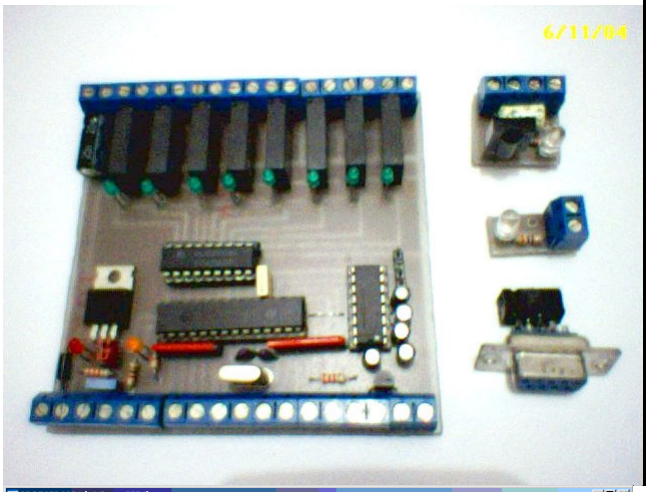
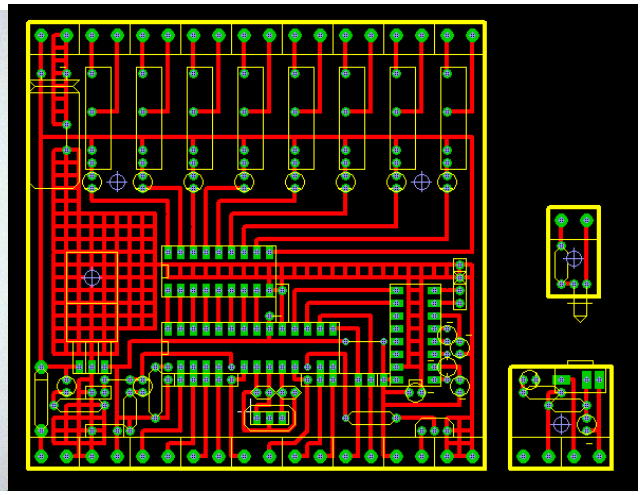
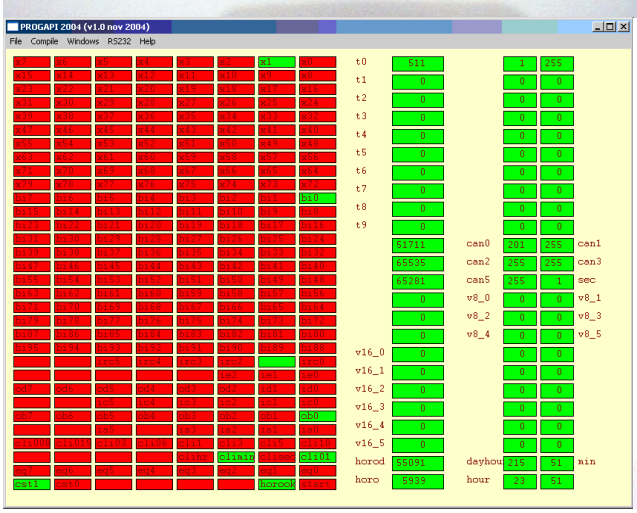
Les automates programmables industriels 2004 d'Ulysse

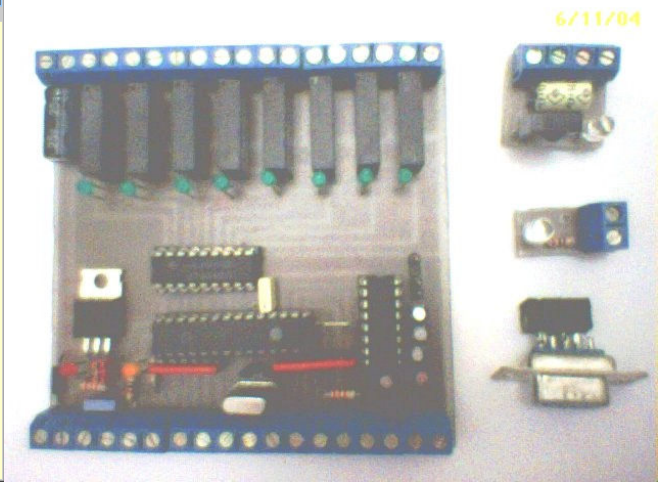
(à base de PIC16F873, 874, 876, 877)

Cette documentation présente tout ce qu'il faut pour comprendre, réaliser soi même et utiliser mes automates programmables industriels.









(Les photos représentent un automate 10 entrées/8sorties, un émetteur et un récepteur pour réaliser une barrière infrarouge et un adaptateur RS232 pour connecter l'automate au PC avec un câble série. L'automate est de la taille d'une disquette (9cm x 9cm))

Ulysse Delmas-Begue novembre 2004
Email : udelmas@chez.com / ulysse.delmas@laposte.net
Site web: http://www.chez.com/udelmas

Table des matières

A. Présentation

Les automates programmables industriels ou API, sont des systèmes électroniques de commande très facile à programmer grâce a l'utilisation du graphcet qui est une représentation graphique du comportement du système à contrôler.

Les API sont très utilisés dans l'industrie dans de nombreux automatismes, mais malheureusement, ils restent très peu connu du grand public. Bien que le prix des API commerciaux baisse régulièrement, ils restent cependant assez coûteux, surtout en ce qui concerne le logiciel de programmation souvent hors de prix, ce qui rend difficile un amortissement pour quelques automates. Néanmoins, les solutions commerciales sont extrêmement robustes.

Lorsqu'en 1998, je mis sur Internet, la description de mes automates à base de PIC16f84, l'engouement fut immédiat aussi bien pour diverses réalisations personnelles que dans l'enseignement. Grâce a l'évolution des microcontrôleurs (UC) et a de nombreuses suggestions, je remets ça, avec cette fois des UC 16F873, 874, 876 et 877 plus adaptés, et en conservant toujours le même esprit d'accessibilité, de partage et de gratuité.

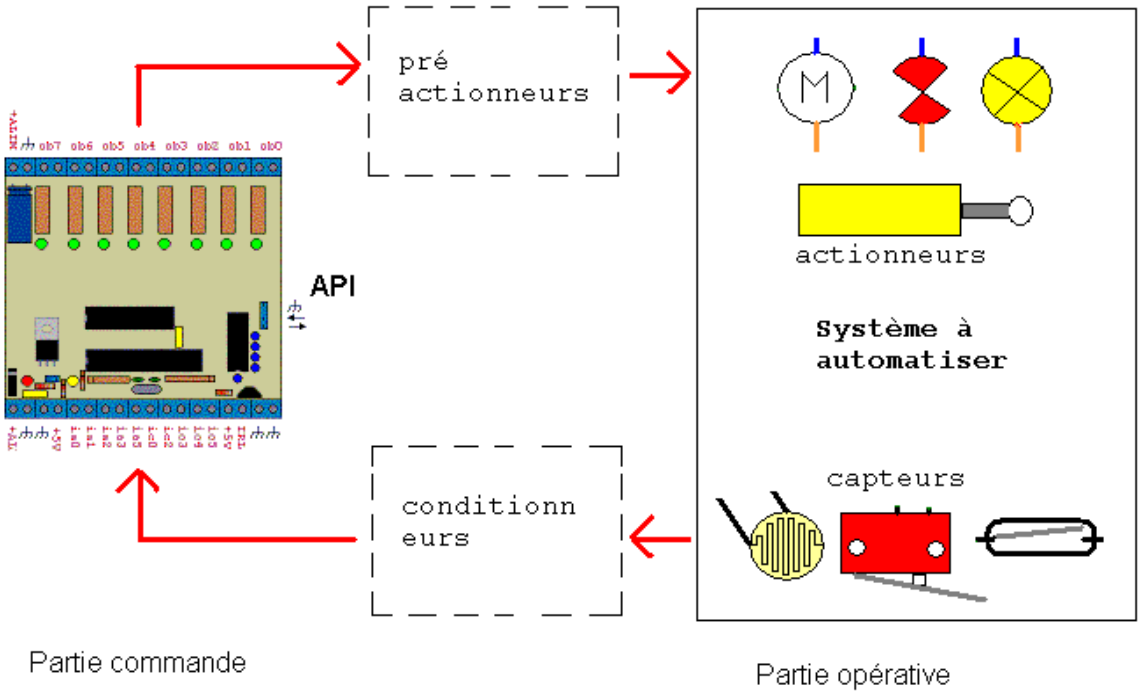
Les chapitres suivants décrivent mes automates 2004. Ils devraient être compréhensibles par tout le monde. Sur ce, bonne lecture et pourquoi pas, si cela vous plait, bonne réalisation.

