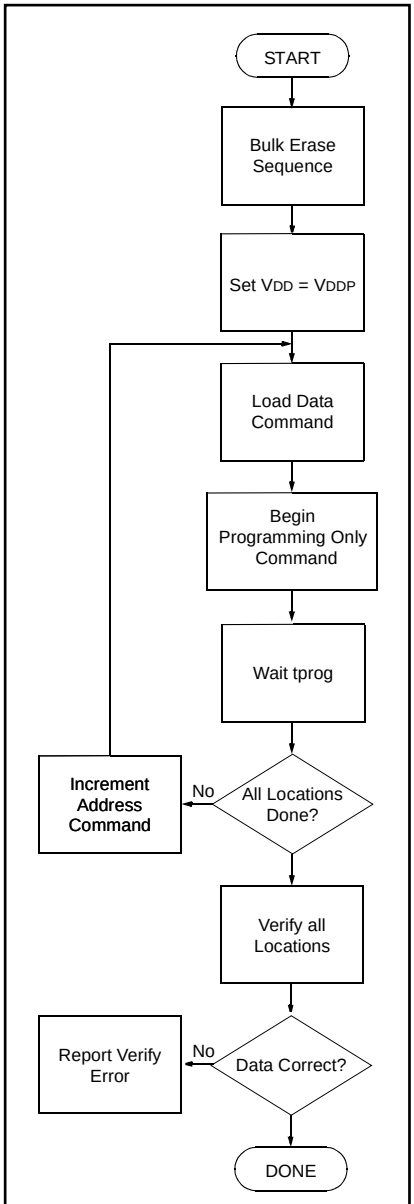
L'ENVOIE DE LA COMMANDE 'INCREMENT ADDRESS'



L'ENVOIE D'UNE DONNÉE VERS LE PIC À PROGRAMMER



L'ORGANIGRAMME DE LA PROGRAMMATION DE LA CONFIGURATION



LA PROGRAMMATION DE LA ROM

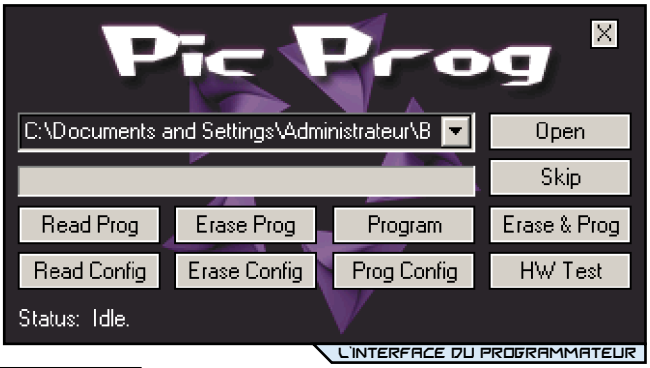
en chargeant un mot de 14 bits (taille d'une instruction) puis en effectuant la commande de programmation, etc... Pour programmer la configuration du PIC (type de quartz, protection de la ROM, etc...) il faut aller à l'addresse 0x2007. Il y a d'autre information à l'addresse 0x2000 telles que la version, la révision et l'identification du PIC.

Les protocoles

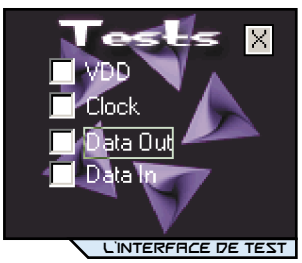
Le PC envoit des informations au PIC d'interface telle que les commandes à envoyer au PIC à programmer ou encore les données à programmer. La transmission s'effectue à 128000bps full-duplex (ligne de d'emission et réception séparée). Pour envoyer une commande au PIC, il faut l'envoyer de la manière 0b11xxxxxx où les 'x' sont le code binaire de la commande. Pour piloter manuelement les lignes de la tension de programmation, des données, et de l'horloge, on envoie 0b01xxxxxx (liste des commandes définie dans l'entête du programme ci-contre). Pour envoyer des données à programmer, on envoie 0b10xxxxxx + 0bxxxxxxxx (14 'x' en tout). Le PIC va répondre qu'il a en effet traité l'information par RX_OK (0x1) et RX_ERROR (0x2). RX_OK signifie que la commande ou l'envoie de donnée a été traitée et RX_ERROR indique que la donnée n'a pas été reçu et donc, que le PC doit la renvoyer. L'envoie des données du PIC au PIC à programmer se fait par des rotations de bits (voir les marcos SENDBIT et GETBIT). La temporisation tprog(le temps que la programmation s'effectue) se gère avec le timer1. Elle est de 4ms sans effacement et de 8ms avec effacement.

Le logiciel PC

Le programme est composé de deux threads et parfois trois. L'utilisation du multi-thread est pour simuler l'exécution simultanée de plusieur programme. En effet, la thread principale traite tout les messages de windows, comme les actions de l'utilisateur sur l'interface. La thread de réception (initialisée quand l'objet IpCOM est créé) traite tout les octets qui arrive depuis le port série. Et parfois la thread de programmation ou de lecture (sinon la thread principale serait bloquée et le logiciel aurai l'air planté) va, en suivant les organigrammes, répondrent à la demande de l'utilisateur. L'interface du programmateur intègre une barre de progression et une barre d'état (Status). Le bouton 'Skip' sert à stopper l'action en cours, 'HW Test' sert à entrer dans le mode de test manuel des I/O du PIC. Le logiciel accepte les fichier du format HEX16 de intel, format que les compilateurs génerent.



L'INTERFACE DU PROGRAMMATEUR



L'INTERFACE DE TEST

Le code source (PIC)

```
;*******************************************************************************;
;       Projet:         Programmateur de PIC
;       Date:           12/12/02
;       Version:        1.0
;*******************************************************************************;
;       Pins Configs:
;       =============
;
;               RB0: commande de VPP (inverse)
;               RC3: CLK
;               RC4: DATA
;               RC6: TX (PC)
;               RC7: RX (PC)
;
;       Protocole: (1 bit start, 8 bit, 0 bit de parite, 1 bit de stop
;       =========
;
;               PC -> Pic:
;                       - emission de la commande
;                       - si besoin de donnee, envoie de 2 octets de donnes
;                       - 11xxxxxx: commande a envoyer au pic
;                       - 01xxxxxx: HW test
;                               * 01 100001: VDD ON
;                               * 01 000001: VDD OFF
;                               * 01 100010: DATA ON
;                               * 01 000010: DATA OFF
;                               * 01 100100: CLOCK ON
;                               * 01 000100: CLOCK OFF
;                               * 01 101000: DATA IN EVENT ON
;                               * 01 001000: DATA IN EVENT OFF
;                       - 10xxxxxx + xxxxxxxx: Data
;*******************************************************************************;
        LIST      p=16F876            ; Définition de processeur
        #include <p16F876.inc>        ; fichier include
; Configuration generale
; ----------------------
        __CONFIG   _CP_OFF & _DEBUG_OFF & _WRT_ENABLE_OFF & _CPD_ON & _LVP_OFF & _BODEN_OFF & _PWRTE_ON
& _WDT_ON & _HS_OSC
                        ; _CP_OFF             Pas de protection du code
                        ; _DEBUG_OFF          RB6 et RB7 en utilisation normale (pas de debug)
                        ; _WRT_ENABLE_OFF     Le programme ne peut pas écrire dans la flash
                        ; _CPD_ON             Memoire EEprom protegee
                        ; _LVP_OFF            RB3 en utilisation normale (programmation)
                        ; _BODEN_OFF          Reset tension hors service (<4V)
                        ; _PWRTE_ON           Demarrage temporise
                        ; _WDT_ON             Watchdog on
                        ; _HS_OSC             Oscillateur haute vitesse (4Mhz<F<20Mhz)

; REGISTRE OPTION_REG (configuration)
; -----------------------------------
OPTIONVAL       EQU     B'10000111'
                        ; RBPU      b7 : 1= Résistance rappel +5V hors service
                        ; INTEDG    b6 : 1= Interrupt sur flanc montant de RB0
                        ;                 0= Interrupt sur flanc descend. de RB0
                        ; TOCS      b5 : 1= source clock = transition sur RA4
                        ;                 0= horloge interne
                        ; TOSE      b4 : 1= Sélection flanc montant RA4(si B5=1)
                        ;                 0= Sélection flanc descendant RA4
                        ; PSA       b3 : 1= Assignation prédiviseur sur Watchdog
                        ;                 0= Assignation prédiviseur sur Tmr0
                        ; PS2/PS0   b2/b0 valeur du prédiviseur
                        ;           000 =  1/1 (watchdog) ou 1/2 (tmr0)
                        ;           001 =  1/2               1/4
                        ;           010 =  1/4               1/8
                        ;           011 =  1/8               1/16
                        ;           100 =  1/16              1/32
                        ;           101 =  1/32              1/64
                        ;           110 =  1/64              1/128
                        ;           111 =  1/128             1/256

; REGISTRE INTCON (contrôle interruptions standard)
; -------------------------------------------------
INTCONVAL       EQU     B'01100000'
                        ; GIE       b7 : masque autorisation générale interrupt
                        ;                ne pas mettre ce bit à 1 ici
                        ;                sera mis en temps utile
```

```
                        ; PEIE      b6 : masque autorisation générale périphériques
                        ; T0IE      b5 : masque interruption tmr0
                        ; INTE      b4 : masque interuption RB0/Int
                        ; RBIE      b3 : masque interruption RB4/RB7
                        ; T0IF      b2 : flag tmr0
                        ; INTF      b1 : flag RB0/Int
                        ; RBIF      b0 : flag interruption RB4/RB7

; REGISTRE PIE1 (contrôle interruptions périphériques)
; ----------------------------------------------------
PIE1VAL                 EQU     B'00110000'
                        ; PSPIE     b7 : Toujours 0 sur PIC 16F786
                        ; ADIE      b6 : masque interrupt convertisseur A/D
                        ; RCIE      b5 : masque interrupt réception USART
                        ; TXIE      b4 : masque interrupt transmission USART
                        ; SSPIE     b3 : masque interrupt port série synchrone
                        ; CCP1IE    b2 : masque interrupt CCP1
                        ; TMR2IE    b1 : masque interrupt TMR2 = PR2
                        ; TMR1IE    b0 : masque interrupt débordement tmr1

; REGISTRE PIE2 (contrôle interruptions particulières)
; ----------------------------------------------------
PIE2VAL                 EQU     B'00000000'
                        ; UNUSED    b7 : inutilisé, laisser à 0
                        ; RESERVED  b6 : réservé, laisser à 0
                        ; UNUSED    b5 : inutilisé, laisser à 0
                        ; EEIE      b4 : masque interrupt écriture EEPROM
                        ; BCLIE     b3 : masque interrupt collision bus
                        ; UNUSED    b2 : inutilisé, laisser à 0
                        ; UNUSED    b1 : inutilisé, laisser à 0
                        ; CCP2IE    b0 : masque interrupt CCP2

; REGISTRE ADCON1 (ANALOGIQUE/DIGITAL)
; ------------------------------------
ADCON1VAL       EQU     B'00000110' ; PORTA en mode digital

; DIRECTION DES PORTS I/O
; -----------------------

DIRPORTA        EQU     B'11111111'     ; Direction PORTA (1=entrée)
DIRPORTB        EQU     B'11111110'     ; Direction PORTB
DIRPORTC        EQU     B'10100111'     ; Direction PORTC

;*****************************************************************************
;                               DEFINES                                      *
;*****************************************************************************
#DEFINE         NEW_CMD                         Flags,0
#DEFINE         READ                            Flags,1
#DEFINE         SEND                            Flags,2
#DEFINE         CMD_ONLY                        Flags,3
#DEFINE         WAIT_4MS                        Flags,4
#DEFINE         WAIT_8MS                        Flags,5

#DEFINE         VPP                             PORTB,0
#DEFINE         CLK                             PORTC,3
#DEFINE         DATA                            PORTC,4

#DEFINE         LOAD_CONFIG                     0x20
#DEFINE         LOAD_DATA_FOR_PROG              0x22
#DEFINE         READ_DATA_FROM_PROG             0x14
#DEFINE         INC_ADDR                        0x6
#DEFINE         BEGIN_ERASE_PROG                0x8
#DEFINE         BEGIN_PROG_ONLY                 0x18
#DEFINE         LOAD_DATA_FOR_DATA              0x23
#DEFINE         READ_DATA_FROM_DATA             0x15
#DEFINE         BULK_ERASE_SETUP1               0x1
#DEFINE         BULK_ERASE_SETUP2               0x7

#DEFINE         RX_OK                           0x1
#DEFINE         ERROR_RX                        0x2

;*****************************************************************************
;                               MACRO                                        *
;*****************************************************************************

                        ; Changement de banques
                        ; ----------------------
```

```
BANK0   macro                           ; passer en banque0
                bcf     STATUS,RP0
                bcf     STATUS,RP1
        endm

BANK1   macro                           ; passer en banque1
                bsf     STATUS,RP0
                bcf     STATUS,RP1
        endm

SENDBIT macro                           ; send bit du carry
                bsf     CLK             ; on monte l'horloge
                bcf     DATA            ; on met DATA off
                btfsc   STATUS,C        ; Carry == 0 ?
                bsf     DATA            ; si non, on le met on
                goto    $+1             ; on attends histoire de...
                goto    $+1
                goto    $+1
                goto    $+1
                goto    $+1

;------------------ CUT -- DEBUG ------------------
;               movlw   D'30'
;               movwf   Loop_wait
;               call    wait4ms
;               decfsz  Loop_wait
;               goto    $-2
;------------------ CUT -- DEBUG ------------------

                bcf     CLK             ; on descend l'horloge
                goto    $+1
                goto    $+1
                goto    $+1
                goto    $+1
                goto    $+1

;------------------ CUT -- DEBUG ------------------
;               movlw   D'30'
;               movwf   Loop_wait
;               call    wait4ms
;               decfsz  Loop_wait
;               goto    $-2
;------------------ CUT -- DEBUG ------------------

        endm

GETBIT  macro                           ; receptioner dans le carry
                bsf     CLK             ; on monte l'horloge
                bcf     STATUS,C        ; clear carry
                goto    $+1             ; on attends histoire de...
                goto    $+1
                goto    $+1
                goto    $+1
                goto    $+1

;------------------ CUT -- DEBUG ------------------
;               movlw   D'30'
;               movwf   Loop_wait
;               call    wait4ms
;               decfsz  Loop_wait
;               goto    $-2
;------------------ CUT -- DEBUG ------------------


                btfsc   DATA            ; Data == 0 ?
                bsf     STATUS,C        ; non, on met le carry
                bcf     CLK             ; on descend l'horloge

                goto    $+1
                goto    $+1
                goto    $+1
                goto    $+1
                goto    $+1
;------------------ CUT ------------------
;               movlw   D'30'
;               movwf   Loop_wait
```

```
;               call    wait4ms
;               decfsz  Loop_wait
;               goto    $-2
;------------------ CUT -----------------


        endm


;*****************************************************************************
;                       VARIABLES BANQUE 0                                  *
;*****************************************************************************

; Zone de 80 bytes
; ----------------

        CBLOCK          0x20            ; Début de la zone (0x20 à 0x6F)
        TXBuffer        : 2             ; Buffer d'emission
        TXPtr           : 1             ; Pointeur d'emission
        RXBuffer        : 3             ; Buffer de reception
        RXPtr           : 1             ; Pointeur de reception

        PicCmd          : 1             ; commande a envoyer
        PicData         : 2             ; data
        Loop            : 1             ; compteur
        Loop_wait       : 1             ; compteur de tempo


         ENDC                           ; Fin de la zone



;*****************************************************************************
;                       VARIABLES ZONE COMMUNE                              *
;*****************************************************************************

; Zone de 16 bytes
; ----------------

        CBLOCK 0x70                     ; Début de la zone (0x70 à 0x7F)
        Flags           : 1             ; Flags
        w_temp          : 1             ; Sauvegarde registre W
        status_temp     : 1             ; sauvegarde registre STATUS
        FSR_temp        : 1             ; sauvegarde FSR (si indirect en interrupt)
        PCLATH_temp     : 1             ; sauvegarde PCLATH (si prog>2K)

        ENDC

;*****************************************************************************
;                       DEMARRAGE SUR RESET                                 *
;*****************************************************************************

        org     0x000                   ; Adresse de départ après reset
        goto    init                    ; Initialiser

; //////////////////////////////////////////////////////////////////////////

;                         I N T E R R U P T I O N S

; //////////////////////////////////////////////////////////////////////////

;*****************************************************************************
;                     ROUTINE INTERRUPTION                                  *
;*****************************************************************************

                        ;sauvegarder registres
                        ;---------------------
        org     0x004           ; adresse d'interruption
        movwf   w_temp                  ; sauver registre W
        swapf   STATUS,w                ; swap status avec résultat dans w
        movwf   status_temp             ; sauver status swappé
        movf    FSR , w                 ; charger FSR
        movwf   FSR_temp                ; sauvegarder FSR
        movf    PCLATH , w              ; charger PCLATH
        movwf   PCLATH_temp             ; le sauver
        clrf    PCLATH                  ; on est en page 0
        BANK0                           ; passer en banque0
```

```
                        ; Interruption TMR0
                        ; -----------------

        btfsc   INTCON,T0IE     ; tester si interrupt timer autorisée
        btfss   INTCON,T0IF     ; oui, tester si interrupt timer en cours
        goto    intsw1          ; non test suivant
        call    inttmr0         ; oui, traiter interrupt tmr0
        bcf     INTCON,T0IF     ; effacer flag interrupt tmr0
        goto    restorereg      ; et fin d'interruption


                        ; Interruption transmission USART
                        ; -------------------------------
intsw1:
        bsf     STATUS,RP0      ; sélectionner banque1
        btfss   PIE1,TXIE       ; tester si interrupt autorisée
        goto    intsw2          ; non sauter
        bcf     STATUS,RP0      ; oui, sélectionner banque0
        btfss   PIR1,TXIF       ; oui, tester si interrupt en cours
        goto    intsw2          ; non sauter
        call    inttx           ; oui, traiter interrupt
                                ; LE FLAG NE DOIT PAS ETRE REMIS A 0
                                ; C'EST L'ECRITURE DE TXREG QUI LE PROVOQUE


                        ; Interruption réception USART
                        ; ----------------------------
intsw2:
        bsf     STATUS,RP0      ; sélectionner banque1
        btfss   PIE1,RCIE       ; tester si interrupt autorisée
        goto    restorereg      ; non sauter
        bcf     STATUS,RP0      ; oui, sélectionner banque0
        btfss   PIR1,RCIF       ; oui, tester si interrupt en cours
        goto    restorereg      ; non sauter
        call    intrc           ; oui, traiter interrupt
                                ; LE FLAG NE DOIT PAS ETRE REMIS A 0
                                ; C'EST LA LECTURE DE RCREG QUI LE PROVOQUE


                        ;restaurer registres
                        ;-------------------
restorereg:
        movf    PCLATH_temp,w   ; recharger ancien PCLATH
        movwf   PCLATH          ; le restaurer
        movf    FSR_temp,w      ; charger FSR sauvé
        movwf   FSR             ; restaurer FSR
        swapf   status_temp,w   ; swap ancien status, résultat dans w
        movwf   STATUS          ; restaurer status
        swapf   w_temp,f        ; Inversion L et H de l'ancien W
                                ; sans modifier Z
        swapf   w_temp,w        ; Réinversion de L et H dans W
                                ; W restauré sans modifier status
        retfie                  ; return from interrupt

;*********************************************************************************
;                          INTERRUPTION TIMER 0                                  *
;*********************************************************************************
inttmr0:

        movf    RXPtr,f         ; test du pointeur de reception
        btfsc   STATUS,Z        ; == 0 ?
        goto    tmr0end         ; oui on ne declare pas d'erreur
        clrf    RXPtr           ; raz du pointeur RX

        movlw   ERROR_RX        ; erreur de reception
        call    sendack         ; on l'envoie
tmr0end:
        return                  ; fin d'interruption timer


;*********************************************************************************
;                       INTERRUPTION RECEPTION USART                             *
;*********************************************************************************
intrc:
        clrf    TMR0
```

```
        movlw   RXBuffer        ; pointer le buffer de reception
        addwf   RXPtr,w         ; indexer avec le pointeur
        movwf   FSR             ; charger dans TSR
        movf    RCREG,w         ; charger la donnee lue
        movwf   INDF            ; et la sauver dans le buffer indexe

                ; Detection du type de donnee recu
                ; --------------------------------

        movf    RXPtr,f         ; test == 0 ?
        btfsc   STATUS,Z        ; ?
        goto    ptr0            ; oui, on gere la commande

                ; Reception du 2eme octet de donnee
                ; --------------------------------

        bcf     STATUS,C
        rlf     RXBuffer+1,f    ; rotation d'un bit vers la gauche
        rlf     RXBuffer,f      ; (bit start + bit stop)
        movf    RXBuffer+1,w    ; on charge le tout dans les donnees
        movwf   PicData+1
        movf    RXBuffer,w
        movwf   PicData
        clrf    RXPtr           ; raz du pointeur de reception
        movlw   RX_OK           ; reception ok
        call    sendack         ; l'envoyer
        return                  ; et on sort

ptr0:
        btfsc   RXBuffer,7      ; b7 == 1?
        btfss   RXBuffer,6      ; b6 == 1?
        goto    rcnext          ; non, on verifie les autres possibilitees
        goto    rccmd           ; oui, on traite une commande
rcnext:
        btfss   RXBuffer,7      ; b7 == 0?
        btfss   RXBuffer,6      ; b6 == 1?
        goto    rcdata          ; non, on recoie des donnees
                                ; oui, on traite le HW test

                ; Gestion manuelle des sorties
                ; ----------------------------
        movlw   RX_OK           ; reception ok
        call    sendack         ; l'envoyer

        btfsc   RXBuffer,5      ; b5 == 0
        goto    hwon            ; non, sortie on
                                ; oui, sortie off
        btfsc   RXBuffer,0      ; VDD ?
        bsf     VPP             ; = 0
        btfsc   RXBuffer,1      ; DATA ?
        bcf     DATA            ; = 0
        btfsc   RXBuffer,2      ; CLOCK ?
        bcf     CLK             ; = 0
        return                  ; on sort
hwon:
        btfsc   RXBuffer,0      ; VDD ?
        bcf     VPP             ; = 1
        btfsc   RXBuffer,1      ; DATA ?
        bsf     DATA            ; = 1
        btfsc   RXBuffer,2      ; CLOCK ?
        bsf     CLK             ; = 1
        return

                ; Gestion des donnees
                ; --------------------
rcdata:
        bcf     RXBuffer,6      ; clr du bit 6 (bit de stop)
        incf    RXPtr,f         ; incrementation du pointeur de reception
        return                  ; on sort

                ; Gestion de la commande pour le PIC
                ; ----------------------------------

rccmd:
        movwf   PicCmd          ; on sauve la commande en memoire
        bsf     NEW_CMD         ; on valide l'envoie de commande
        return                  ; fin d'interruption
```

```
;*************************************************************************
;                     INTERRUPTION TRANSMISSION USART                     *
;*************************************************************************
inttx:
        movf    TXBuffer,w      ; Charger le buffer
        movwf   TXREG           ; dans TXREG
        btfss   TXBuffer,7      ; Data ?
        goto    txend
        movf    TXBuffer+1,w    ; Charger le buffer (2eme octet)
        movwf   TXREG           ; dans TXREG
txend:
        BANK1                   ; on arrte la transmission
        bcf     PIE1,TXIE
        BANK0
        return                  ; on sort


; //////////////////////////////////////////////////////////////////////////

;                           P R O G R A M M E

; //////////////////////////////////////////////////////////////////////////

;*************************************************************************
;                          INITIALISATIONS                                *
;*************************************************************************
init:

                        ; initialisation PORTS (banque 0 et 1)
                        ; ------------------------------------
        BANK0                   ; sélectionner banque0
        bsf     STATUS,RP0      ; passer en banque1
        movlw   ADCON1VAL       ; PORTA en mode digital/analogique
        movwf   ADCON1          ; écriture dans contrôle A/D
        movlw   DIRPORTA        ; Direction PORTA
        movwf   TRISA           ; écriture dans registre direction
        movlw   DIRPORTB        ; Direction PORTB
        movwf   TRISB           ; écriture dans registre direction
        movlw   DIRPORTC        ; Direction PORTC
        movwf   TRISC           ; écriture dans registre direction
        BANK0
        clrf    PORTA           ; Sorties PORTA à 0
        clrf    PORTB           ; sorties PORTB à 0
        clrf    PORTC           ; sorties PORTC à 0
        bsf     VPP             ; VPP = 0
        BANK1

                        ; Registre d'options (banque 1)
                        ; -----------------------------
        movlw   OPTIONVAL       ; charger masque
        movwf   OPTION_REG      ; initialiser registre option

                        ; registres interruptions (banque 1)
                        ; ----------------------------------
        movlw   INTCONVAL       ; charger valeur registre interruption
        movwf   INTCON          ; initialiser interruptions
        movlw   PIE1VAL         ; Initialiser registre
        movwf   PIE1            ; interruptions périphériques 1
        movlw   PIE2VAL         ; initialiser registre
        movwf   PIE2            ; interruptions périphériques 2

                        ; Effacer RAM banque 0
                        ; --------------------
        bcf     STATUS,RP0      ; sélectionner banque 0
        movlw   0x20            ; initialisation pointeur
        movwf   FSR             ; d'adressage indirect
init1:
        clrf    INDF            ; effacer ram
        incf    FSR,f           ; pointer sur suivant
        btfss   FSR,7           ; tester si fin zone atteinte (>7F)
        goto    init1           ; non, boucler
```

```
                        ; USART config
                        ; --------------
        BANK1
        movlw   B'00100100'     ; charger TX config
        movwf   TXSTA           ; dans TXSTA
        movlw   D'9'            ; 20e6/(16*(9+1)) = 125000bps (2.3% d'erreur)
;       movlw   D'86'           ; 20e6/(16*(86+1)) = 14368bps (2.2% d'erreur)
;       movlw   D'129'          ; 20e6/(16*(129+1)) = 9615bps (0.2% d'erreur)
        movwf   SPBRG           ; configurer le baud rate
        BANK0
        movlw   B'00010000'     ; charger RX config
        movwf   RCSTA           ; dans RCSTA

                        ; Timer1 config
                        ; --------------
        movlw   B'00000000'     ; charger config
        movwf   T1CON           ; dans T1CON


                        ; autoriser interruptions (banque 0)
                        ; ---------------------------------
        clrf    PIR1            ; effacer flags 1
        clrf    PIR2            ; effacer flags 2
        bsf     RCSTA,SPEN      ; on lance la reception
        clrf    RXPtr           ; raz du pointeur de reception
        bsf     INTCON,GIE      ; valider interruptions
        clrf    Flags           ; clr flags
        goto    start           ; programme principal


;*****************************************************************************
;                       PROGRAMME PRINCIPAL                                  *
;*****************************************************************************
                        ; attente des 4ms
                        ; -----------------
wait4ms:
        movlw   0xB1            ; charger -20000 (4e-3/0.2e-6 = 20000)
        movwf   TMR1H
        movlw   0xE4
        movwf   TMR1L
        bsf     T1CON,TMR1ON    ; et lancer le timer1 ...
waitl:
        btfsc   PIR1,TMR1IF     ; debordement ?
        goto    waitend         ; oui, on sort
        clrwdt                  ; effacer watch dog
        goto    waitl           ; on boucle

waitend:
        bcf     PIR1,TMR1IF     ; on efface le flag d'interruption
        bcf     T1CON,TMR1ON    ; on arrete le timer
        return
waittx:
        BANK1
        clrwdt                  ; effacer watch dog
        btfsc   PIE1,TXIE       ; reste des caractères à envoyer?
        goto    $-2             ; oui, attendre
        btfss   TXSTA,TRMT      ; buffer de sortie vide?
        goto    $-4             ; non, attendre
        BANK0
        return

                        ; envoie d'un retour
                        ; -------------------
sendack:
        call    waittx          ; on attends l'emmission precedente
        movwf   TXBuffer        ; on charge le message
        BANK1
        bsf     PIE1,TXIE       ; et on l'envoie
        BANK0
        return
                        ; Start
                        ; ------
start:
        movlw   RX_OK           ; on a recu la commande
        call    sendack

        movlw   0x1             ; on reset Flags (tout en gardant NEW_CMD)
```

```
        andwf   Flags,f         ; avec un simple et logique

waitdata:
        clrwdt                  ; effacer watch dog
        btfss   NEW_CMD         ; nouvelles donnees ?
        goto    waitdata        ; non on boucle


                        ; Set Flags
                        ; ----------
        bcf     NEW_CMD         ; on valide

        btfss   PicCmd,3        ; Prog ?
        goto    set1            ; non
        bsf     WAIT_4MS        ; wait 4ms pour la prog
        btfss   PicCmd,4        ; Erase ?
        bsf     WAIT_8MS        ; wait 4ms de plus pour erase
        goto    cmd             ; on envoie la commande
set1:
        btfsc   PicCmd,5        ; Send Data ?
        bsf     SEND            ; oui
        btfsc   PicCmd,4        ; Read Data ?
        bsf     READ            ; oui
        bcf     PicCmd,4        ; on clr
        bcf     PicCmd,5        ; on clr


                        ; Send Command
                        ; ------------
cmd:
        movlw   D'6'            ; 6 bits a envoyer
        movwf   Loop            ; a mettre dans le compteur
cmdl:
        rrf     PicCmd,f        ; placer les bits dans le Carry
        SENDBIT                 ; on envoie
        decfsz  Loop,f          ; on decremente le compteur et si == 0, on sort
        goto    cmdl            ; on envoie tous les bits
        bcf     DATA            ; on clr data
        bcf     CLK             ; et clk
        clrwdt                  ; effacer watch dog

                        ; Branchement
                        ; -----------
        btfsc   SEND            ; on envoie des donnes ?
        goto    send            ; oui, on va dans la routine d'envoi
        btfsc   READ            ; on recois des donnes ?
        goto    read            ; oui, on va dans la routine de reception
                                ; on a recu une simple commande
        btfss   WAIT_4MS        ; on attends 4ms ?
        goto    start           ; non, on boucle
        call    wait4ms         ; oui, on attends
        btfss   WAIT_8MS        ; on attends 4ms de plus ?
        goto    start           ; non, on boucle
        call    wait4ms         ; oui, on attends
        goto    start           ; et revient au depart

                        ; Send Data
                        ; ---------
send:
        movlw   D'16'           ; 16 bits a envoyer
        movwf   Loop            ; a mettre dans le compteur
sendl:
        rrf     PicData,f       ; rotation de bit de PicData
        rrf     PicData+1,f     ; le bit qui sort va etre envoye
        SENDBIT                 ; on envoie le bit
        decfsz  Loop,f          ; on decremente le compteur et si == 0, on sort
        goto    sendl           ; on envoie tous les bits
        bcf     DATA            ; on clr data
        bcf     CLK             ; et clk
        goto    start           ; et revenir au depart

                        ; Read Data
                        ; ---------
read:
        BANK1
        bsf     DATA            ; passer DATA en entree
        BANK0
```

```
        movlw   D'16'           ; 16 bits a lire
        movwf   Loop            ; a mettre dans le compteur
readl:
        GETBIT                  ; on lit le bit de data (dans le carry)
        rrf     PicData,f       ; on l'enregistre
        rrf     PicData+1,f
        decfsz  Loop,f          ; on decremente le compteur et si == 0, on sort
        goto    readl           ; on boucle
        BANK1
        bcf     DATA            ; passer DATA en sortie
        BANK0
        bcf     DATA            ; on clr data
        bcf     CLK             ; et clk
                                ; on verifie les donnees

        bsf     STATUS,C        ; on met le carry
        bcf     PicData,7       ; on enleve le stop bit
        rrf     PicData,f       ; on enleve le start bit
        rrf     PicData+1,f     ; par rotations

        call    waittx

        movf    PicData,w       ; on charge PicData
        movwf   TXBuffer        ; dans le buffer d'emission
        movf    PicData+1,w
        movwf   TXBuffer+1

        BANK1
        bsf     PIE1,TXIE       ; on lance l'emission
        BANK0
        goto    start           ; on revient au depart

        END                     ; directive fin de programme
```