B. Synoptique

Un système automatique est constitué d'une partie commande (un automate dans notre cas) et d'une partie opérative (le système à automatiser). Des actionneurs (moteurs, lampes, vérins …), permettent à la partie commande de faire évoluer l'état de la partie opérative. En retour des capteurs l'informe de l'état du système.

Dans notre cas, l'API pilote le système avec ses sorties qui commandent les actionneurs en fonction de l'état des capteurs et du graphcet. (Si les sorties, ne sont pas adaptées aux actionneurs alors, il faut utiliser des pré actionneurs. Idem pour les entrées avec des conditionneurs)

La figure suivante montre les différentes entités :



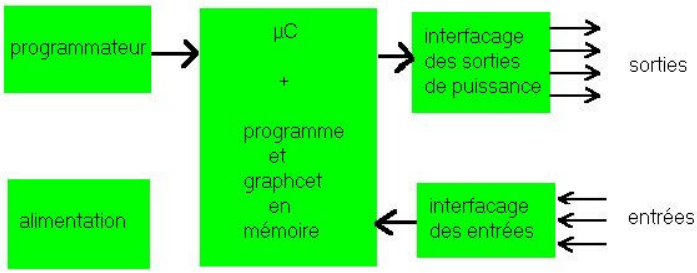
Mes automates sont également capables de gérer des capteurs analogiques, d'utiliser l'horodateur intégré et des compteurs.

Le principe d'utilisation est le suivant : on construit le graphcet de commande sur le PC, puis on le transfère dans l'automate qui l'exécute.

Cette réalisation peut se diviser en 3 grandes parties qui sont :

- la partie électronique
- La partie informatique au niveau du UC de l'automate encore connu sous le nom de firmware ou d'interpréteur.
- La partie informatique au niveau du PC, c'est-à-dire le logiciel utilisateur.

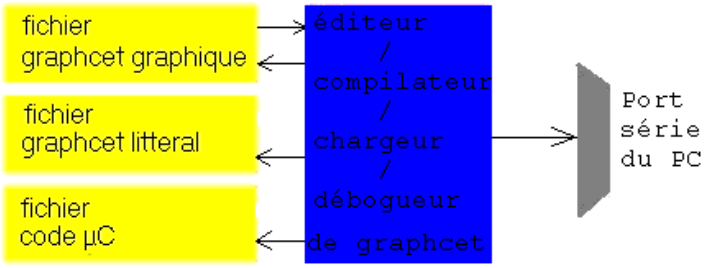
B.1 La partie électronique



Du point de vue électronique l'automate se décompose en 5 sous-ensembles :

- Le microcontrôleur coeur du montage qui grâce à un programme exécute le graphcet qui est dans sa mémoire.
- Le programmateur ou adaptateur RS232, permet de dialoguer avec la PC pour transférer le graphcet et observer l'API.
- L'alimentation
- Le système d'interfaçage d'entrée (ici 10 ou 15 entrées)
- Le système d'interfaçage de sortie (ici 8 ou 14 sorties)

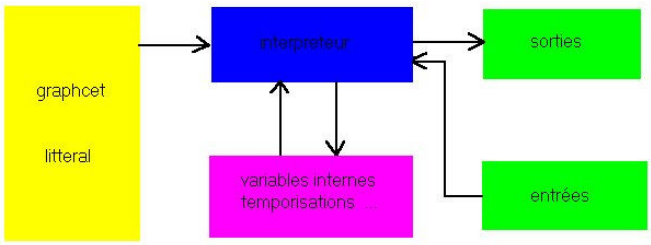
B.2 La partie informatique au niveau du PC



Coté PC, un programme se charge de tout **: « L'éditeur, compilateur chargeur, débogueur de graphcet »**.
Ce programme du nom de **progapi_2004** permet de :

- Créer un graphcet graphique et de le transformer en graphcet littéral qui est une représentation textuelle du graphcet.
- Transférer le nouveau graphcet littéral dans la mémoire FLASH du UC.
- Observer le comportement de l'API et l'évolution du graphcet.

B.3 La partie informatique au niveau du UC



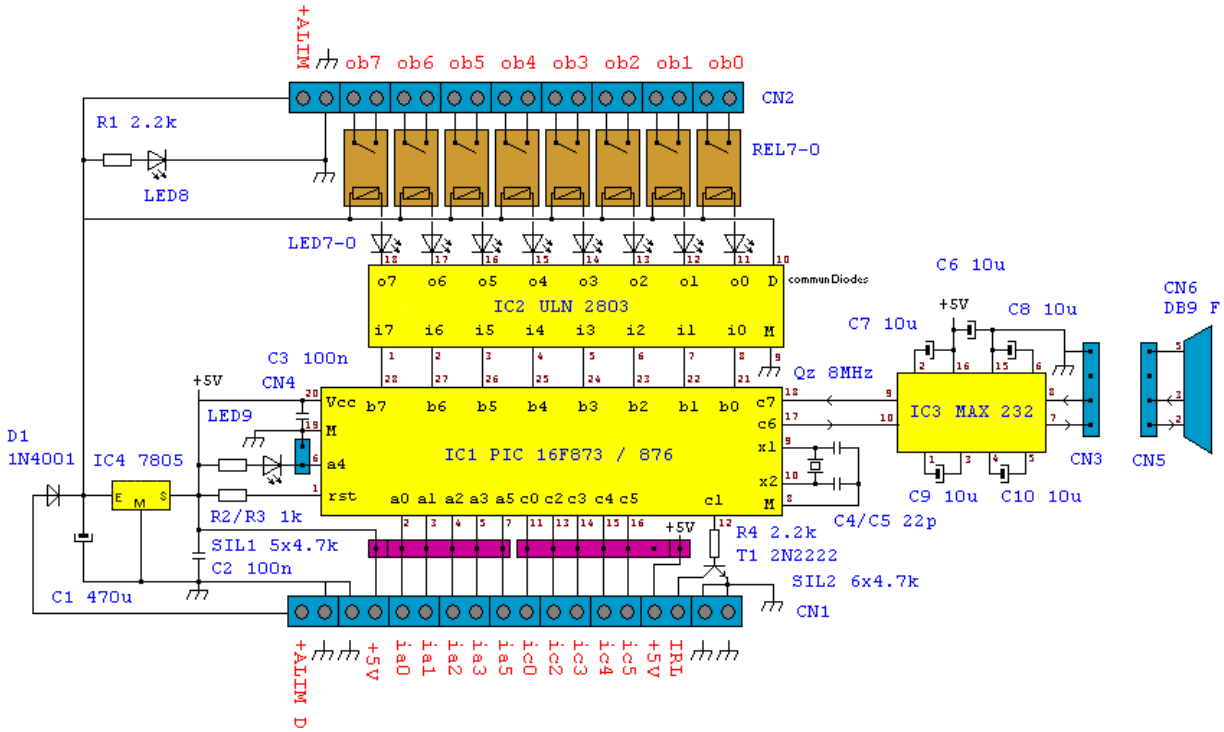
Coté UC, c'est un programme appelé « **interpréteur de graphcet » ou « firmware »** qui gère l'automate en exécutant le graphcet littéral et en mettant à jour les entrées, sorties, variables internes (temporisations...). Il est également capable de dialoguer avec le PC grâce au bootloader/moniteur intégré.