Le code source (PC)

Main.cpp

```
#define WIN32_LEAN_AND_MEAN
#include "windows.h"
#include "resource.h"
#include "shlobj.h"
#include "Com.h"
#include "Pic.h"
#include "Hex.h"
#include "SHELLAPI.H"

#define READ_LEN           0x2008
#define BUFFER_LEN         0xFFFF

#define   HW_VDDON         0x61
#define   HW_VDDOFF        0x41
#define   HW_DATAOUTON     0x62
#define   HW_DATAOUTOFF    0x42
#define   HW_CLOCKON       0x64
#define   HW_CLOCKOFF      0x44
#define   HW_ON            0x68
#define   HW_OFF           0x48


#define   MAXPATH          5

CCom*             lpCom=0;
CPic*             lpPic=0;
CHex*             lpHex=0;
HINSTANCE         hIns;
HBRUSH            hBr;
HWND              hWndDlg;
RECT              rcMain;
short*            buffer;
char              temp;
char              sBuf[5];
char              szFile[256];
HBRUSH            brMain,brHW;
HKEY hKey;

void Error(char* error) {
        MessageBox(0,error,0,MB_ICONERROR);
        delete lpCom;
        exit(0);
}

void Status(char* status) {
        SetDlgItemText(hWndDlg,IDC_STATUS,status);
}

BOOL CALLBACK DialogProcHW(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
        RECT rPos;
        static POINT move,prev;
        switch(uMsg) {
        case WM_LBUTTONDOWN:
                GetCursorPos(&prev);
                SetCapture(hWnd);
                break;
        case WM_LBUTTONUP:
                ReleaseCapture();
                break;
        case WM_MOUSEMOVE:
                if ((wParam == MK_LBUTTON) && (GetCapture() == hWnd)) {
                        GetWindowRect(hWnd,&rPos);
                        GetCursorPos(&move);
                        SetWindowPos(hWnd,0,rPos.left+move.x-prev.x,rPos.top+move.y-prev.y,0,0,
                                SWP_NOSIZE | SWP_NOZORDER);
                        prev=move;
                }
                break;
        case WM_INITDIALOG:
        //      lpPic->SetHWTest(1);
                SetWindowPos(hWnd,HWND_TOPMOST,rcMain.right,rcMain.top,0,0,SWP_NOSIZE);
                break;
        case WM_CLOSE:
        //      lpPic->SetHWTest(0);
                EndDialog(hWnd,0);
                break;
        case WM_COMMAND:
                switch(LOWORD(wParam)) {
                case IDC_VDD:
                        lpPic->SetVDD(IsDlgButtonChecked(hWnd,IDC_VDD) & 0x1);
                        break;
                case IDC_DATAOUT:
                        lpPic->SetDataOut(IsDlgButtonChecked(hWnd,IDC_DATAOUT) & 0x1);
```

```
                        break;
                case IDC_CLOCK:
                        lpPic->SetClock(IsDlgButtonChecked(hWnd,IDC_CLOCK) & 0x1);
                        break;
                case IDC_EXIT:
        //              lpPic->SetHWTest(0);
                        EndDialog(hWnd,0);
                        break;
                }
        case WM_CTLCOLOREDIT:
        case WM_CTLCOLORDLG:
        case WM_CTLCOLORBTN:
        case WM_CTLCOLORSTATIC:
        case WM_CTLCOLORSCROLLBAR:
        case WM_CTLCOLORLISTBOX:
        case WM_CTLCOLORMSGBOX:
                HDC edc=(HDC)wParam;
                SetTextColor(edc,RGB(255,255,255));
                SetBkMode(edc,TRANSPARENT);
                return (int)brHW;
        }
        return 0;
}

void AddPath(char* szPath) {
        char temp[256];
        char line[2];
        line[1]=0;
        DWORD data;
        for(int i=1;i<=MAXPATH;i++) {
                data = 256;
                line[0] = i+0x30;
                if(RegQueryValueEx(hKey,line,0,0,(unsigned char*)temp,&data) != ERROR_SUCCESS) break;
                if(!strcmp(szPath,temp)) {
                        break;
                }
        }
        if(i>MAXPATH) i = MAXPATH;
        for(;i>1;i--) {
                data = 256;
                line[0] = i+0x30-1;
                if(RegQueryValueEx(hKey,line,0,0,(unsigned char*)temp,&data) != ERROR_SUCCESS) continue;
                line[0] = i+0x30;
                RegSetValueEx(hKey,line,0,REG_SZ,(unsigned char*)temp,strlen(temp));
        }
        RegSetValueEx(hKey,"1",0,REG_SZ,(unsigned char*)szPath,strlen(szPath));

}


BOOL CALLBACK DialogProcMain(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {
        HDROP hDrop;
        RECT rPos;
        HDC edc;
        DWORD data;
        static POINT move,prev;
        char szTemp[256];
        switch(uMsg) {
        case WM_LBUTTONDOWN:
                GetCursorPos(&prev);
                SetCapture(hWnd);
                break;
        case WM_LBUTTONUP:
                ReleaseCapture();
                break;
        case WM_MOUSEMOVE:
                if ((wParam == MK_LBUTTON) && (GetCapture() == hWnd)) {
                        GetWindowRect(hWnd,&rPos);
                        GetCursorPos(&move);
                        SetWindowPos(hWnd,0,rPos.left+move.x-prev.x,rPos.top+move.y-prev.y,0,0,SWP_NOSIZE | SWP_NO-
ZORDER);
                        prev=move;
                }
                break;
        case WM_INITDIALOG:
                hWndDlg=hWnd;
                lpPic = new CPic(lpCom,GetDlgItem(hWnd,IDC_PROGRESS));
                if(szFile[0]) lpHex->ReadHexFile(szFile);
                else RegQueryValueEx(hKey,"1",0,0,(unsigned char*)szFile,&data);
                SetDlgItemText(hWnd,IDC_COMBO,szFile);
                SetFocus(GetDlgItem(hWnd,IDC_OPEN));
                lpPic->bRun=0;
                SetEvent(lpPic->hIdle);
                SetEvent(lpPic->hNewData);
                break;
        case WM_CLOSE:
        //      delete lpPic;
        //      delete lpCom;
```

```
//      delete lpHex;
        exit(0);
        break;
case WM_DROPFILES:
        hDrop = (HDROP) wParam;
        DragQueryFile(hDrop,0,szTemp,256);
        DragFinish(hDrop);
        SetDlgItemText(hWnd,IDC_COMBO,szTemp);
        break;
case WM_COMMAND:
        switch(LOWORD(wParam)) {
        case IDC_EXIT:
                TerminateThread(lpPic->hThread,0);
                lpPic->bRun=0;
                SetEvent(lpPic->hIdle);
                lpPic->SetVDD(0);
                exit(0);
                break;
        case IDC_HW:
                GetWindowRect(hWnd,&rcMain);
                DialogBox(hIns,MAKEINTRESOURCE(IDD_HW),0,DialogProcHW);
                break;
        case IDC_ERASE_PROG:
                lpPic->EraseProg();
                break;
        case IDC_ERASE_CONFIG:
                lpPic->EraseConfig();
                break;
        case IDC_READ_PROG:
                lpHex->ClearBuffer();
                lpPic->ReadALL(buffer,READ_LEN);
                break;
        case IDC_READ_CONFIG:
                lpPic->ReadConfig();
                break;
        case IDC_PROG:
                lpPic->WriteProg(buffer);
                break;
        case IDC_COMBO:
                if(HIWORD(wParam)==CBN_DROPDOWN) {
                        SendDlgItemMessage(hWnd,IDC_COMBO,CB_RESETCONTENT,0,0);
                        char temp[256];
                        char line[2];
                        line[1]=0;
                        DWORD data;
                        for(int i=1;i<=MAXPATH;i++) {
                                data = 256;
                                line[0] = i+0x30;
                                if(RegQueryValueEx(hKey,line,0,0,(unsigned char*)temp,&data) !=
                                        ERROR_SUCCESS) break;
                                SendDlgItemMessage(hWnd,IDC_COMBO,CB_ADDSTRING,0,(long)temp);
                        }
                }
                else GetDlgItemText(hWnd,IDC_COMBO,szFile,256);
                break;
        case IDC_OPEN:
                GetDlgItemText(hWnd,IDC_COMBO,szFile,256);
                AddPath(szFile);
                lpHex->ReadHexFile(szFile);
                break;
        case IDC_PROG_E:
                lpPic->WriteProg_E(buffer);
                break;
        case IDC_PROG_CONFIG:
                lpPic->ProgConfig(buffer);
                break;
        case IDC_SKIP:
                TerminateThread(lpPic->hThread,0);
                lpPic->bRun=0;
                SetEvent(lpPic->hIdle);
                lpPic->SetVDD(0);
                SetEvent(lpPic->hIdle);
                SetEvent(lpPic->hNewData);
                break;
        }
case WM_CTLCOLOREDIT:
case WM_CTLCOLORLISTBOX:
        edc=(HDC)wParam;
        SetTextColor(edc,RGB(255,255,255));
        SetBkColor(edc,RGB(0x31,0,0x31));
        return (int)(hBr);
case WM_CTLCOLORDLG:
case WM_CTLCOLORBTN:
case WM_CTLCOLORSTATIC:
case WM_CTLCOLORSCROLLBAR:
case WM_CTLCOLORMSGBOX:
        edc=(HDC)wParam;
        SetTextColor(edc,RGB(255,255,255));
```

```
                SetBkMode(edc,TRANSPARENT);
                return (int)brMain;
        }
        return 0;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
        hBr=CreateSolidBrush(RGB(0x31,0,0x31));
        HBITMAP bMain,bHW;
        bMain=LoadBitmap(hInstance,MAKEINTRESOURCE(IDB_MAIN));
        bHW=LoadBitmap(hInstance,MAKEINTRESOURCE(IDB_HW));
        brMain=CreatePatternBrush(bMain);
        brHW=CreatePatternBrush(bHW);
        hIns=hInstance;
        buffer = new short[READ_LEN];
        lpHex = new CHex();
        buffer = (short*)lpHex->Buffer;
        lpCom = new CCom("COM1: baud=128000 parity=N data=8 stop=1");
        InitCommonControls();
        int len=strlen(lpCmdLine);
        if(len>3) {
                strcpy(szFile,lpCmdLine+1);
                szFile[len-2]=0;
        }
        else szFile[0]=0;

        DWORD coucou;
        RegCreateKeyEx(HKEY_LOCAL_MACHINE,"SOFTWARE\\PicProg",0,0,REG_OPTION_NON_VOLATILE,KEY_ALL_
ACCESS,0,&hKey,&coucou);

        DialogBox(hInstance,MAKEINTRESOURCE(IDD_MAIN),0,DialogProcMain);
        delete lpCom;
        delete lpPic;
        delete lpHex;
        return 0;
}
```

Hex.h

```
// Hex.h: interface for the CHex class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_HEX_H__909CDF37_0539_4D17_80E7_2342FD7AEB94__INCLUDED_)
#define AFX_HEX_H__909CDF37_0539_4D17_80E7_2342FD7AEB94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "windows.h"

class CHex
{
public:
        ClearBuffer();
        Error(char* error);
        unsigned char    ReadNextByte();
        unsigned char    ReadByte(DWORD pos=-1);
        char*    ReadHexFile(char* file);
        void     BuildHexFile(char* file, char* buffer,int* len);
        CHex();
        virtual ~CHex();

        HANDLE   hFile;
        char*    Buffer;
        DWORD    Index;
        unsigned char    Checksum;

};

#endif // !defined(AFX_HEX_H__909CDF37_0539_4D17_80E7_2342FD7AEB94__INCLUDED_)
```

Hex.cpp

```
// Hex.cpp: implementation of the CHex class.
//
//////////////////////////////////////////////////////////////////////

#include "Hex.h"
#include "windows.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
#define SAFE_CLOSE(a) if(a!=INVALID_HANDLE_VALUE) {CloseHandle(a),a=INVALID_HANDLE_VALUE;}
#define BUFFER_LEN 0xFFFF
void Status(char* status);

CHex::CHex() {
        hFile = INVALID_HANDLE_VALUE;
        Buffer=new char[BUFFER_LEN];
}

CHex::~CHex() {
        if(Buffer) delete Buffer;
        SAFE_CLOSE(hFile);
}

void CHex::BuildHexFile(char *file, char *buffer, int *len) {
}

char* CHex::ReadHexFile(char* file) {
        SAFE_CLOSE(hFile);
        hFile = CreateFile(file,
                GENERIC_READ,FILE_SHARE_READ | FILE_SHARE_WRITE,
                0,
                OPEN_EXISTING,
                FILE_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN,
                0);
        if(hFile == INVALID_HANDLE_VALUE) {
                char error[256];
                strcpy(error,"Can't open ");
                strcat(error,file);
                Error(error);
                SAFE_CLOSE(hFile);
                return 0;
        }

        DWORD size = GetFileSize(hFile,0);
        DWORD num;
        char temp;
        ClearBuffer();
        unsigned char strlen=0;
        unsigned int addr;
        unsigned int bufsize=0;
        Index=0;

        while(Index<size) {
                Checksum=0;
                temp=0;

                //Check the ':'
                SetFilePointer(hFile,Index,0,FILE_BEGIN);
                ReadFile(hFile,&temp,1,&num,0);
                Index++;
                if(temp!=':') break;

                //Get line lenght
                strlen = ReadNextByte();

                //Get Address
                addr = ReadNextByte() << 8;
                addr += ReadNextByte();

                //Get line type
                int i;
                char type = ReadNextByte();
                switch(type) {
                case 1:  //EOF reached
                        SAFE_CLOSE(hFile);
                        char status[64];
                        wsprintf(status,"Hex file succefully loaded (%u words)",bufsize/2);
                        Status(status);
                        return Buffer;

                case 0: //Data
                        //Save bytes into Buffer
                        for(i=0;i<strlen;i++) {
                                if(Index>=size) {
                                        Error("Invalid Hex File");
```

```
                                        SAFE_CLOSE(hFile);
                                        return 0;
                                }
                                *(Buffer+addr) = ReadNextByte();
                                addr++;
                                bufsize++;
                        }
                        break;
                default: // dont care
                        for(i=0;i<strlen;i++) {
                                ReadNextByte();
                        }
                        break;
                }
                //Get the checksum and compare
                Checksum=~--Checksum;
                if(ReadByte(-1) != Checksum) break;
                Index+=4;
        }
        Error("Invalid Hex File");
        SAFE_CLOSE(hFile);
        return 0;
}
unsigned char CHex::ReadByte(DWORD pos) {
        static DWORD CurrentIndex=0;

        if(hFile == INVALID_HANDLE_VALUE) return 0;

        if(pos==-1) pos=Index;
        if(CurrentIndex!=pos) {
                SetFilePointer(hFile,pos,0,FILE_BEGIN);
                CurrentIndex=pos;
        }

        char buf[2];
        DWORD num;
        ReadFile(hFile,buf,2,&num,0);
        CurrentIndex+=2;

        buf[0] -= buf[0] > 0x39 ? 0x37 : 0x30;
        buf[1] -= buf[1] > 0x39 ? 0x37 : 0x30;

        return ( ( buf[0] << 4 ) | buf[1]);
}
CHex::Error(char *error) {
        Status(error);
        MessageBox(0,error,0,MB_ICONERROR);
}
unsigned char CHex::ReadNextByte() {
        char ret = ReadByte();
        Checksum += ret;
        Index+=2;
        return ret;
}
CHex::ClearBuffer() {
        memset(Buffer,-1,BUFFER_LEN);
}
```

Com.h

```
// Com.h: interface for the CCom class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_COM_H__73CA3623_BFB5_4CE4_9C4D_8CDEFDEB5ECD__INCLUDED_)
#define AFX_COM_H__73CA3623_BFB5_4CE4_9C4D_8CDEFDEB5ECD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "windows.h"

typedef bool (CALLBACK* RXPROC)(unsigned char*,int);

class CCom
{
public:
        void RXBytePerByte(bool bpb);
        void InitRXProc(RXPROC proc);
        void SendByte(unsigned char* buffer,DWORD len);
        bool GetBit();
        SendBit(bool bit);
        static unsigned long __stdcall CTSloop(void* ptr);
        static unsigned long __stdcall RXThread(void* ptr);
        bool GetCTS() { return CTS; }

        SetTD(bool state);
        SetRTS(bool state);
        SetDTR(bool state);

        CCom(char* config);
        virtual ~CCom();

        HANDLE          hCom;
        bool            CTS;
        bool            quit;
        HANDLE          thread;
        RXPROC          RXCallBack;

        bool            byteperbyte;
};

#endif // !defined(AFX_COM_H__73CA3623_BFB5_4CE4_9C4D_8CDEFDEB5ECD__INCLUDED_)
```

Com.cpp

```
// Com.cpp: implementation of the CCom class.
//
//////////////////////////////////////////////////////////////////////

#include "Com.h"
#include "windows.h"
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

void Error(char* error);
void Status(char* status);

CCom::CCom(char* conf) {
        char config[50];
        byteperbyte=0;
        strcpy(config,conf);
        DCB     DCB;
        hCom = 0;
        quit = 0;
        RXCallBack=0;

        ZeroMemory(&DCB,sizeof(DCB));

        DCB.DCBlength=sizeof(DCB);
        BuildCommDCB(config,&DCB);
        DCB.StopBits=1;
        DCB.fBinary=1;

        for(unsigned char i=0;i<100;i++) {
                if(*(config+i)==0) break;
                if(*(config+i)==':') {
                        *(config+i) = 0;
                        break;
                }
        }

        hCom=CreateFile(config,
```

```
                                        GENERIC_READ | GENERIC_WRITE,
                                        0,
                                        0,
                                        OPEN_EXISTING,
                                        FILE_FLAG_OVERLAPPED,
                                        0);

        if(hCom == INVALID_HANDLE_VALUE) {
                char error[50];
                strcpy(error,"Can't open ");
                strcat(error,config);
        //      Error(error);
        }

        SetCommState(hCom,&DCB);
        SetupComm(hCom,0x1000,0x1000);

        SetCommMask(hCom,EV_CTS);

        COMMTIMEOUTS cto;
        cto.ReadIntervalTimeout=MAXDWORD;
        cto.ReadTotalTimeoutConstant=55555;
        cto.ReadTotalTimeoutMultiplier=MAXDWORD;
        cto.WriteTotalTimeoutConstant=5000;
        cto.WriteTotalTimeoutMultiplier=0;
        SetCommTimeouts(hCom,&cto);

        GetCommState(hCom,&DCB);

        DWORD Id;


        CTS=0;
//      thread=CreateThread(0,0,CTSloop,this,0,&Id);
        thread=CreateThread(0,0,RXThread,this,0,&Id);
}

CCom::~CCom() {
        quit=1;
        SetTD(0);
        SetDTR(0);
        SetRTS(0);
        DWORD exit;
        do {
                GetExitCodeThread(thread,&exit);
        } while(exit==STILL_ACTIVE);
        CloseHandle(hCom);
}
CCom::SetDTR(bool state) {
        if(state) EscapeCommFunction(hCom,SETDTR);
        else EscapeCommFunction(hCom,CLRDTR);
}

CCom::SetRTS(bool state) {
        if(state) EscapeCommFunction(hCom,SETRTS);
        else EscapeCommFunction(hCom,CLRRTS);
}

CCom::SetTD(bool state) {
        if(state) SetCommBreak(hCom);
        else ClearCommBreak(hCom);
}
unsigned long __stdcall CCom::RXThread(void* ptr) {
        CCom* lpCom=(CCom*)ptr;
        unsigned char RXBuffer[100];
        OVERLAPPED ov;
        DWORD readen;
        ZeroMemory(&ov,sizeof(ov));
        ov.hEvent=CreateEvent(0,0,0,0);
        while(1) {
                ReadFile(lpCom->hCom,RXBuffer,100,0,&ov);
                WaitForSingleObject(ov.hEvent,INFINITE);
                GetOverlappedResult(lpCom->hCom,&ov,&readen,0);
                if(!readen) continue;
                RXBuffer[readen]=0;

                if(lpCom->byteperbyte) {
                        for(unsigned int i=0;i<readen;i++) {
                                if(lpCom->RXCallBack) lpCom->RXCallBack(RXBuffer+i,1);
                        }
                }
                else {
                        if(lpCom->RXCallBack) lpCom->RXCallBack(RXBuffer,readen);
                }

        }
        CloseHandle(ov.hEvent);
        return 0;
```

```
}

unsigned long __stdcall CCom::CTSloop(void* ptr) {
        CCom* lpCom=(CCom*)ptr;
        DWORD mask;
        OVERLAPPED ov;
        ZeroMemory(&ov,sizeof(ov));
        ov.hEvent=CreateEvent(0,0,0,0);
        SetThreadPriority(lpCom->thread,THREAD_PRIORITY_HIGHEST);
        while(1) {
                WaitCommEvent(lpCom->hCom,&mask,&ov);
                WaitForSingleObject(ov.hEvent,INFINITE);
                if(lpCom->quit) return 0;
                if(mask==EV_CTS) {
                        lpCom->CTS^=1;
                        mask=0;
                }
        }
        CloseHandle(ov.hEvent);
        return 0;
}

CCom::SendBit(bool bit)
{
        SetRTS(1);
        SetDTR(bit);
        SetRTS(0);
}

bool CCom::GetBit()
{
        SetRTS(1);
        Sleep(2);
        bool bit = GetCTS();
        SetRTS(0);
        return bit;
}

void CCom::SendByte(unsigned char *buffer,DWORD len) {
        static OVERLAPPED ov = {0,0,0,0,0};
        static bool init=0;
        if(!init) {
                init=1;
                ov.hEvent = CreateEvent(0,0,0,0);
        }
        WriteFile(hCom,buffer,len,0,&ov);
        WaitForSingleObject(ov.hEvent,INFINITE);

}

void CCom::InitRXProc(RXPROC proc) {
        RXCallBack=proc;
}

void CCom::RXBytePerByte(bool bpb) {
        byteperbyte = bpb;
}
```

Pic.h

```
// Pic.h: interface for the CPic class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_PIC_H__83D3C7DE_0095_4A09_B1E0_5D3FAFE80DC1__INCLUDED_)
#define AFX_PIC_H__83D3C7DE_0095_4A09_B1E0_5D3FAFE80DC1__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "com.h"

#define   CMD                     0xC0
#define   DATA                    0x80

#define   LOAD_CONFIG             0x20 | CMD
#define   LOAD_DATA_FOR_PROG      0x22 | CMD
#define   READ_DATA_FROM_PROG     0x14 | CMD
#define   INC_ADDR                0x06 | CMD
#define   BEGIN_ERASE_PROG        0x08 | CMD
#define   BEGIN_PROG_ONLY         0x18 | CMD
#define   LOAD_DATA_FOR_DATA      0x23 | CMD
#define   READ_DATA_FROM_DATA     0x15 | CMD
#define   BULK_ERASE_SETUP1       0x01 | CMD
#define   BULK_ERASE_SETUP2       0x07 | CMD

#define   HW_VDDON                0x61
#define   HW_VDDOFF               0x41
#define   HW_DATAOUTON            0x62
#define   HW_DATAOUTOFF           0x42
#define   HW_CLOCKON              0x64
#define   HW_CLOCKOFF             0x44
#define   HW_ON                   0x68
#define   HW_OFF                  0x48

#define   RX_OK                   0x1
#define   ERROR_RX                0x2
#define   ERROR_READ              0x3


class CPic
{
public:
        ReadConfig();
        EraseProg();
        EraseConfig();
        WriteProg_E(short *buffer);
        void HandleData(unsigned char byte);
        void SetHWTest(bool state);
        void SetDataOut(bool state);
        void SetVDD(bool state);
        void SetClock(bool state);
        void HandleData(unsigned char* buffer, int len);
        static bool __stdcall RXProc(unsigned char* RX,int len);
        ProgConfig(short* buffer);
        static unsigned long __stdcall ReadALLThread(void* ptr);
        static unsigned long __stdcall WriteProgThread(void* ptr);
        ReadALL(short* buffer,int len);
        WriteProg(short* buffer);
        CPic(CCom* com,HWND hwndBar);
        virtual ~CPic();
        CCom*    lpCom;
        static   CPic*    lpPic;
        short*   Buffer;
        int              Len;
        volatile bool    bRun;
        int              TXLen;
        unsigned char    TXBuffer[200];
        unsigned char    InPtr;
        unsigned char    DataInBuf[2];

        volatile short   DataIn;

        volatile HANDLE hIdle;
        volatile HANDLE hNewData;

        DWORD    Id;
        HANDLE   hThread;
        HWND     hBar;
        bool     Prog_E;
};

#endif // !defined(AFX_PIC_H__83D3C7DE_0095_4A09_B1E0_5D3FAFE80DC1__INCLUDED_)
```

Pic.cpp

```
// Pic.cpp: implementation of the CPic class.
//
//////////////////////////////////////////////////////////////////////

#include "Pic.h"
#include "windows.h"
#include "commctrl.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
void Status(char* status);

CPic::CPic(CCom* com,HWND hwndBar) {
	hBar=hwndBar;
	lpCom=com;
	bRun=0;
	lpPic=this;
	InPtr=0;
	Buffer=0;
	lpCom->RXBytePerByte(1);
	lpCom->InitRXProc(RXProc);

	hIdle = CreateEvent(0,0,1,0);
	hNewData = CreateEvent(0,0,0,0);
/*
	lpPic->SetDataOut(0);
	lpPic->SetClock(0);
	lpPic->SetVDD(0);
	*/
}

CPic::~CPic() {
	CloseHandle(hIdle);
	CloseHandle(hNewData);
}
unsigned long __stdcall CPic::ReadALLThread(void* ptr) {

	CPic* lpPic=(CPic*)ptr;
	lpPic->bRun=1;
	lpPic->DataIn=0;

	lpPic->SetVDD(0);
	Sleep(1);
	lpPic->SetDataOut(0);
	lpPic->SetClock(0);
	Sleep(1);
	lpPic->SetVDD(1);

	SendMessage(lpPic->hBar,PBM_SETRANGE,0,MAKELPARAM(0,lpPic->Len));
	SendMessage(lpPic->hBar,PBM_SETPOS,0,0);

	for(int i=0;lpPic->bRun && i<lpPic->Len;i++) {
		ResetEvent(lpPic->hNewData);
		SendMessage(lpPic->hBar,PBM_SETPOS,i,0);
		char status[32];

		lpPic->HandleData(READ_DATA_FROM_PROG);


		WaitForSingleObject(lpPic->hNewData,INFINITE);
		if(!lpPic->bRun) {
			lpPic->SetVDD(0);
			return 0;
		}


		*(lpPic->Buffer + i) = lpPic->DataIn;
		wsprintf(status,"bytes read: %i (0x%04X)",i,lpPic->DataIn);
		Status(status);
		lpPic->HandleData(INC_ADDR);
	}
	lpPic->SetVDD(0);
	lpPic->bRun=0;

	return 0;
}

#define BUF_LEN 0x2000
unsigned long __stdcall CPic::WriteProgThread(void* ptr) {
	CPic* lpPic=(CPic*)ptr;
	lpPic->bRun=1;

	lpPic->Len=0;
	for(int i=BUF_LEN;i>=0;i--) {
		if(*(lpPic->Buffer+i) != -1) {
			lpPic->Len=i;
```

```
                        break;
                }
        }

        lpPic->SetDataOut(0);
        lpPic->SetClock(0);
        lpPic->SetVDD(1);

        SendMessage(lpPic->hBar,PBM_SETRANGE,0,MAKELPARAM(0,lpPic->Len));

        int             bwritten=0;
        char    status[128];
        short   temp;

        for(i=0;lpPic->bRun && i<=lpPic->Len;i++) {
                SendMessage(lpPic->hBar,PBM_SETPOS,i,0);
                temp=*(lpPic->Buffer+i);
                if(temp != -1) {
                        temp= temp << 8 | temp >> 8 | DATA;
                        lpPic->HandleData((unsigned char*)&temp,2);

                        lpPic->HandleData(LOAD_DATA_FOR_PROG);
                        lpPic->HandleData(lpPic->Prog_E == 1 ? BEGIN_ERASE_PROG : BEGIN_PROG_ONLY);
                        bwritten++;
                        wsprintf(status,"bytes written: %i (0x%04X)",bwritten,*(lpPic->Buffer+i));
                        Status(status);
                }
                lpPic->HandleData(INC_ADDR);
        }
        lpPic->bRun=0;
        lpPic->Buffer=0;
        lpPic->SetVDD(0);
        Status("Idle.");
        return 0;
}



CPic::ReadALL(short *buffer,int len) {
        if(!buffer || bRun) return 0;
        Buffer=buffer;
        Len=len;
        hThread=CreateThread(0,0,ReadALLThread,this,0,&Id);
        return 0;
}

CPic::WriteProg(short *buffer) {
        if(!buffer || bRun) return 0;
        Buffer=buffer;
        Prog_E=0;
        hThread=CreateThread(0,0,WriteProgThread,this,0,&Id);
        return 0;
}

CPic::WriteProg_E(short *buffer) {
        if(!buffer || bRun) return 0;
        Buffer=buffer;
        Prog_E=1;
        hThread=CreateThread(0,0,WriteProgThread,this,0,&Id);
        return 0;
}

CPic::EraseConfig()
{
        if(bRun) return 0;
        bRun = 1;

        SendMessage(lpPic->hBar,PBM_SETRANGE,0,MAKELPARAM(0,7));
        SendMessage(lpPic->hBar,PBM_SETPOS,0,0);

        SetVDD(0);
        SetDataOut(0);
        SetClock(0);
        Sleep(1);
        SetVDD(1);
        Sleep(1);
        SendMessage(lpPic->hBar,PBM_SETPOS,1,0);
        short temp=0x3FFF;
        temp = temp << 8 | temp >> 8 | DATA;
        lpPic->HandleData((unsigned char*)&temp,2);
        SendMessage(lpPic->hBar,PBM_SETPOS,2,0);

        HandleData(LOAD_CONFIG);
        for(int i=0;i<7;i++) {
                HandleData(INC_ADDR);
        }
        SendMessage(lpPic->hBar,PBM_SETPOS,3,0);
        lpPic->HandleData(BULK_ERASE_SETUP1);
```

```
        SendMessage(lpPic->hBar,PBM_SETPOS,4,0);
        lpPic->HandleData(BULK_ERASE_SETUP2);
        SendMessage(lpPic->hBar,PBM_SETPOS,5,0);
        lpPic->HandleData(BEGIN_ERASE_PROG);
        SendMessage(lpPic->hBar,PBM_SETPOS,6,0);
        lpPic->HandleData(BULK_ERASE_SETUP1);
        SendMessage(lpPic->hBar,PBM_SETPOS,7,0);
        lpPic->HandleData(BULK_ERASE_SETUP2);
        Status("Config Erased");
        SetVDD(0);
        bRun = 0;
        return 0;
}


CPic::EraseProg() {
        if(bRun) return 0;
        bRun = 1;

        SendMessage(lpPic->hBar,PBM_SETRANGE,0,MAKELPARAM(0,8));
        SendMessage(lpPic->hBar,PBM_SETPOS,0,0);

        SetVDD(0);
        SetDataOut(0);
        SetClock(0);
        Sleep(1);
        SetVDD(1);
        Sleep(1);
        SendMessage(lpPic->hBar,PBM_SETPOS,1,0);

        short temp=0x3FFF;
        temp = temp << 8 | temp >> 8 | DATA;
        lpPic->HandleData((unsigned char*)&temp,2);
        SendMessage(lpPic->hBar,PBM_SETPOS,2,0);
        lpPic->HandleData(LOAD_DATA_FOR_PROG);
        SendMessage(lpPic->hBar,PBM_SETPOS,3,0);
        lpPic->HandleData(BULK_ERASE_SETUP1);
        SendMessage(lpPic->hBar,PBM_SETPOS,4,0);
        lpPic->HandleData(BULK_ERASE_SETUP2);
        SendMessage(lpPic->hBar,PBM_SETPOS,5,0);
        lpPic->HandleData(BEGIN_ERASE_PROG);
        SendMessage(lpPic->hBar,PBM_SETPOS,6,0);
        lpPic->HandleData(BULK_ERASE_SETUP1);
        SendMessage(lpPic->hBar,PBM_SETPOS,7,0);
        lpPic->HandleData(BULK_ERASE_SETUP2);
        SetVDD(0);
        SendMessage(lpPic->hBar,PBM_SETPOS,8,0);
        Status("Program Erased");
        bRun = 0;
        return 0;
}

CPic::ProgConfig(short *buffer) {
        Buffer=buffer;
        bRun=1;
        SetDataOut(0);
        SetClock(0);
        SetVDD(1);

        HandleData(LOAD_CONFIG);
        for(int i=0;i<7;i++) {
                HandleData(INC_ADDR);
        }

        short temp=*(Buffer + 0x2007);
        char status[128];
        wsprintf(status,"PIC Configured: 0x%X",temp);
        temp = temp << 8 | temp >> 8 | DATA;
        HandleData((unsigned char*)&temp,2);
        HandleData(LOAD_DATA_FOR_PROG);
        HandleData(BEGIN_ERASE_PROG);

        Status(status);


        SetVDD(0);
        bRun=0;
        return 0;

}

CPic::ReadConfig() {
        bRun=1;
        SetDataOut(0);
        SetClock(0);
        SetVDD(1);
```

```
        HandleData(LOAD_CONFIG);
        for(int i=0;i<7;i++) {
                HandleData(INC_ADDR);
        }
        DataIn=0;
        ResetEvent(hNewData);
        HandleData(READ_DATA_FROM_PROG);

        WaitForSingleObject(lpPic->hNewData,INFINITE);
        if(!bRun) {
                SetVDD(0);
                return 0;
        }
        char status[128];
        wsprintf(status,"Configuration: 0x%X",DataIn);
        Status(status);
        bRun = 0;
        SetVDD(0);
        return 0;
}



CPic*    CPic::lpPic;
bool __stdcall CPic::RXProc(unsigned char *RX,int len) {
        if(lpPic->InPtr == 1) {
                lpPic->InPtr=0;
                *lpPic->DataInBuf = *RX;
                memcpy((void*)&lpPic->DataIn,lpPic->DataInBuf,2);
                SetEvent(lpPic->hNewData);
        }
        else if(*RX & DATA) {
                *(lpPic->DataInBuf+1) = *RX & ~DATA;
                lpPic->InPtr=1;
        }
        else {
                if(*RX == RX_OK) SetEvent(lpPic->hIdle);
                else if(*RX == ERROR_RX) lpPic->lpCom->SendByte(lpPic->TXBuffer,lpPic->TXLen);
        }
        return 0;
}

void CPic::HandleData(unsigned char *buffer, int len) {
        if(WaitForSingleObject(hIdle,100) == WAIT_TIMEOUT) {
                lpPic->bRun=0;
                SetEvent(lpPic->hIdle);
                SetEvent(lpPic->hNewData);
                TerminateThread(lpPic->hThread,0);
                Status("Timed Out...");
        }
        memcpy(TXBuffer,buffer,len);
        TXLen=len;
        ResetEvent(hIdle);
        lpCom->SendByte(TXBuffer,TXLen);
}

void CPic::SetClock(bool state) {
        HandleData(state == 1 ? HW_CLOCKON:HW_CLOCKOFF);
}

void CPic::SetVDD(bool state) {
        HandleData(state == 1 ? HW_VDDON:HW_VDDOFF);
}

void CPic::SetDataOut(bool state) {
        HandleData(state == 1 ? HW_DATAOUTON:HW_DATAOUTOFF);
}

void CPic::SetHWTest(bool state) {
        HandleData(state == 1 ? HW_ON:HW_OFF);
}

void CPic::HandleData(unsigned char byte) {
        HandleData(&byte,1);
}
```

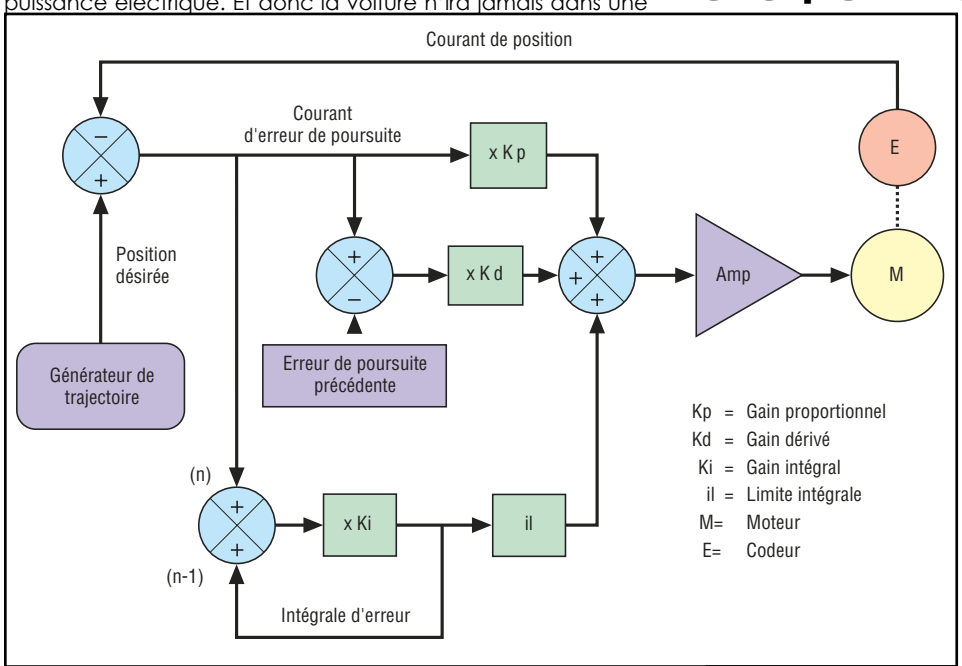
...:: LA VOITURE ::...