C. Etude de la partie électronique

Mon but était de rendre la partie électronique simple à réaliser, elle n'en est pas moins intéressante pour autant.
Je détaille la réalisation d'un automate avec 10 entrées et 8 sorties (API_2004_18).
Il vous est également possible de réaliser une version avec 15 entrées et 14 sorties (API_2004_29) à l'aide des informations présentes mais je ne la détaille pas.

C.1. Schéma électronique

Version 10 entrées et 8 sorties



Nomenclature

```
IC1       : PIC16F873 ou 876    (microcontrôleur)
IC2       : ULN 2803            (amplification de puissance darlington))
IC3       : MAX232              (adaptateur RS232)
IC4       : 7805                (régulateur 5 volts)
D1        : 1N4001              (Diode de redressement)
R1,R4     : 2200 ohms
R2,R3     : 1000 ohms
SIL1      : 5x 4.7k
SIL2      : 6x 4.7k
C1        : 470 uF
C2,C3     : 100 nF
C4,C5     : 22 pF
C6-10     : 10 uF
Qz        : Quartz 8 MHz
Led0-7    : diodes électroluminescentes vertes 3mm (ou ce que vous voulez)
Led8      : diode  électroluminescente  rouge  3mm (ou ce que vous voulez)
Led9      : diode  électroluminescente  jaune  3mm (ou ce que vous voulez)
T1        : transistor 2N2222
REL0-7    : relais de puissance SIL 12V, 1200Ohms, 30V 5A continue, 250V 5A alternatif)
CN1,CN2   : borniers a vis (chacun constitué de 9 bornier à vis de 2 éléments)
CN3       : 2 éléments (d'une barrette auto sécable mâle simple rangée) + 1cavalier
CN4       : 4 éléments (d'une barrette auto sécable mâle simple rangée)
CN5       : 4 éléments (d'une barrette auto sécable femelle simple rangée)
CN6       : connecteur DB9 femelle
SUP1      : support pour IC1 (support tulipe 28 broches, 2x14 étroit)
SUP2      : support pour IC2 (support tulipe 18 broches, 2x9        )
SUP3      : support pour IC3 (support tulipe 16 broches, 2x8        )
CV1       : cavalier pour CN4 (facultatif)
```

C.1 Le UC

Le microcontrôleur IC1 est le cœur du montage. Il est chargé d'interpréter le graphcet littéral chargé dans sa mémoire.

Il est possible d'utiliser indifféremment les PIC16f873 ou 876 pour l'automate 10E/8S. Les différences entre ces UC, concernent uniquement la taille des différentes mémoires. L'automate 15E/14S demande quand à lui un UC avec des E/S supplémentaires. Les PIC 16f874 et 877 sont les équivalents des UC précédents mais avec 11 E/S supplémentaires.

J'ai choisit ces UC pour les raisons suivantes :

- Ils disposent d'une mémoire flash pour stocker le programme (donc il est possible de les reprogrammer en cas de bug ou d'évolution). La mémoire flash peut être programmé par le programme lui-même, ce qui est très pratique pour transférer le graphcet.
- Il est facile de fabriquer un programmateur et de nombreux outils sont disponibles sur Internet. (Je conseille icprog pour le programme et JDM pour la carte de programmation)
- La mémoire RAM est de taille suffisante (tout juste)
- Ils disposent d'un port série, d'un convertisseur analogique numérique et de timers.
- Ces UC se trouvent partout et sont peu coûteux (Microchip envoie même des échantillons gratuitement)
- Enfin ces UC sont un vrai plaisir à programmer en assembleur et l'environnement de développement MPLAB est gratuit et puissant.

Caractéristiques de ces UC :

```
mémoire programme : 4 kmots (16f873/874) / 8 kmots (16f876/877)
RAM               : 192 octets           / 368 octets
EEPROM données    : 128 octets           / 256 octets

entrées/sorties   : 22      (16f873/876) / 33      (16f874/877)
boîtier           : DIL 28               / DIL40
temps de cycle    : 0.5uS à 8MHz
architecture      : RISC

instructions très puissantes donc programme réduit
programmation simple grâce au mode série ISP
flash programmable au minimum 1000 fois et eeprom 1000000 fois
```

Pour plus d'information, télécharger les manuels chez Microchip (le fabriquant des PICS) http://www.microchip.com (en Anglais) et lisez les excellents cours de « BigOnOff » en Français http://abcelectronique.com .

Pour le néophyte:

UC : Un microcontrôleur est un composant qui regroupe au sein d'une puce, un véritable petit ordinateur, il comprend:
- un microprocesseur
- différentes mémoires pour les données et les instructions
- des périphériques (dont des ports d'entrée/sortie E/S, des convertisseurs analogique/numérique pour mesurer des tensions, des compteurs/temporisateurs, un ports série …)

FLASH/EEPROM : mémoires qui conservent les données même lorsque l'alimentation est coupée. Elles sont donc bien adaptées pour stocker un programme ou des constantes. Ces mémoires ont des caractéristiques différentes telles la densité, et l'endurance. Mes API utilisent uniquement la FLASH pour stocker le programme et le graphcet car les PIC utilisent la FLASH pour stocker les instructions du programme et l'EEPROM habituellement réservée au stockage des données est trop petite pour les grahcets. Ces mémoires sont très lentes en écriture

RAM : mémoire volatile qui perd son contenu lorsque l'alimentation est coupée mais très rapide aussi bien en lecture qu'en écriture. Cette mémoire est tout indiquée pour stocker les variables.

RISC : Architecture de UC avec un jeu réduit d'instruction par opposition aux architectures CISC.

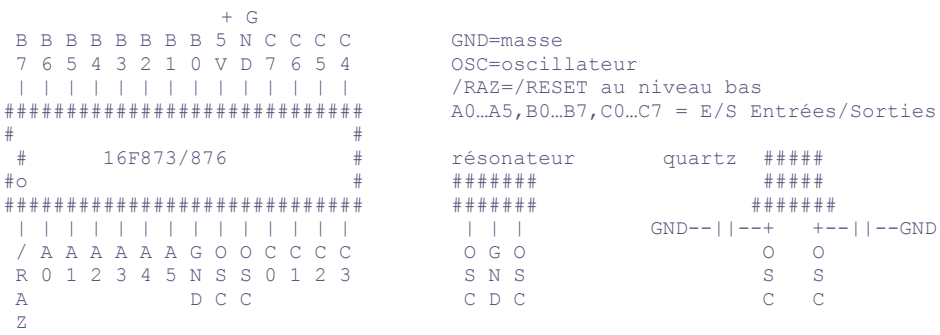
Bit : Unité élémentaire d'information (le fameux : 0 ou 1)

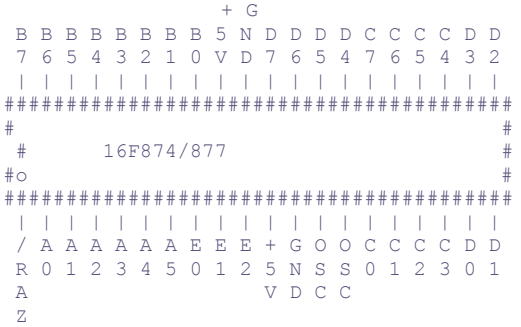
Octet : ensemble de 8 bits (Il est possible de coder 256 états en général 0-255)

Mot : ensemble de bits. Ici 14 bits, ce qui correspond à un emplacement en FLASH et qui correspond à la taille d'une instruction pour les PIC 16xxx.