Introduction

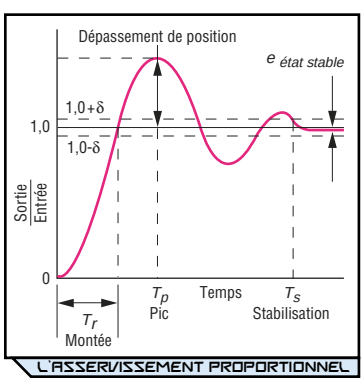
Nous allons traiter cette partie très rapidement, faute de temps, et tout le code source sera sans commentaire... Ces programmes sont loins d'être terminés. Comme le programmateur de PIC, nous avons un programme PIC et un programme PC à faire. Le programme PIC est codé en langage C et nous verrons comment utiliser nos modules. Nous verrons rapidement comment gérer la synchronisation du multi-threading de la télécommande. Nous verrons également ce qu'est le langage orienté objet avec ses classes. L'interface graphique est gérée par DirectX (et donc prenant en compte le blitter materiel, etc...). Puisque nous utilisons une transmission radio, il va falloir vérifier si les transferts de donnés se fond correctement (CRC). Pour gérer les trois capteurs de vitesse, les moteurs, le testeur de batterie, et les transmissions radios, il va falloir définir judicieusement nos prioritées.

L'alimentation des moteurs

Tout d'abord pour que la voiture avance tout droit, les moteurs devront tourner rigouresement à la même vitesse. A plus de 10 000 tr/min, avec les imperfections des moteurs, ils ne pourront pas tourner à la même vitesse si on leur donne la même puissance électrique. Et donc la voiture n'ira jamais dans une direction droite. C'est pourquoi au lieu de commander une puissance, nous allons commander une vitesse. On a donc recourt à l'asservissement en vitesse. Nous choisirons l'asservissement PID (Proportionnel, Intégral, Dérivé). Voyons de plus près ce que chaque boucle modifie dans le système. La boucle proportionnelle corrige avec un coefficient Kp l'erreur entre la vitesse voulue et la vitesse réelle. Si ce coefficient est trop fort, on dépasse la consigne, et donc le système oscille jusqu'à se stabiliser. On constate qu'une erreur dite statique subsiste, ainsi un simple asservissement P n'est pas suffisement fiable. On va rajouer une boucle d'intégration pour corriger cette erreur statique. Si on ne limite pas la valeur de notre intégrale, lorsque l'on impose une consigne que le système n'est pas capable d'atteindre, alors le tèrme intégral va augmenter sans cesse. Puis lorsque l'on voudra ralentir, onsera bloqué en vitesse maximale car le terme intégral sera bien trop grand (ce qui est d'ailleurs très dangereux vu qu'on pert le contrôle du véhicule). C'est pou cela qu'on limite l'intérgale. On constate également que lors d'un démarrage par exemple, le système mécanique a un certain temps de réation. Ainsi, lorsque l'on va appliquer une consigne l'intégrale va augmenter alors que le moteur n'a toujours pas réagit. C'est la que le phénomène d'overshoot, on dépasse violement la valeur limite. Alors ajoute le contrôle dérivé pour améliorer la réponse du système en régime transitoire. Donc on va gérer le PID de façon numérique dans la voiture. L'erreur sera la consigne à appliquer moins la vitesse réelle or les vitesses sont des valeurs algébriques, donc il faudra émuler le signe des vitesses des moteurs logicielement avec des algorythmes qui détectent le passage à zero d'une vitesse. En effet on ne passe pas de 100tr/min à -100tr/min d'un seul coup, on passe par des valeur proche de zero. Les coéfficients se règlent dynamiquement à partir de la télécommande. Les moteurs sont alimentés en PWM. Pour le PID nous utiliserons la résolution maximale qui est 10bits et pour le pilotage manuel des moteurs (donc des puissances) la résolution de 8bit.



L'ASSERVISSEMENT PID



L'ASSERVISSEMENT PROPORTIONNEL

Les capteurs de vitesse

La résolution des capteurs de vitesse des moteurs doit être la meilleure possible pour donner au PID la possiblitée de faire les meilleures corrections sur la puissance à fournir aux moteurs. Pour avoir la meilleur flexibilitée, nous alons utiliser les timer0 et timer1 en mode compteur pour déterminer la fréquence des signaux reçus, donc il n'y a plus que le timer2 pour gérer tout les timeouts de la transmission radio etc... La fréquence maximale des compteurs est de 10MHz. On utilisera des disques à 100 fentes que l'on fixera sur l'axe du moteur. On veut savoir quelle est la vitesse du véhicule avant que le PIC décroche pour savoir si on doit changer de méthode. La vitesse de rotation du moteur maximale avant que le PIC sature est: 60 * 10e6 / 100 = 6000000 tr/min, ce qui est largement assez. Maintenant voyons le temps d'echantillonage. On va utiliser les compteurs sur 8 bits (pour simplifier les calculs de multiplication avec les coefficients, etc...). donc 255 est la valeur limite de notre compteur. On va considerer un temps d'echantillonage de 25ms, et supposons que la vitesse maximale d'un moteur est de 15000 tr/min (en descente, avec de la marge) alors le compteur qui correspond à notre vitesse maximale est de 25e-3 * 15000/60 * 100 = 625. Pour des disques de 36 dents, la valeur maximale est 225. Donc soit on utilise des disques de 36 fentes, doit on utilise des 100 fentes avec une base de temps réglable (qu'il faudra émettre vers la télécommande) avec des seuils de vitesses qui fonctionnerai de manière analogue à une entrée trigger de schmitt.

Les freins

Les freins par résistance sont les freins par défaut, voire de sécurité. On peut inverser le sens de rotation des moteurs pour un freinage optimal. Cependant les roues risquent de ne plus adhérer, alors il faut prévoir un système d'ABS qui sera concu lors du test de la voiture puisque pour l'instant, il nous est impossible d'anticiper les réactions de la voiture.

La transmission

On utilise les modules USART du PIC pour la transmission radio à 4800bps. Le protocol est définit dans l'entête du code source. Il faut quand même savoir que la télécommande peut envoyer deux types d'informations: la configuration de la voiture, telle que les coefficients, ou encore les phares etc.... ou les données relative au déplacement de la voiture comme les consignes du PID ou la puissance des moteurs à fournir (pilotage manuel). L'envoie des données se fait en NRZ pour l'instant mais devrait se faire en encodage manchaster. En effet, pour que la transmission soit optimale, la moyenne de la valeur des bits envoyés doit être égale à 0.5 (c'est à dire qu'il y est autant de 0 que de 1) car les récepteurs possédent un ajusteur de gain automatique. Pour réaliser cet encodage, on envoie un octet sur deux octets. Il faut savoir que les modules Aurel se mettent en veille au bout d'un certain temps, ainsi les premiers bits d'un paquet sont erronés. c'est pourquoi on envoie l'octet àxFF avant chaque transmission (0x55 ou 0xAA à tester pour synchroniser les bits de start et stop). Autre point important : il faut définir une chaine qui va servir d'accroche pour la transmission. J'ai utilisé 0xC9,0x5E mais il semble que c'est bien trop léger. Il faudrai minimum 40bits car les pertubations environantes sont énormes. Mais c'est ici que la vitesse de 4800bps trouve ses limites, puisque le mieux serait d'avoir un aller-retour pour un échantillonage, en effet, en mettant plus de bits d'accroche, on augmante le temps à émettre un paquet et avec 16bits d'accroche, on depasse presque le temps d'un aller-retour sans prendre compte du temps de réaction du convertisseur USB/Série. Seul les tests nous montreront la bonne méthode à emplyer. Pour vérifier l'authenticité des données, on aurai pu employer un checksum (somme de tout les octets du paquet modulo 0xFF) mais lorsqu'un bit change dans le paquet et qu'en même temps, un bit change dans le checksum, il se peut qu'il y ai une grave erreur. Alors on utilise un CRC (cyclic redendancy check), si on modifie un seul bit du paquet, le CRC change complétement de valeur. C'est ce qui fait sa puissance par rapport au checksum. Pour calculer un CRC on peut soit passer par des polynômes qui nécessitent un temps de calcul, ou alors on peut utiliser des tables qui consomment de la mémoire ROM. Nous allons utiliser les tables.

CRC.h

```
const unsigned char CRCTable[256] = {
    0x84, 0x0D, 0x3C, 0xB4, 0xBC, 0xD5, 0x62, 0xC7, 0xA7, 0x76, 0xA4, 0xD7, 0x1F, 0xEB, 0x19, 0x12,
    0x72, 0x7A, 0xA9, 0x68, 0x08, 0xBB, 0xD6, 0x7F, 0x05, 0x9C, 0x65, 0x63, 0x9B, 0x8E, 0xCD, 0x10,
    0x54, 0xEE, 0x96, 0x47, 0xD9, 0x75, 0x8A, 0xCF, 0x79, 0x81, 0x4A, 0x98, 0xD0, 0x5E, 0xCA, 0xE0,
    0x6E, 0x6C, 0x23, 0x86, 0xE2, 0x2A, 0x8F, 0x7B, 0x64, 0xE9, 0x18, 0x32, 0x9F, 0x6D, 0x56, 0xDC,
    0xB3, 0xF7, 0x49, 0x88, 0x14, 0xF0, 0x60, 0x20, 0x28, 0x02, 0x77, 0x80, 0x5C, 0x6F, 0x4D, 0x71,
    0x53, 0xA8, 0xB7, 0xF4, 0x07, 0xF9, 0x61, 0xE4, 0xCC, 0xAA, 0x1A, 0x6A, 0x90, 0x2F, 0xED, 0x66,
    0xD8, 0x83, 0x9A, 0xD1, 0x30, 0x8B, 0x95, 0xA2, 0x0B, 0xB2, 0x3F, 0x22, 0x4B, 0x3E, 0x4C, 0x44,
    0x97, 0x50, 0xE3, 0x7C, 0xC2, 0x99, 0x93, 0x46, 0x45, 0xD2, 0xB0, 0xBE, 0x89, 0x73, 0x51, 0xBD,
    0xBA, 0x9D, 0x01, 0xFD, 0xC5, 0x0C, 0xF1, 0xCE, 0xEC, 0x3A, 0x37, 0xA0, 0xF2, 0x8C, 0xE6, 0x27,
    0x2E, 0x03, 0x67, 0xFE, 0x7E, 0x40, 0xE1, 0xC9, 0xEA, 0x36, 0x57, 0xDF, 0x0F, 0x39, 0x0E, 0x2B,
    0xD3, 0xEF, 0x48, 0x16, 0xA5, 0x55, 0x41, 0xDE, 0x43, 0xF5, 0x5B, 0x21, 0x00, 0x58, 0x33, 0x09,
    0xAD, 0x35, 0xA3, 0x06, 0xFF, 0x5D, 0xB9, 0xAC, 0x2C, 0x29, 0x31, 0xA6, 0x11, 0xA1, 0xBF, 0x94,
    0x59, 0x5A, 0xC8, 0x6B, 0xE8, 0x04, 0xDA, 0x92, 0x1C, 0x34, 0xC0, 0xB5, 0x1D, 0x1B, 0x87, 0xB6,
    0x0A, 0x78, 0x1E, 0xC4, 0xFA, 0xB8, 0x4E, 0x7D, 0x17, 0x24, 0xAF, 0xF3, 0xC1, 0xDB, 0x38, 0xDD,
    0xFC, 0x3D, 0xCB, 0x26, 0xF8, 0x42, 0x70, 0xC6, 0x91, 0x52, 0xB1, 0x2D, 0xE7, 0x5F, 0xC3, 0x69,
    0xD4, 0x82, 0x8D, 0x74, 0x15, 0xF6, 0x4F, 0xFB, 0xAE, 0xE5, 0x3B, 0xAB, 0x9E, 0x25, 0x13, 0x85,
    };
```

main.c

```
/*****************************************************************
/
/       Projet:   Voiture telecommandee (partie voiture)
/       Version: 0.1
/       Notes:    PIC Cadence a 20Mhz
/                 Emission/reception a 4800bps
/                 1 bit start, 1 bit stop, 0 bit parite
/                 un aller-retour = 42ms a 2400bps
/
/                 Protocol reception: (5-6 bytes)
/                 ------------------------------
/                         Packet par defaut
/                         ------------------
/                         - 0xFF (dummy)
/                         - 0xC9
/                         - 0x5E
/                         - flags
/                                 * polarite moteur gauche
/                                 * polarite moteur droit
/                                 * freins
/                                 * packet defaut
/                         - puissance (ou vitesse en PID) moteur gauche
/                         - puissance (ou vitesse en PID) moteur droit
/                         - CRC
/
/                         Packet config
/                         --------------
/                         - 0xFF (dummy)
/                         - 0xC9
/                         - 0x5E
/                         - flags
/                                 * lumieres
/                                 * phares
/                                 * PID
/                                 * ABS
/                                 * packet config
/                         - coefficient de proportionalite
/                         - coefficient d'integration
/                         - coefficient de derivation
```

```
/                       - CRC
/
/               Protocol emission: (7-9 bytes)
/               ------------------------------
/                       - 0xFF (dummy)
/                       - 0xC9
/                       - 0x5E
/                       - flags
/                               * direction (1 bit)
/                               * indicateur batterie (7 bits)
/                       - vitesse moteur gauche
/                       - vitesse moteur droit
/                       - vitesse roue arriere
/                       - puissance moteur gauche (uniquement en PID)
/                       - puissance moteur droit (uniquement en PID)
/                       - CRC
/
/               Configuration PIN:
/               ------------------
/                       - Entrees:
/                               RA0: Tension du testeur de batterie
/                               RA2: Vref-
/                               RA3: Vref+
/                               RA4: Capteur de vitesse moteur gauche
/                               RC0: Capteur de vitesse moteur droit
/                               RB0: Capteur de vitesse roue arriere
/                               RA5: Capteur de direction
/                               RC7: Recepteur radio
/                       - Sorties:
/                               RA1: Commande du testeur de batterie
/                               RB1: LED Power
/                               RB2: LED reception
/                               RB3: LED emmission
/                               RB4: LED mauvaise reception
/                               RB5: LED freinage
/                               RB6: Phares
/                               RB7: Lumieres
/                               RC5: !Freins par resistances
/                               RC1: Puissance du moteur droit
/                               RC2: Puissance du moteur gauche
/                               RC3: Polarite du moteur gauche
/                               RC4: Polarite du moteur droit
/                               RC6: Emmeteur radio
/************************************************************/
#include "pic.h" // PIC 16F876
#include "crc.h" // CRC-8 tables
__CONFIG(HS & WDTEN & PWRTEN & BOREN & \
LVPDIS & DPROT & WRTDIS & DEBUGDIS & UNPROTECT); // fuses config


#define bitno(adr,bit) ((unsigned)(&adr)*8+(bit))
#define bitset(var,bit) var |= 1 << bit
#define bitclr(var,bit) var &= ~(1 << bit)


//PINs defines
#define LED_POWER           RB1             // LED Power
#define LED_RX              RB2             // LED reception
#define LED_TX              RB3             // LED emmission
#define LED_BADPACKET       RB4             // LED mauvaise reception
#define LED_BRAKES          RB5             // LED freinage
#define LIGHTS              RB6             // Phares
#define CAR_LIGHTS          RB7             // Phares
#define BRAKES              RC5             // Freins par resistance
#define BAT_TESTER          RA1             // Commande du testeur de batterie
#define POL_L               RC3             // Polarite du moteur gauche
#define POL_R               RC4             // Polarite du moteur droit
#define DIRECTION           RA5             // Capteur de direction roue arriere

#define SET_BRAKES          { LED_BRAKES = 1; BRAKES = 0; }
#define CLR_BRAKES          { LED_BRAKES = 0; BRAKES = 1; }

volatile struct {
        unsigned        IDLE            : 1;
        unsigned        NEW_SPEED       : 1;
        unsigned        DIRECTION       : 1;
        unsigned        OVERSPEEDL      : 1;
        unsigned        OVERSPEEDR      : 1;
        unsigned        PID             : 1;
        unsigned        ABS             : 1;
        unsigned                        : 1;
} Flags;

volatile struct {
        unsigned        SENDING         : 1;
        unsigned        RX_CONFIG       : 1;
        unsigned        RX_TIMEOUT      : 1;
        unsigned        TMR250          : 1;
```

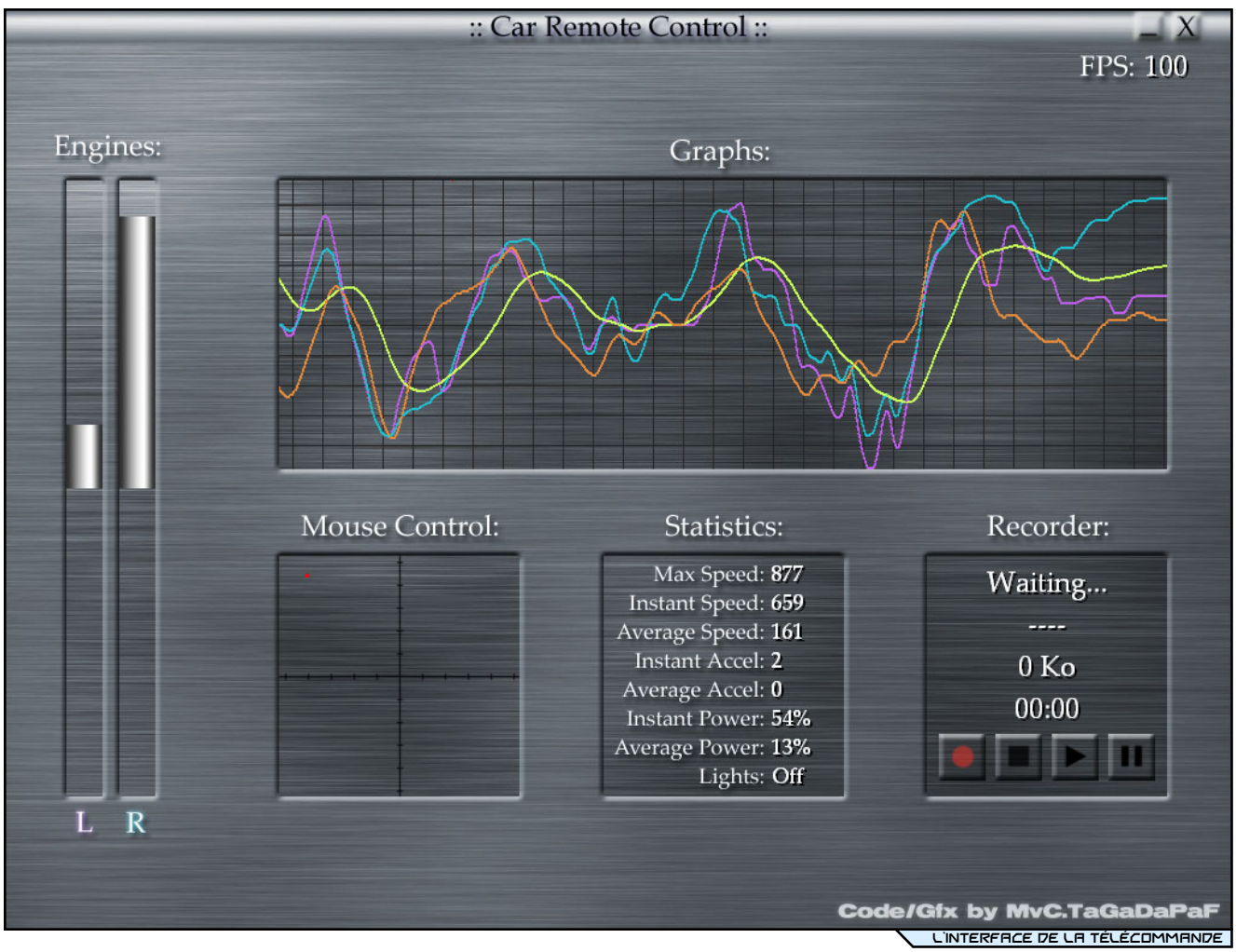
```
        unsigned                : 4;
} Flags2;

volatile unsigned char Battery=0;

volatile unsigned char SpeedL; // vitesse instantanee
volatile unsigned char SpeedR;
volatile unsigned char SpeedB;

volatile unsigned char ThSpeedL; // vitesse theoriques
volatile unsigned char ThSpeedR;

volatile unsigned char Kp;        // coeff
volatile unsigned char Ki;
volatile unsigned char Kd;

Init() {
        TRISA = 0b11111101;             // 1 = entree
        TRISB = 0b00000001;
        TRISC = 0b10000001;
        PORTA = 0;
        PORTB = 0;
        PORTC = 0;


        OPTION = 0b10101000;    // resistances de pull-up off
                                                        // TMR0 : RA4
                                                        // prescale 1

        ADCON0 = 0b10000000;    // Fosc/32
                                                        // Channel RA0
        ADCON1 = 0b10001111;    // RA0 en analogique
                                                        // RA2 Vref-
                                                        // RA3 Vref+
                                                        // le reste en numerique
        TMR1H = 0xFF;
        T1CON = 0b00000111;             // prescale 1
                                                        // not sync
                                                        // TMR1 = RC0

        PR2 = 0xFF;
        T2CON = 0b01100101;             // prescale 4
                                                        // postscale 12
        RCSTA = 0b00010000;
        TXSTA = 0b00100000;     // Low Speed
                                                        // Transmission enabled
        SPBRG = 64;                             // 20e6/(64*(64+1)) = 4808bps (0.16% d'erreur)

        CCP1CON = 0b00001100;   // PWM mode
        CCP2CON = 0b00001100;

        PIE1 = 0b01100011;              // ADIE | RCIE | TMR2IE | TMR1IE
        PIE2 = 0;
        INTCON = 0b01110000;    // PEIE | T0IE | INTE
}

main() {

        Init();
        SPEN = 1;
        GIE = 1;
        LED_POWER = 1;
        Flags.IDLE = 1;

        while(1) {
                asm("clrwdt");
                if(Flags.IDLE) {
                        SET_BRAKES;
                }
                else if(Flags.NEW_SPEED && Flags.PID) {
                        Flags.NEW_SPEED = 0;
                }
        }
}

interrupt isr() {
static unsigned char Timer25;
static unsigned char Timer250;
static unsigned char Count_SpeedB;
static unsigned char Ptr;
static unsigned char CRC;
static unsigned char RxBuffer[7];

        if(TXIF && TXIE) {
                if(Flags2.SENDING) {
                        unsigned char TX;
                        if(Ptr == 0) {
                                CRC = 0;
```

```
                        TX = 0xFF;
                }
                else if(Ptr == 1) TX = 0xC9;
                else if(Ptr == 2) TX = 0x5E;
                else if(Ptr == 3) {
                        bitclr(Battery,7);
                        if(Flags.DIRECTION) bitset(Battery,7);
                        TX = Battery;
                }
                else if(Ptr == 4) TX = SpeedL;
                else if(Ptr == 5) TX = SpeedR;
                else if(Ptr == 6) TX = SpeedB;
                else if(Ptr == 7) {
                        if(Flags.PID) TX = CCPR1H;
                        else {
                                TX = CRC;
                                Ptr = -1;
                                Flags2.SENDING = 0;
                        }
                }
                else if(Ptr == 8) TX = CCPR2H;
                else if(Ptr == 9) {
                        TX = CRC;
                        Ptr = -1;
                        Flags2.SENDING = 0;
                }
                Ptr++;
                if(Ptr) CRC ^= CRCTable[TX];
                TXREG = TX;
        }
        else {
                CREN = 1;
                TXIE = 0;
        }
}

if(RCIF) {
        Flags2.RX_TIMEOUT = 0;
        LED_RX = 1;
        if(FERR) {
                unsigned char RX;
                RX = RCREG;
                CREN = 0;
                CREN = 1;
                LED_BADPACKET = 1;
                LED_RX = 0;
                Ptr = 0;
        }
        else {
                unsigned char RX = RCREG;
                RxBuffer[Ptr] = RX;
                if(Ptr == 0) {
                        CRC = 0;
                        LED_RX = 0;
                        if(RX != 0xC9) Ptr--;
                }
                else if(Ptr == 1 && RX!= 0x5E)  {
                        LED_BADPACKET = 1;
                        LED_RX = 0;
                        Ptr = -1;
                }
                else if(Ptr == 2) {
                        Flags2.RX_CONFIG = 0;
                        if(bitno(RX,0) == 1) Flags2.RX_CONFIG = 1;
                }
                else if(Ptr == 5 && !Flags2.RX_CONFIG) {
                        Flags2.SENDING = 1;
                        TXIE = 1;
                        LED_RX = 0;
                        Ptr = -1;
                        if(RX != CRC) LED_BADPACKET = 1;
                        else {
                                if(bitno(RxBuffer[2],1)) SET_BRAKES
                                else CLR_BRAKES
                                if(bitno(RxBuffer[2],2)) POL_R = 1;
                                else POL_R = 0;
                                if(bitno(RxBuffer[2],3)) POL_L = 1;
                                else POL_L = 0;
                                if(Flags.PID) {
                                        ThSpeedL = RxBuffer[3];
                                        ThSpeedR = RxBuffer[4];
                                }
                                else {
                                        CCPR1L = RxBuffer[3];
                                        CCPR2L = RxBuffer[4];
                                }
                        }
                }
```

```
                        else if(Ptr == 6) {
                                Flags2.SENDING = 1;
                                TXIE = 1;
                                LED_RX = 0;
                                Ptr = -1;
                                if(RX != CRC) LED_BADPACKET = 1;
                                else {
                                        if(bitno(RxBuffer[2],1)) Flags.ABS = 1;
                                        else Flags.ABS = 0;
                                        if(bitno(RxBuffer[2],2)) Flags.PID = 1;
                                        else Flags.PID = 0;
                                        if(bitno(RxBuffer[2],3)) LIGHTS = 1;
                                        else LIGHTS = 0;
                                        if(bitno(RxBuffer[2],4)) CAR_LIGHTS = 1;
                                        else CAR_LIGHTS = 0;
                                        Kp = RxBuffer[3];
                                        Ki = RxBuffer[4];
                                        Kd = RxBuffer[5];
                                }
                        }
                        Ptr++;
                        if(Ptr) CRC ^= CRCTable[RX];
                }
        }

        if(INTF) {
                INTF = 0;
                Count_SpeedB++;
                if(Count_SpeedB == 0) Count_SpeedB--; //overflow handling
        }

        if(TMR0IF) {
                TMR0IF = 0;
                Flags.OVERSPEEDL = 1;
        }

        if(TMR1IF) {
                TMR1IF = 0;
                TMR1H = 0xFF;
                Flags.OVERSPEEDR = 1;
        }

        if(TMR2IF) { // 2.4576ms trigger
                TMR2IF = 0;

                if(!Flags2.SENDING) {
                        if(Flags2.RX_TIMEOUT == 1) {
                                if(Ptr) {
                                        Ptr = 0;
                                        CRC = 0;
                                        LED_RX = 0;
                                        LED_BADPACKET = 1;
                                }
                        }
                        Flags2.RX_TIMEOUT = 1;
                }

                Timer25++;
                Timer250++;
                if(Timer25 == 10) { // Speed trigger
                        Timer25 = 0;
                        SpeedL = TMR0;
                        TMR0 -= SpeedL;
                        SpeedR = TMR1L;
                        TMR1L -= SpeedR;
                        SpeedB = Count_SpeedB;
                        Count_SpeedB = 0;
                        if(Flags.OVERSPEEDL) {
                                Flags.OVERSPEEDL = 0;
                                SpeedL = 0xFF;
                        }
                        if(Flags.OVERSPEEDR) {
                                Flags.OVERSPEEDR = 0;
                                SpeedR = 0xFF;
                        }
                }
                if(Timer250 == 102) {
                        Flags2.TMR250 = 1;
                        Timer250 = 0;
                        LED_BADPACKET = 0;
                        if(Flags.IDLE) LED_POWER ^= 1;
                        else LED_POWER = 1;
                }
        }
}
```

Le logiciel PC

La télécommande doit être gérée en temps-réel avec les courbes de vitesses, de puissances, et d'accéleration en temps-réel. Le programme est composé de plusieurs threads. La thread principale, qui va receptionner les messages de l'utilisateur et redessiner constement la fenêtre. La thread qui va s'occuper de recevoir des données par le port série (et puisqu'on travaille en halt-duplex, on va devoir filtrer la réception parasite qui provient de l'emission, en effet le récepteur Aurel va capter ce que l'émetteur envoie). La thread qui va s'occuper de l'envoie des données avec tout les timings nécéssaires pour la détection de paquets perdus, etc... La thread qui va s'occuper du timing entre le compteur de FPS, le tracage des graphes (précis à la µs près grâce à certaine routine écrite en assembleur dans perf.cpp), et les statistiques. Et la thread qui va gérer le joystick. Lorsqu'on utilise des variables communes entre deux threads, il faut les déclarer 'volatile' sinon la valeur de la variable n'est pas rechagée dans les registres (EAX, EBX, etc...). Pour synchroniser les threads entre-elle, on va utiliser des 'event', des 'mutex' ou encore des 'semaphores'. un event est un évenement. Une thread peut attendre un evenement et reste en veille, dès qu'une aure thread va 'set' l'event, la thread qui attendait va être de nouveau active. Un mutex est commme un ticket que les thread se passent entre-elles. Un semaphore est comme un lot de tickets, une thread peut prendre un tiket, et le reposer. Si il n'y a plus de ticket, on attends qu'un se libère. Ceci est très utilisé lors de partages de ressources. Le C++ est un langage orienté objet. Un objet est créé dynamiquement à patir d'une classe qui peut être hierachisée. Dans notre programme, la classe CModule est la base de tout les modules de notre interface (graphe, barres des moteurs, etc...). Dans cette classe se trouve tout les objets de notre programme, ainsi ils peuvent s'interchanger des informations. Les classes CCom, CEngine, CFps, CGraph, CInput, CRecorder, CSpeed, CStats sont toutes des classes dérivées de CModules. Exemple d'objets : nous avons deux moteurs a piloter qui ont les même propriétés donc on utilise deux objects de CEngine : lpEngineL et lpEngineR. La classe CModule a pour constructeur par défaut l'initialiseur de sufraces DirectX. il intègre aussi la gestion de la souris par défaut, ainsi, si une classes dérivé ne veut pas utiliser les messages provenant souris, elle peut tout simplement les ignorer. C'est l'utilisation du mot-clef 'virtual'. D'ailleurs ces messages souris sont traités avant d'être dispachés au sein du programme puisque la fenêtre d'utilisation peut être resizer sans restrictions, donc il faut éffectuer une sorte de mise à l'echelle pour que chaque module puisse s'y retrouver. Par ailleurs, il faut que le programme traite quel est le module qui est sous la souris. Autre classe à venir : CBar et CButton, ce qui permettra de rendre la fenêtre plus animée. Par ailleurs, il faudra faire un emplacement pour les transmissions et pour le paramétrage de la voiture tel que les coefficients etc...



L'INTERFACE DE LA TÉLÉCOMMANDE

Main.cpp

```
#define WIN32_LEAN_AND_MEAN
#define INITGUID
#define WIDTH 1024
#define HEIGHT 768
#define LEFT 0
#define RIGHT 1
#include "windows.h"
#include "winbase.h"
#include "resource.h"
#include "stdlib.h"
#include "ddraw.h"
#include "engine.h"
#include "graph.h"
#include "stats.h"
#include "speed.h"
#include "recorder.h"
#include "fps.h"
#include "input.h"
#include "com.h"
#include "module.h"
#define SAFE_DELETE(a) if(a) {delete a,a=0;}
#define SAFE_RELEASE(a) if(a) {a->Release(),a=0;}
#define CREATE_SURFACE(a,b,c,d) ddsd.dwWidth=c, ddsd.dwHeight=d, lpDD->CreateSurface(&ddsd,&a,0),
LoadSurface(&a,&b,0,0);
#include "perf.h"
//--------------------------------------------------------------------------------
// Globals Variables
//--------------------------------------------------------------------------------
HWND                    hWnd=0;
HINSTANCE               hIns;
HCURSOR                 hArrow,hSize;
CModule*                lpModule=0;
RECT                    rcWin;
bool                    bWindowed=1,bInit=0,bRun=0;
LPDIRECTDRAW7           lpDD=0;
DDSURFACEDESC2          ddsd;
unsigned long           ulFpsCount=0;
float                   fWinRatio=1;
HANDLE                  hNextTick=0;
//--------------------------------------------------------------------------------
// Surfaces;
//--------------------------------------------------------------------------------
LPDIRECTDRAWSURFACE7 lpDDSPrimary=0;
LPDIRECTDRAWSURFACE7 lpDDSBack=0;
LPDIRECTDRAWSURFACE7 lpDDSBackGround=0;
//--------------------------------------------------------------------------------
// divround() : divide, and round
//--------------------------------------------------------------------------------
long divround(long num1,float num2) {
        _asm {
                FILD num1
                FLD num2
                FDIV
                FRNDINT
                FISTP num1
                WAIT
        }
        return num1;
}
//--------------------------------------------------------------------------------
// mulround() : divide, and round
//--------------------------------------------------------------------------------
long mulround(long num1,float num2) {
        _asm {
                FILD DWORD PTR num1
                FLD num2
                FMUL
                FRNDINT
                FISTP num1
                WAIT
        }
        return num1;
}
//--------------------------------------------------------------------------------
// Delete() Free memory !
//--------------------------------------------------------------------------------
void Delete(void) {
        bInit=0;
        bRun=0;
        WaitForSingleObject(hNextTick,INFINITE);
        lpModule->FreeMemory();
        SAFE_DELETE(lpModule);
        SAFE_RELEASE(lpDDSBackGround);
        SAFE_RELEASE(lpDDSBack);
        SAFE_RELEASE(lpDDSPrimary);
}
//--------------------------------------------------------------------------------
```

```
// Fatal Error !
//-------------------------------------------------------------------------------
void Error(char* error) {
        MessageBox(0,error,0,MB_ICONERROR);
        Delete();
        SAFE_RELEASE(lpDD);
        exit(0);
}
//-------------------------------------------------------------------------------
// LoadSurface(char *szBitmap) : Load a Bitmap in a surface from file
//-------------------------------------------------------------------------------
LPDIRECTDRAWSURFACE7 LoadSurface(char* szBitmap) {
        LPDIRECTDRAWSURFACE7      lpSurf=0;
        DDSURFACEDESC2            sDesc;
        HBITMAP                   hBitmap=0;
        BITMAP                    Bitmap;
        HDC                       hDCb=0,hDCs=0;
        char                      szBitPath[100];
        strcpy(szBitPath,"gfx\\");
        strcat(szBitPath,szBitmap);
        hBitmap=(HBITMAP)LoadImage(hIns,
                                   szBitPath,
                                   IMAGE_BITMAP,
                                   0,
                                   0,
                                   LR_LOADFROMFILE | LR_CREATEDIBSECTION);
        if(!hBitmap) {
                wsprintf(szBitPath,"Error loading %s",szBitmap);
                Error(szBitPath);
        }
        hDCb=CreateCompatibleDC(0);
        SelectObject(hDCb,hBitmap);
        GetObject(hBitmap,sizeof(BITMAP),&Bitmap);
        ZeroMemory(&sDesc,sizeof(sDesc));
        sDesc.dwSize              = sizeof(sDesc);
        sDesc.dwFlags             = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
        sDesc.dwWidth             = Bitmap.bmWidth;
        sDesc.dwHeight            = Bitmap.bmHeight;
        sDesc.ddsCaps.dwCaps= DDSCAPS_OFFSCREENPLAIN;
        if(lpDD->CreateSurface(&sDesc,&lpSurf,0)!=DD_OK) Error("Can't Create surface");
        lpSurf->GetDC(&hDCs);
        BitBlt(hDCs,0,0,sDesc.dwWidth,sDesc.dwHeight,hDCb,0,0,SRCCOPY);
        DeleteDC(hDCb);
        if(hDCs) lpSurf->ReleaseDC(hDCs);
        DeleteObject(hBitmap);
        return lpSurf;
}
//-------------------------------------------------------------------------------
// LoadSurface(int rsBitmap) : Load a Bitmap in a surface from ressource
//-------------------------------------------------------------------------------
LPDIRECTDRAWSURFACE7 LoadSurface(int rsBitmap) {
        LPDIRECTDRAWSURFACE7      lpSurf=0;
        DDSURFACEDESC2                    sDesc;
        HBITMAP                                   hBitmap=0;
        BITMAP                                    Bitmap;
        HDC                                               hDCb=0,hDCs=0;

        hBitmap=LoadBitmap(hIns,MAKEINTRESOURCE(rsBitmap));

        hDCb=CreateCompatibleDC(0);
        SelectObject(hDCb,hBitmap);
        GetObject(hBitmap,sizeof(BITMAP),&Bitmap);
        ZeroMemory(&sDesc,sizeof(sDesc));
        sDesc.dwSize              = sizeof(sDesc);
        sDesc.dwFlags             = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
        sDesc.dwWidth             = Bitmap.bmWidth;
        sDesc.dwHeight            = Bitmap.bmHeight;
        sDesc.ddsCaps.dwCaps= DDSCAPS_OFFSCREENPLAIN;
        if(lpDD->CreateSurface(&sDesc,&lpSurf,0)!=DD_OK) Error("Can't Create surface");
        lpSurf->GetDC(&hDCs);
        BitBlt(hDCs,0,0,sDesc.dwWidth,sDesc.dwHeight,hDCb,0,0,SRCCOPY);
        DeleteDC(hDCb);
        if(hDCs) lpSurf->ReleaseDC(hDCs);
        DeleteObject(hBitmap);
        return lpSurf;
}
//-------------------------------------------------------------------------------
// NextTick()
//-------------------------------------------------------------------------------
#define UPDATE 15000 //us...
unsigned long __stdcall NextTick(void* cool) {
        HANDLE hThread = GetCurrentThread();
        unsigned long time;
        while(bRun) {
                SetThreadPriority(hThread,THREAD_PRIORITY_TIME_CRITICAL);
                time = (DWORD)GetTicks();
                lpModule->NextTick();
                SetThreadPriority(hThread,THREAD_PRIORITY_NORMAL);
```

```
                DWORD wait = (DWORD)GetTicks()-time;
                if((DWORD)GetTicks()-time<UPDATE) {
                        ((CGraph*)lpModule->lpGraph)->sync=1;
                        while((DWORD)GetTicks()-time<=UPDATE) {
                                continue;
                        }
                }
                else ((CGraph*)lpModule->lpGraph)->sync=0;
        }
        return 0;
}
//-----------------------------------------------------------------------------
// Init()
//-----------------------------------------------------------------------------
void Init() {
        if(!bInit) return;
        lpDDSBackGround=LoadSurface(IDB_BACKGROUND);
        lpDDSBack->BltFast(0,0,lpDDSBackGround,0,DDBLTFAST_WAIT);
        if(!bWindowed) lpDDSPrimary->Blt(0,lpDDSBackGround,0,DDBLT_WAIT,0);

        lpModule=new CModule(lpDD,lpDDSBackGround,&fWinRatio);
        if(!lpModule) Error("A+");
        lpModule->lpDDSBack=lpDDSBack;
        lpModule->SetRectEngineL(53,143,83-53,655-143);
        lpModule->SetRectEngineR(97,143,127-97,655-143);
        lpModule->SetRectRecorder(769,455,969-769,655-455);
        lpModule->SetRectFps(950,31,65,30);
        lpModule->SetRectGraph(230,143,969-230,383-143);
        lpModule->SetRectStats(640,455,700-640,655-455);
        lpModule->SetRectSpeed(0,0,1,1);
        lpModule->SetRectInput(230,455,200,200);
        lpModule->SetRectCom(0,0,1,1);

        lpModule->lpRecorder=new CRecorder(hWnd);
        lpModule->lpFps=new CFps(&ulFpsCount);
        lpModule->lpEngineL=new CEngine(LEFT);
        lpModule->lpEngineR=new CEngine(RIGHT);
        lpModule->lpSpeed=new CSpeed();
        lpModule->lpStats=new CStats(739);
        lpModule->lpGraph=new CGraph();
        lpModule->lpInput=new CInput(&rcWin);
        lpModule->lpCom=new CCom("COM1: baud=4800 parity=N data=8 stop=1");

        bRun=1;
        unsigned long Id=0;
        hNextTick=CreateThread(0,0,NextTick,0,0,&Id);

}
//-----------------------------------------------------------------------------
// InitDD()
//-----------------------------------------------------------------------------
void InitDD() {
        if(bWindowed) {
                if(lpDD->SetCooperativeLevel(hWnd,DDSCL_NORMAL)!=DD_OK)
                        Error("Can't define Coop Level");
                ZeroMemory(&ddsd,sizeof(ddsd));
                ddsd.dwSize                             = sizeof(ddsd);
                ddsd.dwFlags                    = DDSD_CAPS;
                ddsd.ddsCaps.dwCaps             = DDSCAPS_PRIMARYSURFACE;
                if(lpDD->CreateSurface(&ddsd,&lpDDSPrimary,0)!=DD_OK)
                        Error("Can't Create Primary Surface");
                ddsd.dwFlags                    = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
                ddsd.dwWidth                    = WIDTH;
                ddsd.dwHeight                   = HEIGHT;
                ddsd.ddsCaps.dwCaps             = DDSCAPS_OFFSCREENPLAIN;
                if(lpDD->CreateSurface(&ddsd,&lpDDSBack,0)!=DD_OK)
                        Error("Can't Create Back Buffer Surface");
                LPDIRECTDRAWCLIPPER lpClipper=0;
                if(lpDD->CreateClipper(0,&lpClipper,0)!=DD_OK)
                        Error("Can't Create Clipper");
                if(lpClipper->SetHWnd(0,hWnd)!=DD_OK) Error("Can't Get hWnd");
                if(lpDDSPrimary->SetClipper(lpClipper)!=DD_OK)
                        Error("Can't Select Clipper");
                SAFE_RELEASE(lpClipper);
        }
        else {
                if(lpDD->SetCooperativeLevel(hWnd,DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN | DDSCL_ALLOWREBOOT)!=DD_OK)
                        Error("Can't define Coop Level");
                if(lpDD->SetDisplayMode(WIDTH,HEIGHT,32,0,0)!=DD_OK) {
                        if(lpDD->SetDisplayMode(WIDTH,HEIGHT,16,0,0)!=DD_OK)
                                Error("Can't Switch to 1024x768 (16,32)");
                }
                ZeroMemory(&ddsd,sizeof(ddsd));
                ddsd.dwSize                             = sizeof(ddsd);
                ddsd.dwFlags                    = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
                ddsd.dwBackBufferCount    = 1;
                ddsd.ddsCaps.dwCaps             = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP | DDSCAPS_COMPLEX;
                if(lpDD->CreateSurface(&ddsd,&lpDDSPrimary,0)!=DD_OK)
```

```
                        Error("Can't Create Primary Surface");
                DDSCAPS2 ddscaps;
                ZeroMemory(&ddscaps,sizeof(ddscaps));
                ddscaps.dwCaps=DDSCAPS_BACKBUFFER;
                if(lpDDSPrimary->GetAttachedSurface(&ddscaps,&lpDDSBack)!=DD_OK)
                        Error("Can't Create Back Buffer Surface");
                ZeroMemory(&rcWin,sizeof(RECT));
                fWinRatio=1;
        }
        bInit=1;
        return;
}
//-----------------------------------------------------------------------------
// SwitchMode()
//-----------------------------------------------------------------------------
void SwitchMode() {
        bInit=0;
        DestroyWindow(hWnd);
        Delete();
        bWindowed^=1;
        if(bWindowed) hWnd = CreateWindow("CarRemote",
                                          "Car Remote Control",
                                          WS_VISIBLE | WS_POPUP,
                                          (GetSystemMetrics(SM_CXSCREEN)-WIDTH)/2,
                                          (GetSystemMetrics(SM_CYSCREEN)-HEIGHT)/2,
                                          WIDTH,HEIGHT,
                                          0,
                                          0,
                                          hIns,
                                          0);
        else hWnd = CreateWindow("CarRemote",
                                 "Car Remote Control",
                                 WS_VISIBLE | WS_POPUP,
                                 (GetSystemMetrics(SM_CXSCREEN)-WIDTH)/2,
                                 (GetSystemMetrics(SM_CYSCREEN)-HEIGHT)/2,
                                 WIDTH,
                                 HEIGHT,
                                 0,
                                 0,
                                 hIns,
                                 0);
        InitDD();
        Init();
        if(bWindowed) SetWindowPos(hWnd,
                                 0,
                                 (GetSystemMetrics(SM_CXSCREEN)-WIDTH)/2,
                                 (GetSystemMetrics(SM_CYSCREEN)-HEIGHT)/2,
                                 WIDTH,
                                 HEIGHT,
                                 SWP_NOZORDER);
}
//-----------------------------------------------------------------------------
// RefreshScreen !
//-----------------------------------------------------------------------------
#define MAXFPS 1000/100
void Refresh() {
        static long last;
        if(!bInit) return;
        HRESULT hr;

        //Starting to blt every surfaces....
        lpModule->RefreshScreen();
        last=MAXFPS-GetTickCount()+last;
        if(last>MAXFPS) last=MAXFPS;
        else if(last<0) last=0;


        Sleep(last);
        while(true) {
                if(bWindowed) hr=lpDDSPrimary->Blt(&rcWin,lpDDSBack,0,DDBLT_WAIT,0);
                else hr=lpDDSPrimary->Flip(0,DDFLIP_WAIT | DDFLIP_NOVSYNC);
                //else hr=lpDDSPrimary->Blt(0,lpDDSBack,0,DDBLT_WAIT,0);
                if(hr==DDERR_SURFACELOST) {
                        lpDDSPrimary->Restore();
                        lpDDSBack->Restore();
                        continue;
                }
                if(hr!=DDERR_WASSTILLDRAWING) break;
                continue;
        }
        ulFpsCount++;
        last=GetTickCount();
}
//-----------------------------------------------------------------------------
// Clip();
//-----------------------------------------------------------------------------
bool Clip(int x,int y,RECT *rc,POINT *pt) {
        if(x >= rc->left && x <= rc->right && y>=rc->top && y<=rc->bottom) {
```

```
                pt->x=x-rc->left;
                pt->y=y-rc->top;
                return 1;
        }
        return 0;
}
//-------------------------------------------------------------------------------
// ProcessMouse();
//-------------------------------------------------------------------------------
void ProcessMouse(int x,int y,bool LButton,bool RButton,bool moving) {
        x=mulround(x,fWinRatio);
        y=mulround(y,fWinRatio);
        POINT pt;

        if(Clip(x,y,&lpModule->rcEngineL,&pt)) ((CEngine*)lpModule->lpEngineL)->ProcessMouse(pt.x,pt.y,LButton,RButto
n,moving);
        else if(Clip(x,y,&lpModule->rcEngineR,&pt)) ((CEngine*)lpModule->lpEngineR)->ProcessMouse(pt.x,pt.y,LButton,R
Button,moving);
        else if(Clip(x,y,&lpModule->rcGraph,&pt)) ((CGraph*)lpModule->lpGraph)->ProcessMouse(pt.x,pt.y,LButton,RButto
n,moving);
        else if(Clip(x,y,&lpModule->rcFps,&pt)) ((CFps*)lpModule->lpFps)->ProcessMouse(pt.x,pt.y,LButton,RButton,movi
ng);
        else if(Clip(x,y,&lpModule->rcInput,&pt)) ((CInput*)lpModule->lpInput)->ProcessMouse(pt.x,pt.y,LButton,RButto
n,moving);
        else if(Clip(x,y,&lpModule->rcRecorder,&pt)) ((CRecorder*)lpModule->lpRecorder)->ProcessMouse(pt.x,pt.y,LButt
on,RButton,moving);
        else if(Clip(x,y,&lpModule->rcStats,&pt)) ((CStats*)lpModule->lpStats)->ProcessMouse(pt.x,pt.y,LButton,RButto
n,moving);
}
//-------------------------------------------------------------------------------
// WindowProc
//-------------------------------------------------------------------------------
LRESULT CALLBACK WindowProc(HWND hWnd, unsigned uMsg, WPARAM wParam, LPARAM lParam) {
        static RECT rcpos,rPos;
        static bool bResizing=0;
        static POINT move,prev,pt;
    switch (uMsg) {
        case WM_COMMAND:
                if(LOWORD(wParam)==IDM_SWITCHMODE) {
                        SwitchMode();
                }
                break;
        case WM_LBUTTONDOWN:
                if (rPos.right-move.x<=20 && rPos.bottom-move.y<=20) {
                        bResizing=1;
                        SetCursor(hSize);
                        SetCapture(hWnd);
                        break;
                }
                GetCursorPos(&prev);
                if(bWindowed && HIWORD(lParam)<=30) SetCapture(hWnd);
                ProcessMouse(LOWORD(lParam),HIWORD(lParam),1,(wParam & MK_RBUTTON) ? 1:0,0);
                break;
        case WM_LBUTTONUP:
                if(bWindowed) {
                        ReleaseCapture();
                        bResizing=0;
                }
                prev.x=LOWORD(lParam);
                prev.y=HIWORD(lParam);
                if(prev.x>=973 && prev.x<=997 && prev.y>=3 && prev.y<=27) {
                        Delete();
                        PostMessage(hWnd,WM_QUIT,0,0);
                }
                else if(prev.x>=942 && prev.x<=966 && prev.y>=3 && prev.y<=27) {
                        ShowWindow(hWnd,SW_MINIMIZE);
                }
                ProcessMouse(LOWORD(lParam),HIWORD(lParam),0,(wParam & MK_RBUTTON) ? 1:0,0);
                break;
        case WM_RBUTTONDOWN:
                ProcessMouse(LOWORD(lParam),HIWORD(lParam),wParam & MK_LBUTTON,1,0);
                break;
        case WM_RBUTTONUP:
                ProcessMouse(LOWORD(lParam),HIWORD(lParam),wParam & MK_LBUTTON,0,0);
                break;
        case WM_MOUSEMOVE:
                pt.x=LOWORD(lParam);
                pt.y=HIWORD(lParam);
                GetCursorPos(&move);
                GetWindowRect(hWnd,&rPos);
                if(bWindowed) {
                        if (rPos.right-move.x<=200 && rPos.bottom-move.y<=200 || bResizing) {
                                SetCursor(hSize);
                                if (wParam == MK_LBUTTON) {
                                        if(!bResizing) SetCapture(hWnd);
                                        int xr,yr;
                                        if((move.x-rPos.left)*3/4 >= move.y-rPos.top) {
                                                xr = move.x-rPos.left;
```

```
                                        yr = (move.x-rPos.left)*3/4;
                                }
                                else {
                                        xr = (move.y-rPos.top)*4/3;
                                        yr = move.y-rPos.top;
                                }
                                if(xr <= 5) xr = 5;
                                if(yr <= 5) yr = 5;
                                SetWindowPos(hWnd,
                                                0,
                                                0,
                                                0,
                                                xr,yr,
                                                SWP_NOZORDER | SWP_NOMOVE);
                                GetWindowRect(hWnd,&rcWin);
                                fWinRatio=(float)WIDTH/(float)(rPos.right-rPos.left);
                                Refresh();
                                bResizing=1;
                        }
                }
                else if ((wParam == MK_LBUTTON) && (GetCapture() == hWnd)) {
                        SetWindowPos(hWnd,
                                0,
                                rPos.left+move.x-prev.x,
                                rPos.top+move.y-prev.y,
                                0,
                                0,
                                SWP_NOSIZE | SWP_NOZORDER);
                        prev=move;
                }
        }
        ProcessMouse(pt.x,
                        pt.y,
                        (bool)(wParam & MK_LBUTTON),
                        (wParam & MK_RBUTTON) ? 1:0,
                        1);
        break;
    case WM_MOVE:
        GetWindowRect(hWnd,&rcWin);
        break;
    case WM_CREATE:
        if(bWindowed) {
                rcpos.left=(GetSystemMetrics(SM_CXSCREEN)-WIDTH)/2;
                rcpos.top=(GetSystemMetrics(SM_CYSCREEN)-HEIGHT)/2;
                rcpos.right=rcpos.left+WIDTH-5;
                rcpos.bottom=rcpos.top+HEIGHT-5;
        }
        break;
    case WM_CLOSE:
        Delete();
        PostMessage(hWnd,WM_QUIT,0,0);
        break;
    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
  }
  return 1;
}
//------------------------------------------------------------------------------
// WinMain
//------------------------------------------------------------------------------
int WINAPI WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpszCmdLine,int nCmdShow) {
    hIns=hInstance;
    hArrow=LoadCursor(0,IDC_ARROW);
    hSize=LoadCursor(0,IDC_SIZENWSE);

    WNDCLASS wc;
    wc.style=CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
    //wc.style=CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc=WindowProc;
    wc.cbClsExtra=0;
    wc.cbWndExtra=sizeof(DWORD);
    wc.hInstance=hInstance;
    wc.hIcon=0;
    wc.hCursor=hArrow;
    wc.hbrBackground=0;
    wc.lpszMenuName=0;
    wc.lpszClassName="CarRemote";
    if(!RegisterClass(&wc)) Error("Can't Reg Class");
    hWnd = CreateWindow("CarRemote",
                        "Car Remote Control",
                        WS_VISIBLE | WS_POPUP,
                        (GetSystemMetrics(SM_CXSCREEN)-WIDTH)/2,
                        (GetSystemMetrics(SM_CYSCREEN)-HEIGHT)/2,
                        WIDTH,
                        HEIGHT,
                        0,
                        0,
                        hInstance,
```

```
                                    0);
        if(!hWnd) Error("Can't Create Window");
        HACCEL hAccel=LoadAccelerators(hInstance,MAKEINTRESOURCE(IDR_KEY));
        if(DirectDrawCreateEx(0,(void**)&lpDD, IID_IDirectDraw7 ,0)!=DD_OK) Error("Can't init DDraw");
        //if(DirectDrawCreateEx((GUID*)DDCREATE_EMULATIONONLY,(void**)&lpDD, IID_IDirectDraw7 ,0)!=DD_OK)
Error("Can't init DDraw");
        InitPerf();
        InitDD();
        Init();
        MSG msg;
        ShowWindow(hWnd,nCmdShow);
        UpdateWindow(hWnd);
        SetForegroundWindow(hWnd);
    while(true) {
        if(PeekMessage(&msg,0,0,0,PM_NOREMOVE)) {
            if(!(GetMessage(&msg,0,0,0))) break;
            if(!(TranslateAccelerator(hWnd,hAccel,&msg))) {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
        }
                 else {
                         Refresh();
                 }
        }

        Delete();
        SAFE_RELEASE(lpDD);
        DeleteObject(hArrow);
        DeleteObject(hSize);
        return 0;
}
```