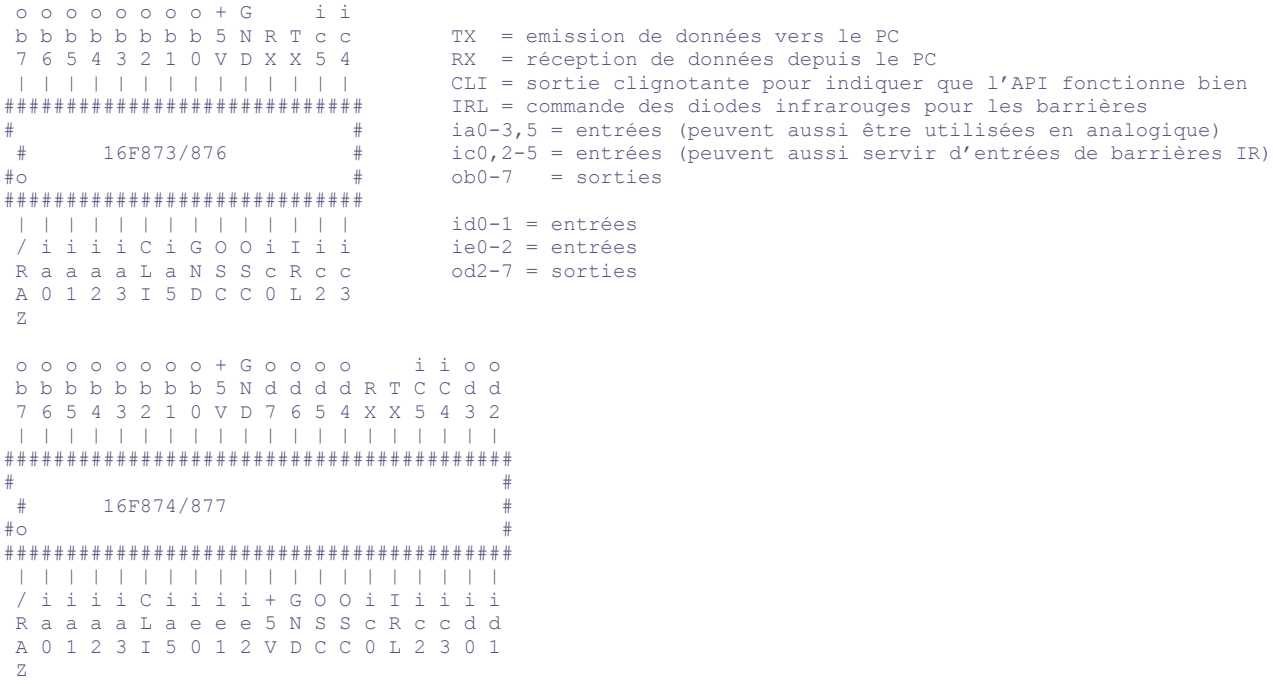
Brochages des UC :

Brochages





Brochages avec l'interpréteur



Composants annexes au UC

Pour fonctionner, un UC à besoin d'un oscillateur pour faire battre son cœur. Ici le quartz de 8 MHz fournit 8 millions d'impulsions par seconde. Le UC divise cette fréquence par 4 et peut donc exécuter 2 millions d'instructions par seconde soit une instruction toutes les 0.5 microsecondes. Les 2 condensateurs céramiques C4 et C5 sont nécessaires au bon fonctionnement du quartz. Il est possible d'utiliser un résonateur céramique à la place du quartz et de ces condensateurs, mais ce dernier est moins précis qu'un quartz (0.5% au lieu de 0.02%). L'utilisation du quartz est impératif si l'horodateur est utilisé (dérive de 17s max par jour au lieu de 7 minutes).

Le RESET du UC est mis au 5V à travers la résistance R3. Dans cette configuration, le UC est uniquement initialisé à la mise sous tension.

Pour fonctionner, le UC à besoin d'une tension stable. L'alimentation fournit du 5V bien propre. Le condensateur C3 filtre les hautes fréquences qui pourraient perturber le PIC ainsi que celles qu'il génère.

La patte CLI permet de faire clignoter la Led 9 dont le courant est limité par R2. Elle indique le bon fonctionnement de l'automate. Le connecteur CN3 permet de mettre cette patte à la masse, ce qui permet à la mise sous tension de lancer le bootloader, mais vous ne devriez pas en avoir besoin pour le moment. Il est a noté que la mise à la masse de cette patte ne peut pas détruire la sortie car c'est un collecteur ouvert.

C.2 étude des entrées

Les entrées sont accessibles via le bornier à vis CN1 afin d'y connecter facilement les capteurs.

Ces API peuvent gérer différents types de capteurs :

- Capteurs tout ou rien TOR
- Capteurs analogiques
- Capteurs analogiques utilisés en TOR
- Capteurs pour faire des barrières IR

Les capteurs TOR

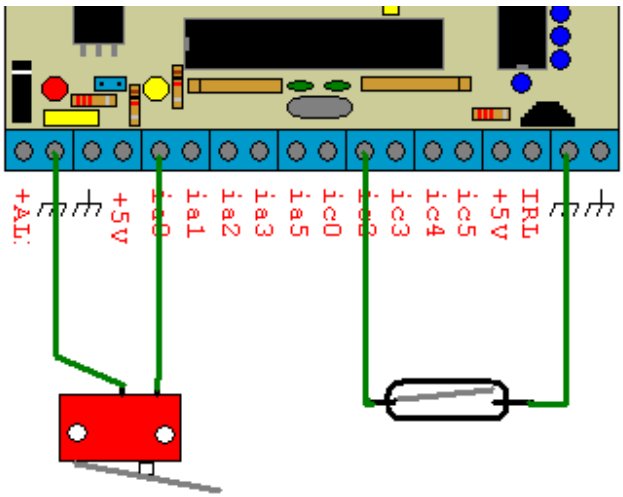
Les capteurs TOR, sont des capteurs à 2 états. Ils sont aussi connus sous le nom de « capteurs à lame ». En effet ils utilisent le plus souvent une lame métallique pour faire passer ou non un courant. Les interrupteurs en sont le plus simple exemple.

Voici quelques exemples de ces capteurs :

- Divers interrupteurs manipulés par l'utilisateur: Interrupteurs à bascule, à glissière, boutons poussoirs, touches …
- Divers interrupteurs actionnés par le système: Capteur à levier, à tige, à galet
- ILS : Interrupteur à lame souple : Ils sont constitués d'une lame dans une ampoule de verre qui fait contact et donc ferme l'interrupteur (le courant peut passer) en présence d'un aimant.
- Contacts de relais : Des montages électronique peuvent délivrer leurs informations par le biais d'un relais. Ils contrôlent la bobine du relais et le commutateur est utilisé comme entrée. C'est par exemple le cas de certains capteurs infrarouges passif utilisés dans les alarmes.

Les entrées du PIC sont reliées au +5V par l'intermédiaire des résistance de « pull-up » des réseaux SIL. Un réseau SIL est simplement un ensemble de résistance dans un paquetage en ligne plus facile à utiliser et qui prend moins de place que les résistances traditionnelles. Dans les SIL utilisés, il y a une résistance entre le commun et chacune des autres pattes. En l'absence de capteur ou en présence d'un capteur ouvert (inactif), le UC voit 5V. C'est-à-dire un niveau haut ou « 1 ». Le firmware (programme du PIC) inverse cet état, si bien qu'il voit « 0 ». Pour voir un état « 1 », il suffit de mettre l'entrée correspondante à la masse. Il suffit donc de connecter le capteur entre la masse et l'entrée désirée. Ainsi lorsque le capteur est « actif » ou fermé, les 2 lames se touchent et l'entrée voit la masse et l'API en déduit que le capteur est actif.