Perf.h

```
#include "windows.h"
#ifndef PERF_INC
#define PERF_INC
LONGLONG GetTicks();
void StartBench();
void StopBench();
void InitPerf();
#endif
```

Perf.ccp

```
#include "windows.h"
#define BENCH
#ifdef BENCH
DWORD countertsc=1;
LONGLONG bench1=0;
DWORD bench2=0;
DWORD ulBench=0;
DWORD startEIP=0;
bool bBenchInit=0;
#ifdef _DEBUG
LONGLONG GetTicks() { //slow version, safe
        static LONGLONG tsc;
        _asm {
                RDTSC //read time stamp counter
                mov DWORD PTR tsc,eax
                mov DWORD PTR tsc+4,edx
        }
        return (LONGLONG)((ULONGLONG)tsc/(ULONGLONG)countertsc); //using __ualldiv()
}
#else
LONGLONG GetTicks() { //faster version, unsafe ?
        _asm {
                RDTSC
                push eax
                mov eax,edx
                xor edx,edx
                div dword ptr countertsc //div edx:eax with ecx
                mov ecx,eax
                pop eax
                div dword ptr countertsc
                mov edx,ecx
                ret
        }
        return 0; //dummy
}
#endif
void StartBench() {
        if(startEIP) return; //already running
        _asm {
                mov eax,[ebp+4] //get EIP
                mov dword ptr startEIP,eax
        }
        bench2=GetTickCount(); //store date
        bench1=GetTicks();
```

```
}
void StopBench() {
	static LONGLONG bench1bis=0;
	if(!startEIP) return; //not running
	bench1bis=GetTicks();
	bench2=GetTickCount()-bench2; //calculate delta
	static DWORD endEIP=0;
	if(!bBenchInit) { //used to calibrate
		startEIP=0;
		return;
	}
	_asm {
		mov eax,[ebp+4] //Get EIP
		sub eax,5
		mov dword ptr endEIP,eax
	}
	bench1bis-=ulBench; //
	if(bench1bis>=bench1) bench1=bench1bis-bench1;
	else bench1=0;
	static char szBench[80]; //not unsing stack
	wsprintf(szBench,
			"Start EIP: %08X\nEnd EIP: %08X\n\n%uµs\n%ums",
			startEIP,
			endEIP,
			(unsigned long)bench1,
			(unsigned long)bench2);
	startEIP=0; //not running anymore
	MessageBox(0,szBench,"BenchMark",0); //dispay bench result
}
#define BENCHPASS 5000
void CalibrateBench() { //get execution time for startbench() and stopbench()
	LONGLONG pass1,pass2;
	GetCurrentProcessId();
	pass1=GetTicks();
	for(unsigned int i=BENCHPASS;i>0;i--) {
		_asm nop
	}
	pass1=GetTicks()-pass1;
	pass2=GetTicks();
	for(i=BENCHPASS;i>0;i--) {
		StartBench();
		StopBench();
	}
	pass2=GetTicks()-pass2;
	ulBench=((DWORD)pass2-(DWORD)pass1)/BENCHPASS;
	bBenchInit=1;
}
void CalibrateTicks() {
	LONGLONG freq,count,count2;
	LONGLONG tsc1,tsc2;

	QueryPerformanceFrequency((LARGE_INTEGER*)&freq); //get a precise timer
	QueryPerformanceCounter((LARGE_INTEGER*)&count);
	_asm {
		RDTSC
		mov DWORD PTR tsc1,eax
		mov DWORD PTR tsc1+4,edx
	}
	Sleep(500); //wait a little while...
	QueryPerformanceCounter((LARGE_INTEGER*)&count2);
	_asm {
		RDTSC
		mov DWORD PTR tsc2,eax
		mov DWORD PTR tsc2+4,edx
	}
	count2-=count;
	tsc2-=tsc1;
	double tsc=((double)tsc2*(double)freq)/(double)count2; //calculate CPU clock
	tsc*=0.000001; //µs presision
	double show=0;
	_asm {
		fld qword ptr tsc
		fst show
		FRNDINT //round result
		fistp dword ptr countertsc
		wait //safe
	}
//	char szCPU[50]= "Found CPU Frequency: ";
//	gcvt(show,20,szCPU+21);
//	MessageBox(0,szCPU,"CPU frequency",0);
}
#endif
#ifdef BENCH
void InitPerf() {
	CalibrateTicks();
	CalibrateBench();
}
#endif
```

Module.h

```
// Module.h: interface for the CModule class.
//
//////////////////////////////////////////////////////////////////////
#include "ddraw.h"
#include "windows.h"



#if !defined(AFX_MODULE_H__6A5A26B9_16FB_4274_9647_74CD3B0B0747__INCLUDED_)
#define AFX_MODULE_H__6A5A26B9_16FB_4274_9647_74CD3B0B0747__INCLUDED_



#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000



class CModule
{
public:
        void FreeMemory();
        CModule(LPDIRECTDRAW7 lpDD_,LPDIRECTDRAWSURFACE7 lpSAppBack_,float* ratio);

        SetRectRecorder(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectFps(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectEngineL(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectEngineR(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectSpeed(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectStats(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectGraph(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectInput(unsigned int x,unsigned int y,unsigned int width,unsigned int height);
        SetRectCom(unsigned int x,unsigned int y,unsigned int width,unsigned int height);

        void RefreshScreen();
        virtual void ProcessMouse(int x,int y,bool LButton,bool RButton,bool moving) { };
        virtual void NextTick();
        static Init(CModule* ptr);
        CModule() { }
        virtual ~CModule();

        LPDIRECTDRAWSURFACE7              lpSBack;
        LPDIRECTDRAWSURFACE7              lpSurf;
        RECT                                              rc;
        bool                                              bUpdate;

        static LPDIRECTDRAWSURFACE7 lpDDSBack;
        static LPDIRECTDRAWSURFACE7       lpSAppBack;
        static LPDIRECTDRAW7              lpDD;

        static RECT                                       rcRecorder;
        static RECT                                       rcFps;
        static RECT                                       rcEngineL;
        static RECT                                       rcEngineR;
        static RECT                                       rcSpeed;
        static RECT                                       rcStats;
        static RECT                                       rcGraph;
        static RECT                                       rcInput;
        static RECT                                       rcCom;

        static CModule*                           lpEngineL;
        static CModule*                           lpFps;
        static CModule*                           lpEngineR;
        static CModule*                           lpGraph;
        static CModule*                           lpStats;
        static CModule*                           lpSpeed;
        static CModule*                           lpRecorder;
        static CModule*                           lpInput;
        static CModule*                           lpCom;

        static float*                             fWinRatio;


private:
        SetRect(RECT* rc,unsigned int x,unsigned int y,unsigned int width,unsigned int height);
};

#endif // !defined(AFX_MODULE_H__6A5A26B9_16FB_4274_9647_74CD3B0B0747__INCLUDED_)
```

Module.cpp

```
// Module.cpp: implementation of the CModule class.
//
//////////////////////////////////////////////////////////////////////

#include "Module.h"
#include "windows.h"


#define SAFE_DELETE(a) if(a) {delete a,a=0;}
#define SAFE_RELEASE(a) if(a) {a->Release(),a=0;}
LPDIRECTDRAWSURFACE7 LoadSurface(char* szBitmap);
void Error(char* error);

//Static definitions
LPDIRECTDRAW7                   CModule::lpDD;
LPDIRECTDRAWSURFACE7    CModule::lpDDSBack;
LPDIRECTDRAWSURFACE7    CModule::lpSAppBack;
RECT                                    CModule::rcRecorder;
RECT                                    CModule::rcFps;
RECT                                    CModule::rcEngineL;
RECT                                    CModule::rcEngineR;
RECT                                    CModule::rcSpeed;
RECT                                    CModule::rcStats;
RECT                                    CModule::rcGraph;
RECT                                    CModule::rcInput;
RECT                                    CModule::rcCom;

CModule*                                CModule::lpEngineL;
CModule*                                CModule::lpFps;
CModule*                                CModule::lpEngineR;
CModule*                                CModule::lpGraph;
CModule*                                CModule::lpStats;
CModule*                                CModule::lpSpeed;
CModule*                                CModule::lpRecorder;
CModule*                                CModule::lpInput;
CModule*                                CModule::lpCom;

float*                                  CModule::fWinRatio;


//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CModule::CModule(LPDIRECTDRAW7 lpDD_,LPDIRECTDRAWSURFACE7 lpSAppBack_,float* ratio) {
        fWinRatio=ratio;
        lpDD=lpDD_;
        lpSAppBack=lpSAppBack_;
}


CModule::~CModule() { }

CModule::SetRect(RECT* rc,unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        rc->left=x;
        rc->top=y;
        rc->right=x+width;
        rc->bottom=y+height;
}
CModule::SetRectRecorder(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcRecorder,x,y,width,height);
}
CModule::SetRectFps(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcFps,x,y,width,height);
}
CModule::SetRectEngineL(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcEngineL,x,y,width,height);
}
CModule::SetRectEngineR(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcEngineR,x,y,width,height);
}
CModule::SetRectSpeed(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcSpeed,x,y,width,height);
}
CModule::SetRectStats(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcStats,x,y,width,height);
}
CModule::SetRectGraph(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcGraph,x,y,width,height);
}
CModule::SetRectInput(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcInput,x,y,width,height);
}
CModule::SetRectCom(unsigned int x,unsigned int y,unsigned int width,unsigned int height) {
        SetRect(&rcCom,x,y,width,height);
```

```
}

CModule::Init(CModule* ptr) {
        RECT* rc;
        if(ptr==lpFps) rc=&rcFps;
        else if(ptr==lpRecorder) rc=&rcRecorder;
        else if(ptr==lpStats) rc=&rcStats;
        else if(ptr==lpSpeed) rc=&rcSpeed;
        else if(ptr==lpGraph) rc=&rcGraph;
        else if(ptr==lpEngineL) rc=&rcEngineL;
        else if(ptr==lpEngineR) rc=&rcEngineR;
        else if(ptr==lpInput) rc=&rcInput;
        else if(ptr==lpCom) rc=&rcCom;
        else return 0;
        DDSURFACEDESC2 ddsd;
        ZeroMemory(&ddsd,sizeof(ddsd));
        ddsd.dwSize                         = sizeof(ddsd);
        ddsd.dwFlags               = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
        ddsd.dwWidth               = rc->right-rc->left;
        ddsd.dwHeight              = rc->bottom-rc->top;
        ddsd.ddsCaps.dwCaps        = DDSCAPS_OFFSCREENPLAIN;
        if(lpDD->CreateSurface(&ddsd,&ptr->lpSBack,0)!=DD_OK) Error("Can't Create surface");
        if(lpDD->CreateSurface(&ddsd,&ptr->lpSurf,0)!=DD_OK) Error("Can't Create surface");
        ptr->lpSBack->Blt(0,lpSAppBack,rc,DDBLT_WAIT,0);
        ptr->lpSurf->Blt(0,lpSAppBack,rc,DDBLT_WAIT,0);
        return 0;
}

void CModule::RefreshScreen() {
        if(lpFps && lpFps->bUpdate) lpDDSBack->BltFast(rcFps.left,rcFps.top,lpFps->lpSurf,0,DDBLTFAST_WAIT);
                lpFps->bUpdate = 0 ;
        if(lpEngineL && lpEngineL->bUpdate) lpDDSBack->BltFast(rcEngineL.left,rcEngineL.top,lpEngineL-
>lpSurf,0,DDBLTFAST_WAIT);        lpEngineL->bUpdate = 0 ;
        if(lpEngineR && lpEngineR->bUpdate) lpDDSBack->BltFast(rcEngineR.left,rcEngineR.top,lpEngineR-
>lpSurf,0,DDBLTFAST_WAIT);        lpEngineR->bUpdate = 0 ;
        if(lpGraph && lpGraph->bUpdate) lpDDSBack->BltFast(rcGraph.left,rcGraph.top,lpGraph->lpSurf,0,DDBLTFAST_
WAIT);                    lpGraph->bUpdate = 0 ;
        if(lpStats && lpStats->bUpdate) lpDDSBack->BltFast(rcStats.left,rcStats.top,lpStats->lpSurf,0,DDBLTFAST_
WAIT);                    lpStats->bUpdate = 0 ;
        if(lpSpeed && lpSpeed->bUpdate) lpDDSBack->BltFast(rcSpeed.left,rcSpeed.top,lpSpeed->lpSurf,0,DDBLTFAST_
WAIT);                    lpSpeed->bUpdate = 0 ;
        if(lpRecorder && lpRecorder->bUpdate) lpDDSBack->BltFast(rcRecorder.left,rcRecorder.top,lpRecorder-
>lpSurf,0,DDBLTFAST_WAIT); lpRecorder->bUpdate = 0 ;
        if(lpInput && lpInput->bUpdate) lpDDSBack->BltFast(rcInput.left,rcInput.top,lpInput->lpSurf,0,DDBLTFAST_
WAIT);                    lpInput->bUpdate = 0 ;
        if(lpCom && lpCom->bUpdate) lpDDSBack->BltFast(rcCom.left,rcCom.top,lpCom->lpSurf,0,DDBLTFAST_WAIT);
                lpCom->bUpdate = 0 ;
}

void CModule::FreeMemory() {
        SAFE_DELETE(lpFps);
        SAFE_DELETE(lpRecorder);
        SAFE_DELETE(lpStats);
        SAFE_DELETE(lpSpeed);
        SAFE_DELETE(lpGraph);
        SAFE_DELETE(lpEngineL);
        SAFE_DELETE(lpEngineR);
        SAFE_DELETE(lpInput);
        SAFE_DELETE(lpCom);
}

void CModule::NextTick() {
        lpFps->NextTick();
        lpStats->NextTick();
        lpGraph->NextTick();
}
```

Com.h

```
// Com.h: interface for the CCom class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_COM_H__73CA3623_BFB5_4CE4_9C4D_8CDEFDEB5ECD__INCLUDED_)
#define AFX_COM_H__73CA3623_BFB5_4CE4_9C4D_8CDEFDEB5ECD__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "windows.h"
#include "ddraw.h"
#include "module.h"


typedef bool (CALLBACK* RXPROC)(unsigned char*,int);

class CCom : public CModule
{
public:
	bool SendConfig;
	void SendByte(char* buffer,DWORD len);
	static unsigned long __stdcall RXThread(void* ptr);
	static unsigned long __stdcall LoopThread(void* ptr);


	SetTD(bool state);
	SetRTS(bool state);
	SetDTR(bool state);

	CCom(char* config);
	virtual ~CCom();

	HANDLE		hCom;
	HANDLE		hRXThread;
	HANDLE		hLoopThread;
	HANDLE		hRX;
	HANDLE		hRXAck;

	unsigned char RXBuffer[256];
	volatile DWORD readen;
};

#endif // !defined(AFX_COM_H__73CA3623_BFB5_4CE4_9C4D_8CDEFDEB5ECD__INCLUDED_)
```

Com.cpp

```
// Com.cpp: implementation of the CCom class.
//
//////////////////////////////////////////////////////////////////////

#include "Com.h"
#include "crc.h"
#include "windows.h"
#include "engine.h"
#include "speed.h"
#include "perf.h"
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

void Error(char* error);
void Status(char* status);

CCom::CCom(char* conf) {
	lpCom=this;
	Init(this);
	char config[50];
	strcpy(config,conf);
	DCB	DCB;
	hCom = 0;
	SendConfig = 1;

	ZeroMemory(&DCB,sizeof(DCB));

	DCB.DCBlength=sizeof(DCB);
	BuildCommDCB(config,&DCB);



	for(unsigned char i=0;i<100;i++) {
		if(*(config+i)==0) break;
```

```
                if(*(config+i)==':') {
                        *(config+i) = 0;
                        break;
                }
        }

        hCom=CreateFile(config,
                                        GENERIC_READ | GENERIC_WRITE,
                                        0,
                                        0,
                                        OPEN_EXISTING,
                                        FILE_FLAG_OVERLAPPED,
                                        0);

        if(hCom == INVALID_HANDLE_VALUE) {
                char error[50];
                strcpy(error,"Can't open ");
                strcat(error,config);
                //Error(error);
        }

        SetupComm(hCom,0x1000,0x1000);
        SetCommState(hCom,&DCB);

        SetCommMask(hCom,EV_CTS);

        COMMTIMEOUTS cto;
        cto.ReadIntervalTimeout=MAXDWORD;
        cto.ReadTotalTimeoutConstant=55555;
        cto.ReadTotalTimeoutMultiplier=MAXDWORD;
        cto.WriteTotalTimeoutConstant=5000;
        cto.WriteTotalTimeoutMultiplier=0;
        SetCommTimeouts(hCom,&cto);

        DWORD Id;


        hRX = CreateEvent(0,0,0,0);
        hRXAck = CreateEvent(0,0,0,0);
        hRXThread=CreateThread(0,0,RXThread,this,0,&Id);
        hLoopThread=CreateThread(0,0,LoopThread,this,0,&Id);
}

CCom::~CCom() {
        SetTD(0);
        SetDTR(0);
        SetRTS(0);
        TerminateThread(hRXThread,0);
        TerminateThread(hLoopThread,0);
        WaitForSingleObject(hRXThread,INFINITE);
        WaitForSingleObject(hLoopThread,INFINITE);
        CloseHandle(hCom);
}
CCom::SetDTR(bool state) {
        if(state) EscapeCommFunction(hCom,SETDTR);
        else EscapeCommFunction(hCom,CLRDTR);
}

CCom::SetRTS(bool state) {
        if(state) EscapeCommFunction(hCom,SETRTS);
        else EscapeCommFunction(hCom,CLRRTS);
}

CCom::SetTD(bool state) {
        if(state) SetCommBreak(hCom);
        else ClearCommBreak(hCom);
}

unsigned long __stdcall CCom::RXThread(void* ptr) {
        CCom* lpCom=(CCom*)ptr;
        lpCom->readen=0;
        ZeroMemory(&lpCom->RXBuffer,sizeof(lpCom->RXBuffer));
        OVERLAPPED ov;
        DWORD readen;
        ZeroMemory(&ov,sizeof(ov));
        ov.hEvent=CreateEvent(0,0,0,0);
        SetThreadPriority(GetCurrentThread(),THREAD_PRIORITY_TIME_CRITICAL);
        while(1) {
                ReadFile(lpCom->hCom,lpCom->RXBuffer,256,0,&ov);
                WaitForSingleObject(ov.hEvent,INFINITE);
                GetOverlappedResult(lpCom->hCom,&ov,&readen,0);
                if(!readen) continue;
                lpCom->readen=readen;
                SetEvent(lpCom->hRX);
                WaitForSingleObject(lpCom->hRXAck,INFINITE);
        }
}
```

```
unsigned long __stdcall CCom::LoopThread(void* ptr) {
	CCom* lpCom=(CCom*)ptr;
	char TxBuffer[16];
	ZeroMemory(TxBuffer,sizeof(TxBuffer));
	while(1) {
		if(lpCom->SendConfig) {

			//WARNING !!!! REMOVE
			lpCom->SendConfig = 0;
			//WARNING !!!! REMOVE

			TxBuffer[0] = 0x1; //flags
			TxBuffer[1] = 0; //Kp
			TxBuffer[2] = 0; //Ki
			TxBuffer[3] = 0; //Kd
			lpCom->SendByte(TxBuffer,4);
		}
		else {
			int ThrottleL,ThrottleR;
			if(lpCom->lpEngineL) ThrottleL = ((CEngine*)lpCom->lpEngineL)->GetThrottle();
			if(lpCom->lpEngineR) ThrottleR = ((CEngine*)lpCom->lpEngineR)->GetThrottle();

			TxBuffer[0] = (ThrottleL >= 0 ? 0x8:0) | (ThrottleR >= 0 ? 0x4:0);
			TxBuffer[1] = (unsigned char) abs(ThrottleL);
			TxBuffer[2] = (unsigned char) abs(ThrottleR);
			//StartBench();
			lpCom->SendByte(TxBuffer,3);
			//StopBench();
		}

		if(WaitForSingleObject(lpCom->hRX,100) == WAIT_TIMEOUT) {
			bool tim = 0;
			continue;
		}
		SetEvent(lpCom->hRXAck);
	}
	return 0;
}

#define HEADER_LEN 3
void CCom::SendByte(char *buffer,DWORD len) {
	static OVERLAPPED ovw = {0,0,0,0,0};
	static OVERLAPPED ovr = {0,0,0,0,0};
	static bool init=0;
	static unsigned char Buffer[256] = { 0xFF, 0xC9, 0xE5 };
	if(!init) {
		init=1;
		ovw.hEvent = CreateEvent(0,0,0,0);
		ovr.hEvent = CreateEvent(0,0,0,0);
	}
	unsigned char CRC = CRCTable[0xC9] ^ CRCTable[0xE5];
	for(unsigned int i=0;i<len;i++) {
		Buffer[HEADER_LEN + i] = *(buffer+i);
		CRC ^= CRCTable[*(buffer+len)];
	}
	Buffer[HEADER_LEN + i] = CRC;
	len += HEADER_LEN+1;
	WriteFile(hCom,Buffer,len,0,&ovw);
	WaitForSingleObject(ovw.hEvent,INFINITE);

	Buffer[0]=0;
	// Flush RX Buffer
	DWORD readeni=0;
	while(readeni<len) {
		if(WaitForSingleObject(hRX,50) == WAIT_TIMEOUT) {
			break;
		}
		readeni+=readen;
		SetEvent(hRXAck);
	}
}
```

Engine.h

```
#if !defined(AFX_ENGINE_H__132BD222_4FE9_414F_B4EB_4BC6849EDFD7__INCLUDED_)
#define AFX_ENGINE_H__132BD222_4FE9_414F_B4EB_4BC6849EDFD7__INCLUDED_
#include "windows.h"
#include "ddraw.h"
#include "module.h"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CEngine : public CModule
{
public:
        CEngine(char Left_Right);
        virtual ~CEngine();
        int GetThrottle()                       { return Throttle; }
        void SetThrottle(int throttle);
        virtual void ProcessMouse(int x,int y,bool LButton,bool RButton,bool moving);
        bool    draw;
private:
        LPDIRECTDRAWSURFACE7            lpSFullBar;
        int                             Throttle;
};

#endif // !defined(AFX_ENGINE_H__132BD222_4FE9_414F_B4EB_4BC6849EDFD7__INCLUDED_)
```

Engine.cpp

```
#include "Engine.h"
#include "resource.h"
#ifndef LEFT
#define LEFT 0
#endif
#ifndef RIGHT
#define RIGHT 1
#endif
#define RELEASE(a) if(a) {a->Release(),a=0;}

LPDIRECTDRAWSURFACE7 LoadSurface(int rsBitmap);
void Error(char* error);
CEngine::CEngine(char Left_Right) {
        lpSBack=0;
        LPDIRECTDRAWSURFACE7     lpSBar;
        //lpSBar=LoadSurface("engine_pattern.bmp");
        lpSBar=LoadSurface(IDB_ENGINE_PATTERN);
        draw=1;
        if(Left_Right) lpEngineR=this;
        else lpEngineL=this;
        Init(this);
        DDSURFACEDESC2 ddsd;
        ZeroMemory(&ddsd,sizeof(ddsd));
        ddsd.dwSize                     = sizeof(ddsd);
        ddsd.dwFlags            = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
        ddsd.dwWidth            = 30;
        ddsd.dwHeight           = 256;
        ddsd.ddsCaps.dwCaps     = DDSCAPS_OFFSCREENPLAIN;
        if(lpDD->CreateSurface(&ddsd,&lpSFullBar,0)!=DD_OK) Error("Can't Create surface");
        Throttle=1;
        SetThrottle(0);
        for(int i=0;i<512;i++) lpSFullBar->BltFast(0,i,lpSBar,0,DDBLTFAST_WAIT);
        RELEASE(lpSBar);
}
CEngine::~CEngine() {
        RELEASE(lpSFullBar);
        RELEASE(lpSurf);
        RELEASE(lpSBack);
}
void CEngine::SetThrottle(int throttle) {
        if(Throttle==throttle) return;
        if(throttle>256) throttle=256;
        else if(throttle<-256) throttle=-256;
        Throttle=throttle;
        lpSurf->Blt(0,lpSBack,0,DDBLT_WAIT,0);
        if(throttle!=0) {
                static RECT rc1 = {0,0,30,0};
                rc1.bottom=abs(throttle);
                if(throttle>0) lpSurf->BltFast(0,256-throttle,lpSFullBar,&rc1,DDBLTFAST_WAIT);
                else lpSurf->BltFast(0,256,lpSFullBar,&rc1,DDBLTFAST_WAIT);
        }
        bUpdate = 1;
        return;
}
void CEngine::ProcessMouse(int x,int y,bool LButton,bool RButton,bool moving) {
        if (LButton && !moving) draw^=1;
}
```

Fps.h

```
// Fps.h: interface for the CFps class.
//
//////////////////////////////////////////////////////////////////////
#if !defined(AFX_FPS_H__3372B201_123D_11D9_8D11_D8EF24B26B71__INCLUDED_)
#define AFX_FPS_H__3372B201_123D_11D9_8D11_D8EF24B26B71__INCLUDED_

#include "windows.h"
#include "ddraw.h"
#include "module.h"
#define COUNT 20

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CFps : public CModule
{
public:
        CFps(unsigned long* counter);
        virtual ~CFps();
        virtual void NextTick();

        unsigned long*                  ulFpsCount;
        HFONT                                   hFont;
        unsigned long                   ulCount[COUNT+1][2];
};

#endif // !defined(AFX_FPS_H__3372B201_123D_11D9_8D11_D8EF24B26B71__INCLUDED_)
```

Fps.cpp

```
// Fps.cpp: implementation of the CFps class.
//
//////////////////////////////////////////////////////////////////////

#include "Fps.h"
#define RELEASE(a) if(a) {a->Release(),a=0;}
void Error(char* error);
#define FPS_X 950
#define FPS_Y 34
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CFps::CFps(unsigned long* counter) {
        lpFps=this;
        ulFpsCount=counter;
        Init(this);
        hFont = CreateFont(32,0,0,0,0,0,0,0,0,0,
                                OUT_TT_PRECIS,
                                CLIP_TT_ALWAYS,
                                ANTIALIASED_QUALITY,
                                FF_DONTCARE,
                                "Palatino Linotype");
        ulCount[COUNT][0]=0;
}

CFps::~CFps() {
        RELEASE(lpSurf);
        RELEASE(lpSBack);
        DeleteObject(hFont);
}

void CFps::NextTick() {
        float fFps;
        char szFps[10];
        unsigned char len;
        static unsigned int index=COUNT;

        index++;
        if(index>COUNT) index=0;


        ulCount[index][0]=GetTickCount();
        ulCount[index][1]=*ulFpsCount;
        *ulFpsCount=0;

        lpSurf->Blt(0,lpSBack,0,0,0);
        if(ulCount[COUNT][0]==0) return;
        unsigned long Fps=0;
        unsigned long time=0;

        for(int i=0;i<COUNT;i++) Fps+=ulCount[i][1];
        Fps*=1000;
```

```
        if(index==COUNT) fFps=(float)Fps/(float)(ulCount[COUNT][0]-ulCount[0][0]);
        else fFps=(float)Fps/(float)(ulCount[index][0]-ulCount[index+1][0]);


        if(fFps>=100) gcvt(fFps,4,szFps);
        else if(fFps>=10) gcvt(fFps,3,szFps);
        else gcvt(fFps,2,szFps);
        len=strlen(szFps);
        if(*(szFps+len-1)=='.') *(szFps+len-1)=0;


        HDC hDCFps=0;
        lpSurf->GetDC(&hDCFps);
        if(hDCFps) {
                SelectObject(hDCFps,hFont);
                SetBkMode(hDCFps,TRANSPARENT);
                SetTextColor(hDCFps,RGB(0,0,0));
                TextOut(hDCFps,2,2,szFps,strlen(szFps));
                SetTextColor(hDCFps,RGB(255,255,255));
                TextOut(hDCFps,0,0,szFps,strlen(szFps));
                lpSurf->ReleaseDC(hDCFps);
                bUpdate = 1;
        }
}
```

Graph.h

```
// Graph.h: interface for the CGraph class.
//
//////////////////////////////////////////////////////////////////////


#if !defined(AFX_GRAPH_H__6DAE7046_C661_461C_A10B_F6B752DA9837__INCLUDED_)
#define AFX_GRAPH_H__6DAE7046_C661_461C_A10B_F6B752DA9837__INCLUDED_

#include "windows.h"
#include "ddraw.h"
#include "Engine.h"
#include "stats.h"
#include "module.h"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGraph : public CModule
{
public:
        CGraph();
        virtual void NextTick();
        virtual ~CGraph();

        int                     Width;

        LPDIRECTDRAWSURFACE7    lpS1;
        LPDIRECTDRAWSURFACE7    lpS2;
        LPDIRECTDRAWSURFACE7    lpSPattern;
        LPDIRECTDRAWSURFACE7    lpSWarning;

        bool                    sync;

        HPEN                    Pen1;
        HPEN                    Pen2;
        HPEN                    Pen3;
        HPEN                    Pen4;

};

#endif // !defined(AFX_GRAPH_H__6DAE7046_C661_461C_A10B_F6B752DA9837__INCLUDED_)
```

Graph.cpp

```
// Graph.cpp: implementation of the CGraph class.
//
//////////////////////////////////////////////////////////////////////

#include "Graph.h"
#include "ddraw.h"
#include "stats.h"
#include "resource.h"
//#include "Engine.h"
#define RELEASE(a) if(a) {a->Release(),a=0;}
#define UPDATE 10
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
LPDIRECTDRAWSURFACE7 LoadSurface(int rsBitmap);
void Error(char* error);
unsigned long __stdcall Graphing(void* graph);


CGraph::CGraph() {
        lpGraph=this;
        sync=1;
        Width = rcGraph.right-rcGraph.left;

        lpSPattern=0;
        lpS1=0;
        lpS2=0;
        lpSWarning=0;
        //lpSPattern=LoadSurface("graph_pattern.bmp");
        lpSPattern=LoadSurface(IDB_GRAPH_PATTERN);
        Init(this);

        DDSURFACEDESC2 ddsd;
        ZeroMemory(&ddsd,sizeof(ddsd));
        ddsd.dwSize                 = sizeof(ddsd);
        ddsd.dwFlags                = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;
        ddsd.dwWidth                = Width;
        ddsd.dwHeight               = 240;
        ddsd.ddsCaps.dwCaps         = DDSCAPS_OFFSCREENPLAIN;
        if(lpDD->CreateSurface(&ddsd,&lpS1,0)!=DD_OK) Error("Can't Create surface");
        if(lpDD->CreateSurface(&ddsd,&lpS2,0)!=DD_OK) Error("Can't Create surface");
        ddsd.dwHeight               = 60;
    if(lpDD->CreateSurface(&ddsd,&lpSWarning,0)!=DD_OK) Error("Can't Create surface");

    DDBLTFX fx;
    ZeroMemory(&fx,sizeof(fx));
    fx.dwSize=sizeof(fx);
    fx.dwFillColor=RGB(255,0,255);
    lpS1->Blt(0,0,0,DDBLT_WAIT | DDBLT_COLORFILL,&fx);
    lpS2->Blt(0,0,0,DDBLT_WAIT | DDBLT_COLORFILL,&fx);

    HFONT hFont = CreateFont(57,0,0,0,0,0,0,0,0,
                                OUT_TT_PRECIS,
                                CLIP_TT_ALWAYS,
                                ANTIALIASED_QUALITY,
                                FF_DONTCARE,
                                "Palatino Linotype");
    HDC dc;
         RECT rect={0,0,Width,60};
         lpSWarning->Blt(0,lpSBack,&rect,DDBLT_WAIT,0);
    lpSWarning->GetDC(&dc);
    SelectObject(dc,hFont);
         SetBkMode(dc,TRANSPARENT);
         SetTextColor(dc,RGB(0,0,0));
         TextOut(dc,22,2,"WARNING: Synchonisation lost",28);
         SetTextColor(dc,RGB(255,255,255));
         TextOut(dc,20,0,"WARNING: Synchonisation lost",28);
         lpSWarning->ReleaseDC(dc);
         DeleteObject(hFont);


         DDPIXELFORMAT format;
         ZeroMemory(&format,sizeof(format));
         format.dwSize=sizeof(format);
         lpS1->GetPixelFormat(&format);

         DDCOLORKEY ck;
         ck.dwColorSpaceHighValue=format.dwRBitMask | format.dwBBitMask;
         ck.dwColorSpaceLowValue=format.dwRBitMask | format.dwBBitMask;
         lpS1->SetColorKey(DDCKEY_SRCBLT,&ck);
         lpS2->SetColorKey(DDCKEY_SRCBLT,&ck);
         Pen1=CreatePen(PS_SOLID,2,RGB(30,193,208));
         Pen2=CreatePen(PS_SOLID,2,RGB(194,95,240));
         Pen3=CreatePen(PS_SOLID,2,RGB(203,255,92));
         Pen4=CreatePen(PS_SOLID,2,RGB(232,140,59));
```

```
        RECT rc  = {0,0,25,25};
        RECT rcd = {0,0,25,25};
        lpSurf->Blt(0,lpSBack,0,DDBLT_WAIT,0);
        while(rc.top<=240) {
                if(rc.bottom>240) rc.bottom=240;
                rcd.bottom =240-rc.top;
                if(rcd.bottom >25) rcd.bottom=25;
                while(rc.left<=Width) {
                        rcd.right=Width-rc.left;
                        if(rcd.right>25) rcd.right=25;
                        if(rc.right>Width) rc.right=Width;
                        lpS1->Blt(&rc,lpSPattern,&rcd,DDBLT_WAIT,0);
                        rc.left+=25;
                        rc.right+=25;
                }
                rc.top+=25;
                rc.bottom+=25;
                rc.left=0;
                rc.right=25;
        }
        lpS2->BltFast(0,0,lpS1,0,DDBLTFAST_WAIT);
        rc.left=Width-1;
}

CGraph::~CGraph() {
        RELEASE(lpSBack);
        RELEASE(lpS1);
        RELEASE(lpS1);
        RELEASE(lpSurf);
        RELEASE(lpSPattern);
        RELEASE(lpSWarning);
        DeleteObject(Pen1);
        DeleteObject(Pen2);
        DeleteObject(Pen3);
        DeleteObject(Pen4);
}

void CGraph::NextTick() {
        static RECT               rcd = {0,0,25,25};
        static unsigned char      cycle = 14;
        static unsigned char      cycleP = 0;
        static long               Temp = 0;
        static HDC                dc = 0;
        static int                PrevL[5];
        static int                PrevR[5];
        static RECT               rc  = {319,0,320,25};
        static RECT               rc1 = {1,0,320,240};
        static RECT               rc2 = {0,0,319,240};
        static POINT              prev1 = {318,120};
        static POINT              prev2 = {318,120};
        static POINT              prev3 = {318,120};
        static POINT              prev4 = {318,120};
        static bool               init = 0;
        if (!init) {
                rc.left=Width-1;
                rc.right=Width;
                rc1.right=Width;
                rc2.right=Width-1;
                prev1.x=Width-2;
                prev2.x=Width-2;
                prev3.x=Width-2;
                prev4.x=Width-2;
                init = 1;
        }
        lpS1->Blt(&rc2,lpS2,&rc1,DDBLT_WAIT,0);
        rc.top=0;
        rc.bottom=25;
        while(rc.top<=240) {
                if(rc.bottom>240) rc.bottom=240;
                rcd.bottom =240-rc.top;
                if(rcd.bottom >25) rcd.bottom=25;
                rcd.left=cycle;
                rcd.right=cycle+1;
                lpS1->Blt(&rc,lpSPattern,&rcd,DDBLT_WAIT,0);
                rc.top+=25;
                rc.bottom+=25;
        }
        //Get Device Context
        lpS1->GetDC(&dc);

        // Draw L Throttle
        if(((CEngine*)lpEngineL)->draw) {
                SelectObject(dc,Pen2);
                Temp=0;
                PrevL[cycleP]=((CEngine*)lpEngineL)->GetThrottle();
                for(unsigned int i=0;i<5;i++) Temp+=PrevL[i];
                Temp/=5L;
                Temp=120-(Temp*120)/256;
```

```
                if(Temp==240) Temp=239;
                else if(Temp==0) Temp=1;
                MoveToEx(dc,prev1.x,prev1.y,0);
                LineTo(dc,Width,Temp);
                prev1.y=Temp;
        }
        //Draw R Throttle
        if(((CEngine*)lpEngineR)->draw) {
                SelectObject(dc,Pen1);
                Temp=0;
                PrevR[cycleP]=((CEngine*)lpEngineR)->GetThrottle();
                for(int i=0;i<5;i++) Temp+=PrevR[i];
                Temp/=5L;
                Temp=120-(Temp*120)/256;
                if(Temp==240) Temp=239;
                else if(Temp==0) Temp=1;
                MoveToEx(dc,prev2.x,prev2.y,0);
                LineTo(dc,Width,Temp);
                prev2.y=Temp;
        }

        //Draw Speed;
        SelectObject(dc,Pen3);
        Temp=((CStats*)lpStats)->ISpeed;
        Temp=120-(Temp*120)/1590;
        if(Temp==240) Temp=239;
        else if(Temp==0) Temp=1;
        MoveToEx(dc,prev3.x,prev3.y,0);
        LineTo(dc,Width,Temp);
        prev3.y=Temp;

        //Draw Acceleration
        SelectObject(dc,Pen4);
        Temp=((CStats*)lpStats)->IAccel;

        Temp=120-(Temp*240)/120;
        if(Temp==240) Temp=239;
        else if(Temp==0) Temp=1;
        MoveToEx(dc,prev4.x,prev4.y,0);
        LineTo(dc,Width,Temp);
        prev4.y=Temp;

        //Desync ?
        if(!sync) SetPixel(dc,Width-2,0,RGB(255,0,0));

        //Done
        lpS1->ReleaseDC(dc);
        dc=0;

        lpSurf->BltFast(0,0,lpSBack,0,DDBLTFAST_WAIT);
        //if(!sync) lpSurf->BltFast(0,0,lpSWarning,0,DDBLTFAST_WAIT);
        lpSurf->BltFast(0,0,lpS1,0,DDBLTFAST_WAIT | DDBLTFAST_SRCCOLORKEY);
        cycle++;
        cycleP++;
        if(cycle==25) cycle=0;
        if(cycleP==5)cycleP=0;
        LPDIRECTDRAWSURFACE7 lpSTemp;
        lpSTemp=lpS1;
        lpS1=lpS2;
        lpS2=lpSTemp;
        bUpdate = 1;
        return;
}
```