La figure suivante montre un capteur à levier connecté sur l'entrée ia0 et un ILS sur l'entrée ic2.



Le nom des entrées est du type i<lettre><chiffre> avec i pour input (entrée en Anglais) , lettre pour le nom du port du PIC (ici port A ou C) et chiffre pour numéro de l'entrée de ce port. Vous remarquerez certainement que les entrées ia4 et ic1 n'existent pas. Ceci est normal, car ces pattes du PIC sont utilisées à d'autres fins.

Pour visualiser l'état des entrées, il est possible d'utiliser le programme sur PC « progapi_2004 » ou de rajouter une led et sa résistance entre le +5V et l'entrée correspondante. Je n'ai pas mis ces leds sur la carte car elles perturbent les capteurs analogiques et augmentent la consommation. Vous pouvez toute fois les rajouter si cette fonctionnalité vous semble indispensable.

Afin de simplifier la réalisation, les entrées, sont directement reliées aux pattes du UC. Pour ceux souhaitant une isolation, il est possible d'ajouter des optocoupleurs comme cela se fait dans le monde industriel.

Je conseille de monter les SIL sur un support tulipe afin de pouvoir les changer si besoin. En effet, la valeur de 4.7k est bien adaptée pour la plupart des utilisations, mais il faudra peut être la réduire si vous utiliser des longs fils afin de réduire les parasites en employant un plus fort courant. Certains capteurs analogiques peuvent nécessiter une autre valeur ou la suppression de la résistance de pull-up qui pourrait perturber la tension à mesurer.

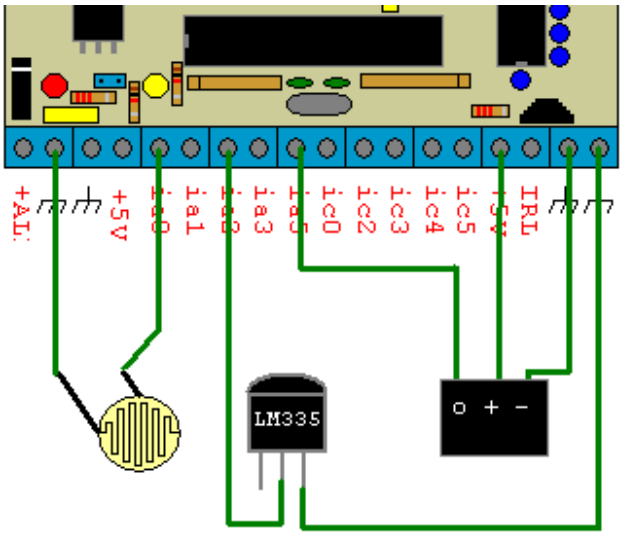
Les capteurs analogiques

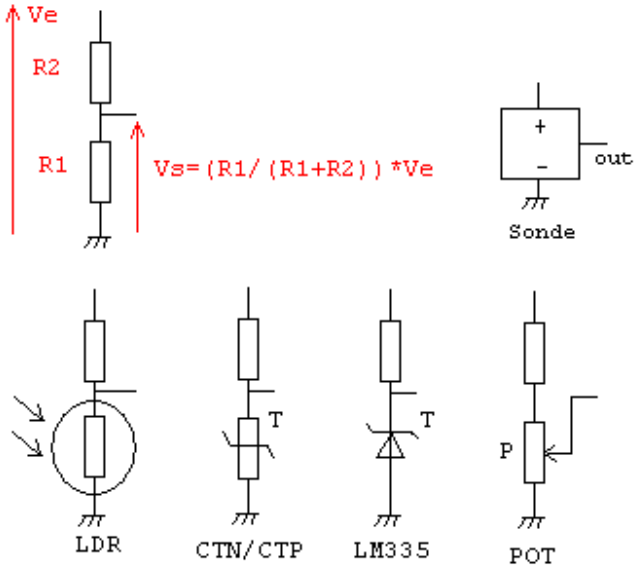
Les capteurs analogiques ont une caractéristique (résistance, tension,…) qui varie en fonction de la grandeur physique à mesurer. Ils ont donc une infinité de valeur possible contrairement aux capteurs TOR qui n'en ont que 2. L'API est capable de mesurer une tension sur le port A grâce au convertisseur analogique/numérique du PIC. Ce dernier donne une valeur comprise entre 0 et 255 en fonction de la tension d'entrée. 0 pour 0V et 255 pour 5V. Il suffit de faire une règle de 3 pour trouver les correspondances. Il est à noter que le PIC est capable de faire des mesures sur 10bits (0-1023), mais pour qu'une telle précision ait un sens, il aurait fallut faire un plus gros travail sur le découplage du UC et sur les plans de masse. Pour mesurer une résistance, il suffit d'utiliser un pont diviseur pour obtenir une tension qui varie en fonction de la résistance.

Voici quelques exemples de ces capteurs :

- LDR : « Light Dependant Resitor », une résistance qui varie en fonction de la lumière.
- CTN/CTP : sonde à coefficient de température négatif/Positif : Résistance qui varie en fonction de la température (qui augmente lorsque la température augmente pour les CTP)
- Capteurs potentiomètriques : Pour mesurer un angle ou une petite distance grâce à de bon vieux potard.
- Sonde active quelconque fournissant une tension en fonction de la grandeur à mesurer. Les sondes actives ont besoin d'être alimentées.
- LM335 : Une sorte de diode zener dont la tension inverse est proportionnelle à la température
- …

La figure suivante montre comment connecter une LDR, une sonde active et un LM335.





Le CAN peut uniquement mesurer des tensions, pour les capteurs résistif, il suffit d'utiliser un pont de résistance. La tension de sortie d'un tel pont se calcul par la formule Vs=(R1/(R1+R2))*Ve. Si Ve est 5V, R2 une résistance de pull-up d'un SIL et R1 la résistance du capteur : Vs=(R1/(R1+4700))*5V. Par exemple pour un potentiomètre variant de 0 à 10k : Vs=0V pour 0Ohms à 3.4V pour 10k Ohms. Le résultat de la conversion est can=(255/5)*Vs=51*Vs soit 173.

Le capteur LM335 fournit une tension de 2.73V + 0.01V/degrés. En mode 8 bits le can peut mesurer des tension de 5V/255=0.02V. Donc une précision de 2°C minimum est atteinte avec un branchement direct. Il est également possible de calibrer le capteur, en mettant un potentiomètre de 10 k sur la patte non utilisé, mais vu la précision médiocre de la mesure, ce n'est pas vraiment nécessaire. Un montage pour augmenter la variation est plus approprié. Ces montages visant à adapter un capteur sont connus sous le nom de conditionneur.

Le logiciel « progapi » permet de voir la mesure du CAN pour chacune des pattes du port A. Vous pouvez vous en servir pour étudier le comportement de vos capteurs. Il est également possible de tracer les diagrammes.

Les capteurs analogiques utilisés en TOR

Il s'agit simplement d'utiliser un capteur analogique et de définir un point de basculement au-delà duquel le capteur est inactif et en deçà actif. Ce point est réglé à 2.5V pour le port A et aux alentour de 2V pour le port C. Sous ces valeurs les capteurs sont vus comme actif et inversement (Les états sont inversés).

Prenons le cas d'une LDR dont la résistance vaut 1k lorsqu'elle est bien éclairée et 1M lorsqu'elle est dans l'obscurité. Avec les règles précédentes :

- Vs=(1000/(1000+4700))*5=0.88V éclairée (donc actif)
- Vs=(1000000/(1000000+4700))*5=4.98V dans le noir (donc inactif)

Il est ainsi possible de réaliser un interrupteur crépusculaire sans ce servir du CAN.