Input.h

```
// Input.h: interface for the CInput class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_INPUT_H__33706F52_51AA_4F74_9AFF_C1BB50C53347__INCLUDED_)
#define AFX_INPUT_H__33706F52_51AA_4F74_9AFF_C1BB50C53347__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include "windows.h"
#include "ddraw.h"
#include "Engine.h"
#include "stats.h"
#include "module.h"


class CInput : public CModule
{
public:
        virtual void ProcessMouse(int x,int y,bool LButton,bool RButton,bool moving);
        void ProcessControl(int x,int y,bool FastTurn);
        CInput(RECT* rc);
        virtual ~CInput();

        RECT*    lprcWin;
};

#endif // !defined(AFX_INPUT_H__33706F52_51AA_4F74_9AFF_C1BB50C53347__INCLUDED_)
```

Input.cpp

```
// Input.cpp: implementation of the CInput class.
//
//////////////////////////////////////////////////////////////////////

#include "Input.h"
#include "engine.h"
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CInput::CInput(RECT* rc) {
        lprcWin=rc;
        lpInput=this;
        Init(this);
}

CInput::~CInput() {
}
long divround(long num1,float num2);
void CInput::ProcessMouse(int x,int y,bool LButton,bool RButton,bool moving) {
        RECT rc;
        HDC dc;
        if(LButton) {
                rc.left=lprcWin->left+divround(rcInput.left,*fWinRatio)+1;
                rc.top=lprcWin->top+divround(rcInput.top,*fWinRatio)+1;
                rc.right=lprcWin->left+divround(rcInput.right,*fWinRatio);
                rc.bottom=lprcWin->top+divround(rcInput.bottom,*fWinRatio);
                ClipCursor(&rc);

                lpSurf->BltFast(0,0,lpSBack,0,DDBLTFAST_WAIT);
                lpSurf->GetDC(&dc);
                SetPixel(dc,x-1,y-1,RGB(255,0,0));
                SetPixel(dc,x,y-1,RGB(255,0,0));
                SetPixel(dc,x+1,y-1,RGB(255,0,0));
                SetPixel(dc,x-1,y,RGB(255,0,0));
                SetPixel(dc,x,y,RGB(255,0,0));
                SetPixel(dc,x+1,y,RGB(255,0,0));
                SetPixel(dc,x-1,y+1,RGB(255,0,0));
                SetPixel(dc,x,y+1,RGB(255,0,0));
                SetPixel(dc,x+1,y+1,RGB(255,0,0));
                lpSurf->ReleaseDC(dc);
                bUpdate = 1;

                x=x - ( rcInput.right - rcInput.left ) /2;
                y=( rcInput.bottom - rcInput.top ) /2 - y;

                x=(x*254)/100;
                y=(y*270)/100;
                if(x>245) x=256;
                if(x<-245) x=-256;
                if(y>250) y=256;
                if(y<-250) y=-256;
                if(x>-10 && x<10) x=0;
```

```
                if(y>-10 && y<10) y=0;
                ProcessControl(x,y,RButton);
        }
        else ClipCursor(0);
}

void CInput::ProcessControl(int x,int y,bool FastTurn) {
        int L=y,R=y;
        if(x!=0) {
                if(x<0) {
                        if(FastTurn) L=L*(x)/256;
                        else L=(L*(256+x))/256;
                }
                else {
                        if(FastTurn) R=-R*(x)/256;
                        else R=(R*(256-x))/256;
                }
        }
        ((CEngine*)lpEngineL)->SetThrottle(L);
        ((CEngine*)lpEngineR)->SetThrottle(R);
}
```

Recorder.h

```
// Recorder.h: interface for the CRecorder class.
//
//////////////////////////////////////////////////////////////////////
#if !defined(AFX_RECORDER_H__C3DD5296_8277_4960_B93B_32B40A08CF97__INCLUDED_)
#define AFX_RECORDER_H__C3DD5296_8277_4960_B93B_32B40A08CF97__INCLUDED_

#include "windows.h"
#include "ddraw.h"
#include "commdlg.h"
#include "module.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CRecorder : public CModule
{
public:
        void DrawText(HDC dc,char* szText,RECT* rc);
        void DecRect(RECT* rc);
        void IncRect(RECT* rc);
        bool OpenFile(char* FileName);
        void RefreshButtons();
        void PushPause();
        void PushStop();
        void PushPlay();
        void PushRec();
        void Play(char* szFile);
        CRecorder(HWND hWnd);
        virtual ~CRecorder();
        OPENFILENAME                    open;
        LPDIRECTDRAWSURFACE7    lpSButtonDown;

        HFONT                                   hFont;
        HANDLE                                  hFile;
        char                                    szFileName[100];
        char                                    szTime[20];
        char                                    szSize[20];
        char                                    szStatus[20];
        bool                                    bFileName;
        bool                                    bTime;
        bool                                    bSize;
        bool                                    bStatus;


        HANDLE                                  hRefresh;
        HANDLE                                  hRecord;
        bool                                    bRun;
        unsigned char                   buttons;

};

#endif // !defined(AFX_RECORDER_H__C3DD5296_8277_4960_B93B_32B40A08CF97__INCLUDED_)
```

Recorder.cpp

```
// Recorder.cpp: implementation of the CRecorder class.
//
//////////////////////////////////////////////////////////////////////

#include "Recorder.h"
#include "resource.h"

#define RELEASE(a) if(a) {a->Release(),a=0;}
void Error(char* error);
LPDIRECTDRAWSURFACE7 LoadSurface(int rsBitmap);
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
unsigned long __stdcall RefreshRecorder(void* tac);
CRecorder::CRecorder(HWND hWnd) {
        lpRecorder=this;
        lpSButtonDown=0;
        buttons=0;
        hRecord=0;
        hRefresh=0;
        hFile=0;
        bTime=1;
        bSize=1;
        bFileName=1;
        bStatus=1;

        open.lStructSize=sizeof(open);
        open.hInstance=(HINSTANCE)GetWindowLong(hWnd,GWL_HINSTANCE);
        open.lpstrFilter="Car PlayBack Files (*.car)\0*.car\0";
        open.lpstrCustomFilter=0;
        open.nMaxCustFilter=0;
        open.nFilterIndex=1;
        open.nMaxFile=300;
        open.lpstrFileTitle=0;
        open.nMaxFileTitle=0;
        open.lpstrInitialDir=0;
        open.lpstrTitle=0;
        open.Flags=OFN_EXPLORER | OFN_OVERWRITEPROMPT;
        open.nFileOffset=0;
        open.nFileExtension=0;
        open.lpstrDefExt="car";
        open.lCustData=0;
        open.lpfnHook=0;
        open.lpTemplateName=0;
        open.hwndOwner=hWnd;

        Init(this);

        //lpSButtonDown=LoadSurface("recorder_buttons.bmp");
        lpSButtonDown=LoadSurface(IDB_RECORDER_BUTTONS);
        RefreshButtons();
        hFont = CreateFont(32,0,0,0,0,0,0,0,0,
                                                OUT_TT_PRECIS,
                                                CLIP_TT_ALWAYS,
                                                ANTIALIASED_QUALITY,
                                                FF_DONTCARE,
                                                "Palatino Linotype");
        bRun=1;
        strcpy(szTime,"00:00");
        strcpy(szStatus,"Waiting...");
        strcpy(szSize,"0 Ko");
        strcpy(szFileName,"----");
        unsigned long Id=0;
        hRefresh=CreateThread(0,0,&RefreshRecorder,this,0,&Id);
}

CRecorder::~CRecorder() {
        bRun=0;
        unsigned long ex=STILL_ACTIVE;
        while(ex==STILL_ACTIVE) {
                GetExitCodeThread(hRefresh,&ex);
                Sleep(1);
        }
        if(hRecord) CloseHandle(hRecord);
        CloseHandle(hRefresh);
        RELEASE(lpSButtonDown);
        RELEASE(lpSBack);
        RELEASE(lpSurf);
        if(hFile) CloseHandle(hFile);
        DeleteObject(hFont);

}

unsigned long __stdcall RefreshRecorder(void* tac) {
        CRecorder*      rec=0;
```

```
	HDC		dc=0;
	RECT		rcSurf = {0,0,200,147};
	RECT		rcStatus = {2,5,198,40};
	RECT		rcFileName = {2,40,198,75};
	RECT		rcSize = {2,75,198,110};
	RECT		rcTime = {2,110,198,145};
	rec=(CRecorder*)tac;
	rec->lpSurf->BltFast(0,0,rec->lpSBack,&rcSurf,DDBLTFAST_WAIT);
	while(rec->bRun==1) {
		rec->lpSurf->GetDC(&dc);

		SelectObject(dc,rec->hFont);
		SetBkMode(dc,TRANSPARENT);
			if(rec->bStatus) {
				rec->DrawText(dc,rec->szStatus,&rcStatus);
				rec->bStatus=0;
			}
			if(rec->bFileName) {
				rec->DrawText(dc,rec->szFileName,&rcFileName);
				rec->bFileName=0;
			}
			if(rec->bSize) {
				rec->DrawText(dc,rec->szSize,&rcSize);
				rec->bSize=0;
			}
			if(rec->bTime) {
				rec->DrawText(dc,rec->szTime,&rcTime);
				rec->bTime=0;
			}
		rec->lpSurf->ReleaseDC(dc);
		dc=0;
		Sleep(100);
	}
	return 0;
}
BOOL CALLBACK DialogProca(HWND hwndDlg,UINT uMsg,WPARAM wParam,LPARAM lParam) {
	switch (uMsg) {
	case WM_INITDIALOG:
	//	return 1;
		break;
	case WM_LBUTTONDOWN:

		EndDialog(hwndDlg,0);
		break;
	}
	return 0;
}
unsigned long __stdcall Record(void* tac) {
	//return 0;
		CRecorder*	rec=0;
		rec=(CRecorder*)tac;
		rec->open.lpstrTitle="Choose the file to record to";
		rec->open.lpstrFile=0;
		if(!GetSaveFileName(&rec->open)) {
			return 0;
		}
		//int hop=DialogBox(rec->open.hInstance,MAKEINTRESOURCE(101),rec->open.hwndOwner,DialogProca);
		HWND hop=CreateDialog(rec->open.hInstance,MAKEINTRESOURCE(101),rec->open.hwndOwner,DialogProca);
		ShowWindow(hop,SW_SHOWNORMAL);
		MSG msg;
		while(GetMessage(&msg,hop,0,0)) {
			if(msg.message==WM_LBUTTONDOWN) break;
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
		return 0;
}
void CRecorder::Play(char* szFile) {
	if(OpenFile(szFile)) {

	}
}
void CRecorder::PushRec() {
	char vlan[256];
	OpenFile(vlan);
	if(buttons==1) {
		if(hRecord) TerminateThread(hRecord,0);
		if(hRecord) CloseHandle(hRecord);
		hRecord=0;
		unsigned long Id=0;
		hRefresh=CreateThread(0,0,&Record,this,0,&Id);
	}
	buttons=0;
	RefreshButtons();
}

void CRecorder::PushPlay() {
	if(buttons==2) {
```

```
        }
        buttons=0;
}

void CRecorder::PushStop() {
        if(buttons==3) {
        }
        buttons=0;
}

void CRecorder::PushPause() {
        if(buttons==4) {

        }
        buttons=0;
}

void CRecorder::RefreshButtons() {
        RECT            rc;
        RECT            rcSurf={0,147,200,200};
        lpSurf->BltFast(0,147,lpSBack,&rcSurf,DDBLTFAST_WAIT);
        rc.top=0;
        rc.bottom=39;
        switch(buttons) {
        case 1:
                rc.left=0;
                rc.right=39;
                lpSurf->BltFast(10,148,lpSButtonDown,&rc,DDBLTFAST_WAIT);
                break;
        case 2:
                rc.left=47;
                rc.right=86;
                lpSurf->BltFast(57,148,lpSButtonDown,&rc,DDBLTFAST_WAIT);
                break;
        case 3:
                rc.left=94;
                rc.right=133;
                lpSurf->BltFast(104,148,lpSButtonDown,&rc,DDBLTFAST_WAIT);
                break;
        case 4:
                rc.left=141;
                rc.right=180;
                lpSurf->BltFast(151,148,lpSButtonDown,&rc,DDBLTFAST_WAIT);
                break;
        }
}
bool CRecorder::OpenFile(char* FileName) {
        if(hFile) CloseHandle(hFile);
        hFile=0;
        strcpy(szFileName,FileName);
        hFile=CreateFile(szFileName,
                                        GENERIC_READ | GENERIC_WRITE,
                                        FILE_SHARE_READ,
                                        0,
                                        OPEN_ALWAYS,
                                        FILE_ATTRIBUTE_NORMAL | FILE_FLAG_SEQUENTIAL_SCAN,
                                        0);
        if(hFile==INVALID_HANDLE_VALUE) {
                strcpy(szFileName,"Cant't open file");
                return 0;
        }
        return 1;
}
void CRecorder::IncRect(RECT *rc) {
        rc->left+=2;
        rc->top+=2;
        rc->right+=2;
        rc->bottom+=2;
}
void CRecorder::DecRect(RECT *rc) {
        rc->left-=2;
        rc->top-=2;
        rc->right-=2;
        rc->bottom-=2;
}

void CRecorder::DrawText(HDC dc,char *szText, RECT *rc) {
        SetTextColor(dc,RGB(0,0,0));
        IncRect(rc);
        lpSurf->BltFast(rc->left,rc->top,lpSBack,rc,DDBLTFAST_WAIT);
        ::DrawText(dc,szText,strlen(szText),rc,DT_CENTER);
        DecRect(rc);
        SetTextColor(dc,RGB(255,255,255));
        ::DrawText(dc,szText,strlen(szText),rc,DT_CENTER);
}
```

Speed.h

```
// Speed.h: interface for the CSpeed class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_SPEED_H__36079A01_3ED1_4DDE_92D5_21FE9B929198__INCLUDED_)
#define AFX_SPEED_H__36079A01_3ED1_4DDE_92D5_21FE9B929198__INCLUDED_
#include "module.h"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CSpeed : public CModule
{
public:
        CSpeed();
        virtual ~CSpeed();
        SetSpeed(int speed);
        void ResetMaxSpeed()            { MaxSpeed=Speed; }
        int GetSpeed()                          { return Speed; }
        int GetMaxSpeed()                       { return MaxSpeed; }

private:
        int                     Speed;
        int                     MaxSpeed;

};

#endif // !defined(AFX_SPEED_H__36079A01_3ED1_4DDE_92D5_21FE9B929198__INCLUDED_)
```

Speed.cpp

```
// Speed.cpp: implementation of the CSpeed class.
//
//////////////////////////////////////////////////////////////////////

#include "Speed.h"

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CSpeed::CSpeed() {
        lpSpeed=this;
        Init(this);
        Speed=0;
        MaxSpeed=0;
}

CSpeed::~CSpeed() {
}

CSpeed::SetSpeed(int speed) {
        Speed=speed;
        if(Speed>MaxSpeed) MaxSpeed=Speed;
}
```

Stats.h

```
// Stats.h: interface for the CStats class.
//
//////////////////////////////////////////////////////////////////////
#if !defined(AFX_STATS_H__BE575F69_04F0_4F00_8FE5_D1ABABA5021D__INCLUDED_)
#define AFX_STATS_H__BE575F69_04F0_4F00_8FE5_D1ABABA5021D__INCLUDED_

#include "windows.h"
#include "ddraw.h"
#include "Engine.h"
#include "Speed.h"
#include "module.h"


#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CStats : public CModule
{
public:
        CStats(unsigned int len);
        virtual void NextTick();
        virtual ~CStats();

        HFONT                                   hFont;
        int*                                    Speed;
        int*                                    Accel;
        int*                                    Power;
        unsigned int                    Lenght;
        long                                    ISpeed;
        long                                    IAccel;
};

#endif // !defined(AFX_STATS_H__BE575F69_04F0_4F00_8FE5_D1ABABA5021D__INCLUDED_)
```

Stats.cpp

```
// Stats.cpp: implementation of the CStats class.
//
//////////////////////////////////////////////////////////////////////
#define BENCH

#include "Stats.h"
#include "Engine.h"
#include "Speed.h"
#include "perf.h"
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
void Error(char* error);
unsigned long __stdcall Stating(void* tac);
#define RELEASE(a) if(a) {a->Release(),a=0;}

#define UPDATE 0
//#define LENGHT 320

CStats::CStats(unsigned int len) {
        lpStats=this;
        Lenght=len;

        Speed=new int[Lenght];
        Accel=new int[Lenght];
        Power=new int[Lenght];
        ISpeed=0;
        IAccel=0;

        ZeroMemory(Speed,Lenght*sizeof(int));
        ZeroMemory(Accel,Lenght*sizeof(int));
        ZeroMemory(Power,Lenght*sizeof(int));
        hFont=0;

        Init(this);
        hFont = CreateFont(24,0,0,0,500,0,0,0,0,
                                        OUT_TT_PRECIS,
                                        CLIP_TT_ALWAYS,
                                        ANTIALIASED_QUALITY,
                                        FF_DONTCARE,
                                        "Palatino Linotype");
}

CStats::~CStats() {
        RELEASE(lpSBack);
        RELEASE(lpSurf);
        DeleteObject(hFont);
        delete Speed;
```

```
        delete Accel;
        delete Power;
}
#define AccelSmooth 10
void CStats::NextTick() {
        static bool                     sync=1;
        static unsigned long            Time = 0;
        static long                     Temp = 0;
        static unsigned int             cycle = 0;
        static unsigned int             cyclea = Lenght-2;
        static RECT                     rc1 = {2,4,58,196};
        static RECT                     rc2 = {0,2,60,198};
        static HDC                      dc = 0;
        static char                     szTemp[200];
        static DDBLTFX                  fx;
        static long                     MSpeed=0;
        static long                     ASpeed=0;
        static long                     AAccel=0;
        static long                     IPower=0;
        static long                     APower=0;
        static long                     AccelS[AccelSmooth];
        static char                     szLights[4];
        int                             L=0;
        int                             R=0;

        strcpy(szLights,"Off");

    ZeroMemory(&fx,sizeof(fx));
    fx.dwSize=sizeof(fx);
    fx.dwFillColor=RGB(255,0,255);

        {
                Time=GetTickCount();

                L=((CEngine*)lpEngineL)->GetThrottle();
                R=((CEngine*)lpEngineR)->GetThrottle();

                //IPower=abs(L)+abs(R);
                IPower=L+R; //Used for fake speed
                IPower=IPower*100/512;
                *(Power+cycle)=IPower;

                Temp=0;
                for(unsigned int i=50,j=cycle;i>0;i--) {
                        Temp+=*(Power+j);
                        if(j==0) j=Lenght;
                        j--;
                }
                Temp=Temp*13/50;
                ((CSpeed*)lpSpeed)->SetSpeed(Temp);
                ISpeed=((CSpeed*)lpSpeed)->GetSpeed();
                *(Speed+cycle)=ISpeed;
                MSpeed=((CSpeed*)lpSpeed)->GetMaxSpeed();

                APower=0;
                for(i=0;i<Lenght;i++) APower+=*(Power+i);
                APower/=(long)Lenght;


                ASpeed=0;
                for(i=0;i<Lenght;i++) ASpeed+=*(Speed+i);
                ASpeed/=(long)Lenght;

                *(Accel+cycle)=(ISpeed-*(Speed+cyclea));


                long accel=0;
                for(i=5,j=cycle;i>0;i--) {
                        accel+=*(Accel+j);
                        if(j==0) j=Lenght;
                        j--;
                }
                AccelS[cycle%AccelSmooth]=accel/5L;
                accel=0;

                for(i=0;i<AccelSmooth;i++) {
                        accel+=AccelS[i];
                }
                accel/=(long)AccelSmooth+5;
                IAccel=accel;



                AAccel=0;
                for(i=0;i<Lenght;i++) AAccel+= *(Accel+i);
                AAccel/=(long)Lenght;

                wsprintf(szTemp,
```

```
                                "%li\n%li\n%li\n%li\n%li\n%li%%\n%li%%\n%s",
                                MSpeed,
                                ISpeed,
                                ASpeed,
                                IAccel,
                                AAccel,
                                IPower,
                                APower,
                                szLights);

                lpSurf->BltFast(0,0,lpSBack,0,0);

                lpSurf->GetDC(&dc);
                SelectObject(dc,hFont);
                SetBkMode(dc,TRANSPARENT);
                SetBkColor(dc,RGB(255,0,255));
                SetTextColor(dc,RGB(0,0,0));
                DrawText(dc,szTemp,strlen(szTemp),&rc1,DT_LEFT);
                SetTextColor(dc,RGB(255,255,255));
                DrawText(dc,szTemp,strlen(szTemp),&rc2,DT_LEFT);
                lpSurf->ReleaseDC(dc);

                Temp=UPDATE-(GetTickCount()-Time);
                if(Temp<=0) { //Handle Desync
                        Temp=0;
                        sync=0;
                }
                else sync=1;
                if(Temp>=UPDATE) Temp=UPDATE; //Protection
                cycle++;
                cyclea++;
                if(cycle==Lenght) cycle=0;
                if(cyclea==Lenght) cyclea=0;
                if(Temp!=0) Sleep(Temp); //Sync
        }
        bUpdate = 1;
        return;
}
```

...:: CONCLUSION ::...

Le Bilan

Grâce aux TPE, nous avons élargi notre connaissance en mécanique et en électronique numérique. A present, nous nous ouvrons vers des projets de domotique, robotiques, etc... En résumé, ce fut une éxperience en temps limitée qui nous a fait apprendre énormement de chose.

Remerciements : Fabrice Delauré - Mr Grialou et sa secrétaire - Mr Duviau - Mr Lechevrel - et tout ceux qui nous ont aidé.