Les capteurs de barrières infrarouges

Les barrières infrarouges sont très utilisées dans les automatismes, mais les prix sont souvent prohibitifs. Afin de réduire le coût, je propose d'en réaliser avec de simples récepteurs de télécommandes infrarouges et de faire les traitements nécessaires dans le PIC.

Une barrière infrarouge est constituée d'un émetteur, le plus souvent une diode infrarouge qui émet un rayon infrarouge et un récepteur qui indique si le rayon est coupé ou non. Pour les récepteurs, il est possible d'utiliser des récepteurs normalement utilisés pour recevoir les signaux des télécommandes comme le TSOP1830, TSOP1836, TSM5360 … Les nombres 30 ou 36 identifient la fréquence de réception (30 à 38kHz en général). Il faut donc moduler la LED infrarouge à cette fréquence. Ces récepteurs ne sont pas faits pour recevoir en continue, il faut également émettre en discontinue.

Afin de se passer de circuits annexes, le UC génère tout seul et toute les 50ms un signal de 30kHz pendant 3ms. Ceci ce fait à l'aide d'un module PWM configuré à 30KHz 50% qui contrôle la patte C1 du UC, ce qui explique pourquoi, l'entrée ic1 n'existe pas. Le transistor T1 amplifie ce faible pour fournir jusqu'à 100mA aux leds infrarouge.

Ce transistor se comporte comme un interrupteur avec la masse, il faut donc brancher les leds entre le +5V et son collecteur accessible via CN1 sous le nom IRL. La résistance de chaque led infrarouge est fonction de l'intensité souhaitée. Par exemple avec 270Ohms, il est possible de faire une barrière d'1.5m (I=U/R=(5-Vled)/R=(5-2)/270=0.011=11mA). Bien entendu, plus le courant est élevé et plus la portée de la barrière est élevée. Il est également possible de mettre plusieurs diodes et d'utiliser le +ALIM pour plus de puissance optique.

La partie émetteur constituée d'une led infrarouge LDIR1 et d'une résistance de limitation R7.

Le cœur du récepteur est un récepteur IR TSOP1830, TSOP1836 ou TSM5360. Il est alimenté en 5V au travers du circuit R6, C11. Outre le fait de fournir une alimentation stable, ce dispositif filtre également et efficacement la perturbation de 30kHz que véhicule l'alimentation. Cette perturbation infime est générée bien entendu par l'émetteur est suffit à verrouiller la PLL du récepteur qui indique ainsi qu'un rayon est reçu alors qu'il n'en est rien.

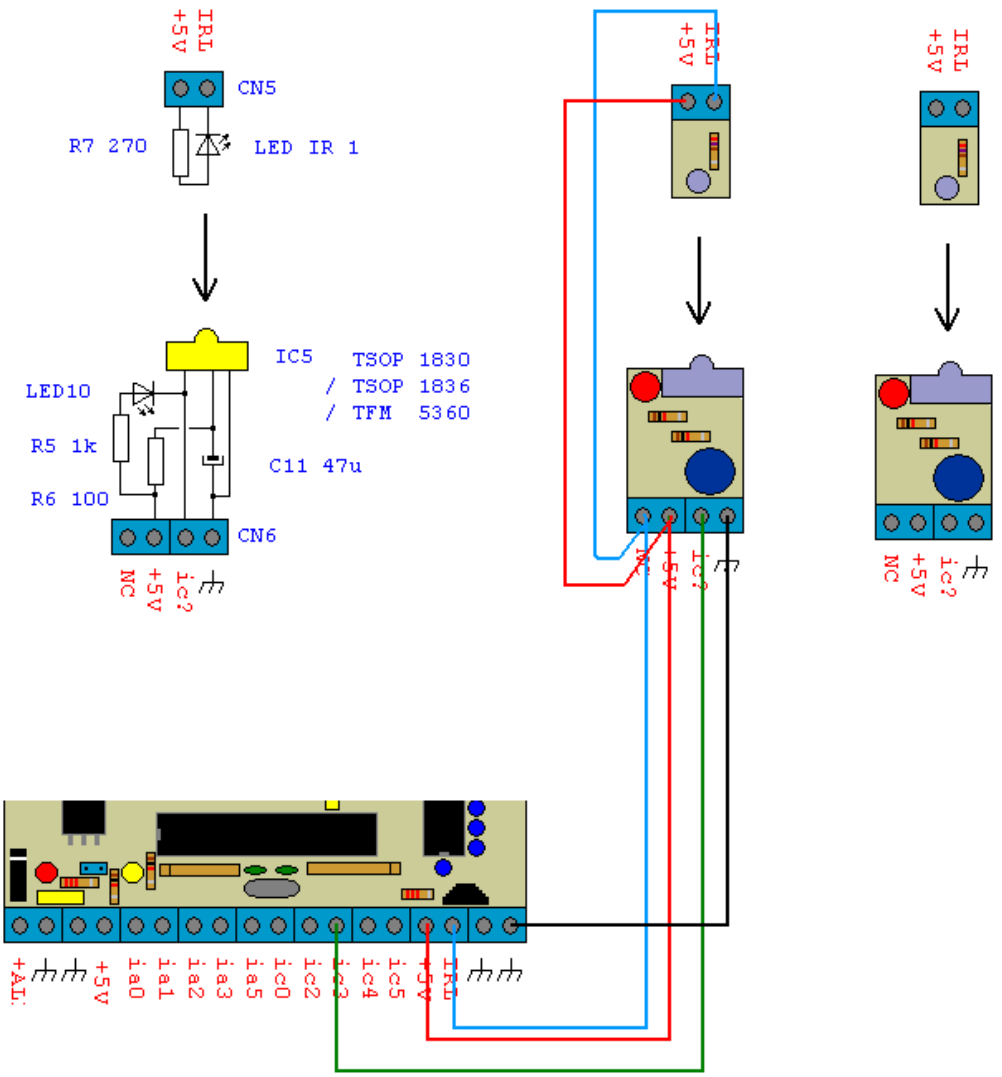
La led10 dont le courant est limité par R5, indique la présence d'un rayon. En effet, lorsque le récepteur détecte un rayon, il met sa sortie à collecteur ouvert à la masse, ce qui allume la led et active l'entrée correspondante du PIC. Je conseille d'utiliser une led haute luminosité car en présence d'un rayon, la sortie est active durant 3ms toutes les 50ms soit 6% du temps.

Le UC doit lire les entrées du port C à un instant bien précis, ceci explique qu'il faille utiliser les entrées irc0,2-5 à la place des entrées ic0,2-5. Les 5 entrées C sont capables de recevoir une barrière infrarouge, il est donc possible d'utiliser au maximum 5 barrières.

En présence d'un rayon, la led doit scintiller à 20Hz. Un scintillement discontinu indique que la réception n'est pas parfaite et qu'il faut y remédier sous peine de fournir de fausses informations. Ceci peut se faire en réduisant la longueur, en alignant mieux l'émetteur et le récepteur ou en augmentant la puissance optique en utilisant un plus grand courant pour la led IR.

Bien entendu, le récepteur peut être perturbé par d'autres dispositifs infrarouges comme une télécommande, il faut en tenir compte pour les applications critiques.

La figure suivante présente le schéma électrique et montre comment connecter une barrière.



C.3 Etude des sorties

Les 8 lignes du port B du UC sont utilisées par les sorties. A ce niveau, l'intensité maximale est de 20mA par sortie sous 5V. Le réseau de transistor darlington IC2, amplifie en courant ces sorties et permet d'utiliser une tension différente de 5V. Lorsqu'une sortie du PIC est à 1 (5V), l'entrée correspondante de l'ULN voit 5V, et relie sa sortie correspondante à la masse. Dans le cas inverse la sortie n'est reliée à rien du tout. Le circuit accepte un courant maximal de 500mA par sortie sous une tension de 50V max. Les sorties de l'ULN pilotent des relais électromagnétiques. Les contacts des relais sont disponibles sur le connecteur CN2 et l'utilisateur peut donc y connecter ses actionneurs.

L'utilisation de relais a plusieurs avantages :

- Il n'y a pas de liaison électrique entre les sorties et le reste de l'API. Il est ainsi possible d'utiliser du 220V.
- Il est possible d'utiliser aussi bien de l'alternatif que du continu.
- Il est possible de commuter de grosses puissances.

Les relais que je préconise ont les caractéristiques suivantes : 1T, 12V, 1200Ohms, 5A 30V CC, 5A 250V AC. Ces caractéristiques sûrement très obscures pour le néophyte ;-) signifient :

- 1T : Il n'y a qu'un seul interrupteur qui relie les pattes C (commun) et T (travail) du relais lorsque la bobine est alimentée. Il existe des relais dit 1RT qui possèdes un inverseur, c'est-à-dire qu'au repos les pattes C et R (repos) sont reliés. Les pattes C et T étant connectés au travail c'est-à-dire lorsque la bobine est alimentée. Il existe aussi des relais avec plusieurs inverseurs, par exemple 2RT …

- 12V, 1200Ohms sont les caractéristiques de la bobine. J'ai choisi une résistance très élevée afin de consommer le moins possible. En effet, un courant de 10mA suffit à activer ce relais. Pour cette raison, j'ai pu mettre en série avec le relais des leds pour indiquer l'état des sorties. Si vous utilisez un autre type de relais, il sera sans doute impossible d'utiliser les leds en série. Il faudra alors les mettre en parallèle aux bobines avec une résistance de protection. J'ai retenu la valeur 12V au lieu de 5V pour alimenter les relais avec la tension disponible en amont du régulateur afin de charger ce dernier le moins possible.
- 5A, 30V CC. Le relais est capable de déconnecter un courant de 5A maximum en continue. Avec un courant plus important, le relais peut rester en position travail, même avec la bobine désactivé. Les contacts sont également dimensionnés pour cette intensité. Il est aussi déconseillé d'utiliser plus de 30V continu.
- 5A 250V AC. De même l'intensité maximale est de 5A en alternatif. Il est possible d'utiliser le relais jusqu'à 250V, vous pourrez donc commuter des charges de 1100W max sur le secteur (5A*220V). La tension maximale est bien plus grande qu'en continu car le courant alternatif passe par 0 A et le relais peut donc toujours être déconnecté, ce qui n'est pas le cas du continu.

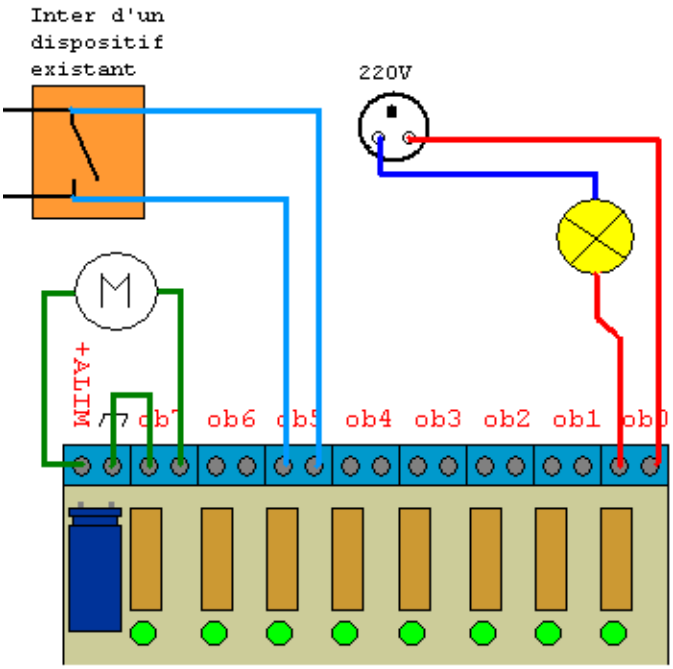
Il est bien entendu possible d'utiliser d'autres relais si vous ne trouvez pas ces relais assez particulier tant par leur format SIL que par leur basse consommation, ou si vous avez besoin d'autres caractéristiques. Pensez alors à supprimer les leds ou à les mettre en parallèle et à modifier le CI.

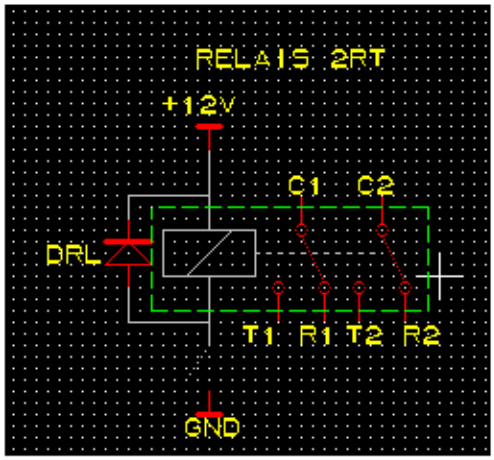
Lorsque l'alimentation d'une bobine est coupée pour désactiver un relais, cette dernière réagit en délivrant une haute tension inverse de l'ordre de 100V capable d'endommager les circuits électroniques. Pour éliminer cette tension dangereuse, il est nécessaire d'utiliser une diode de roue libre DRL en parallèle avec la bobine. Le circuit IC2 possède également des DRL. Il suffit pour les utiliser de connecter la patte « commun diode » à l'alimentation des relais.

Il est bien entendu possible d'utiliser à la place des relais, d'autres solutions comme des triacs en alternatif ou encore branchez directement une charge continue sur l'ULN ou des leds en sortie du PIC. Mais la solution à relais est la plus universelle.

La figure suivante montre :

- un relais 2RT avec une DRL
- une lampe 220 V pilotée par la sortie ob0
- un moteur d'un ventilateur 12V piloté par ob7 tirant son alimentation de l'automate (s'il est alimenté en 12V)
- un dispositif existant pouvant être piloté par un interrupteur. La sortie ob5 permet maintenant de le contrôler.





C.4 Etude de l'alimentation

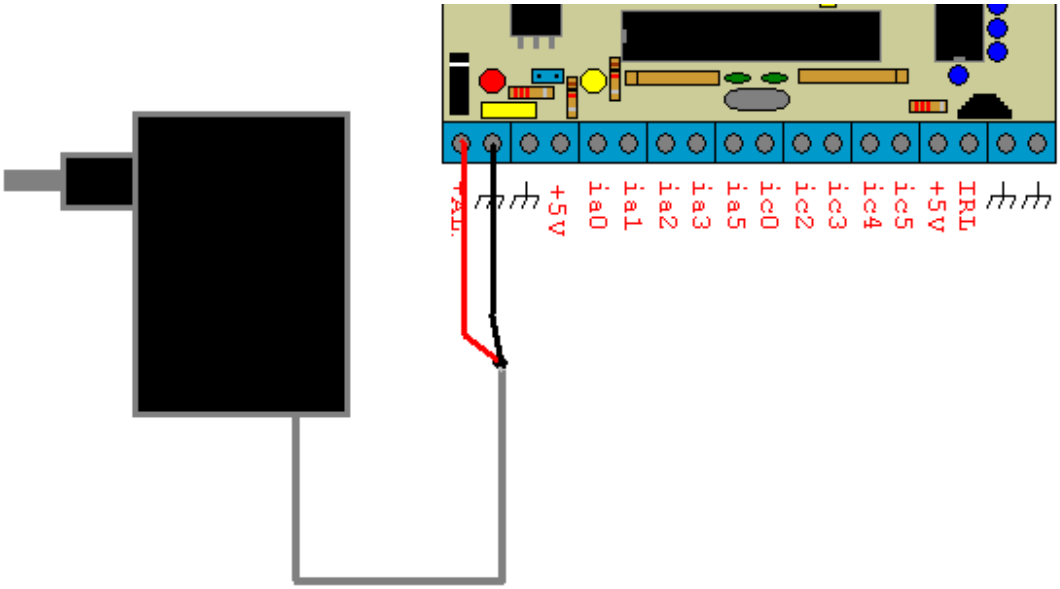
Afin de ne pas alourdir la réalisation et pour autoriser le fonctionnement avec une batterie, je n'ai pas inclus l'alimentation 220V sur la carte. Vous pouvez néanmoins l'ajouter simplement en vous aidant d'articles sur le sujet. En effet, il suffit juste de rajouter un fusible, un transformateur et un redresseur.

Je propose d'utiliser un bloc secteur tout simple qui n'a pas besoin d'être régulé. Un modèle des plus banale comme 12V, 500mA suffit amplement. Sa tension doit être en rapport avec les relais utilisés. Notez que la tension à vide peut monter jusqu'à 12V*1.4=16.8. Mais elle redescend lorsque l'on tire de l'intensité. Ceci ne pose aucun problème.

Il est également possible d'utiliser une batterie 12V pour les applications autonomes.

La diode D1 protège le montage en cas d'erreur de branchement. Le condensateur C1 lisse la tension procurée par le bloc secteur. La led8 dont le courrant est limité par R1 indique la présence de la tension. Le régulateur IC4 fournit du 5V bien régulé à l'ensemble du montage. Le condensateur C2 filtre cette tension et élimine les parasites hautes fréquences.

La tension de l'alimentation est également disponible sur le connecteur CN2 pour des actionneurs compatibles avec elle. Dans le cas d'utilisation de cette tension pour alimenter un actionneur, il est nécessaire de calculer l'intensité de l'alimentation en conséquence, de dimensionner la diode D1 et d'augmenter la valeur de C1. Sauf pour de très petits actionneurs, je déconseille d'utiliser l'alimentation de l'API pour les actionneurs.



C.5 Etude de la partie communication RS232

Cette partie permet de faire dialoguer le PIC et le PC à travers une liaison série RS232. Il est ainsi possible de télécharger le graphcet et d'observer le comportement de l'API.

Un bout de code du firmware connu sous le nom de bootloader/moniteur (bmon) permet de lire et modifier n'importe quelle mémoire du PIC (RAM, FLASH, EEPROM, registres). A la mise sous tension, si le cavalier de CN3 est présent, alors seul bmon fonctionne. J'avais prévus ceci afin de pouvoir charger un nouveau firmware, mais ce n'est pas nécessaire actuellement. Dans le cas contraire, le firmware appelle bmon à chaque IT, toute les 330us. Il est ainsi possible d'observer le fonctionnement de l'interpréteur. Pour charger le graphcet, progapi stoppe l'interpréteur, charge le graphcet puis réinitialise et relance l'interpréteur. Avec bmon, il est possible de se servir du montage comme d'une interface ou pour déboguer un programme PIC.

La liaison RS232 est une liaison série normalisée et très utilisée dans le monde de l'informatique. Vous devriez trouvez 2 ports sur votre PC sous la forme de 2 DB9 mâle (ou 1 DB25 mâle + 1DB9 mâle pour les anciens PC). Si vous n'en trouvez pas sur certains portables ou sur les futures PC, il faudra utiliser un câble USB-RS232. Sur une liaison RS232, les bits informations circulent les uns après les autres sur le même fil. Le nombre de fil est alors réduit à son plus simple minimum (une masse, un fil pour le sens PC-->API et un autre pour le sens API-->PC). Les niveaux sur cette liaison sont -12V pour un « 1 » et +12V pour un « 0 ». Ils sont donc totalement incompatibles avec ceux du PIC +5V pour « 1 » et 0V pour « 0 ». Le circuit IC3, un célèbre MAX 232 se charge d'adapter ces niveaux. Les multiples condensateurs servent à la pompe de charge intégrée qui permet de produire du +/-12V à partir du 5V de la carte.

Je n'ai pas mis directement un DB9 sur la carte, il faut donc réaliser un adaptateur DB9. Vous pouvez donc souder directement 4 éléments femelles sur un DB9 puis l'enficher sur CN3 puis utiliser un câble série droit DB9-F(femelle), DB9-M(mâle). Ou alors confectionner un câble avec les 4 éléments d'un coté et un DB9-F de l'autre.

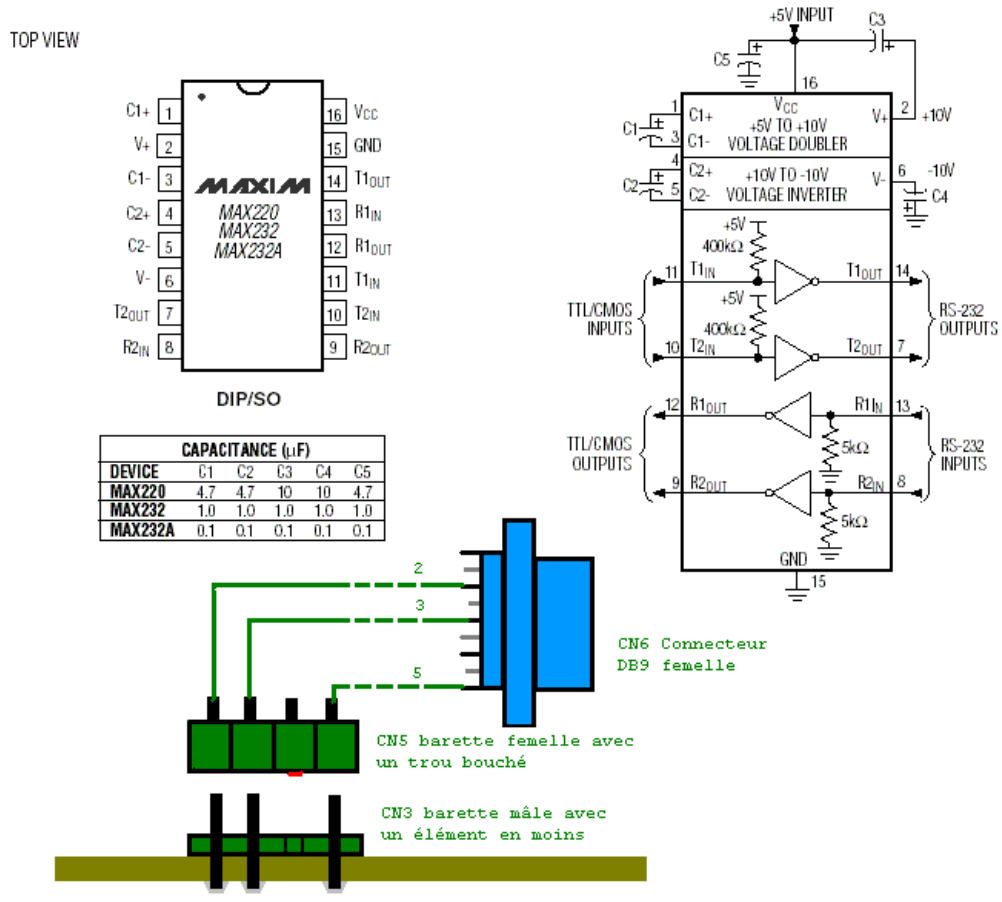
Si vous utiliser un vieux DB25 alors inverser les broches 2 et 3 et remplacer la 5 par la 7.

Bmon est conçu pour être adressable, c'est-à-dire qu'avec un montage quelque peu différent, il serait possible de brancher jusqu'à 16 automates. Avec le câblage actuel, ce n'est pas possible mais il faut néanmoins configurer progapi pour utiliser l'adresse utilisée actuellement et fixée à 3.

Il est également possible de se passer de la partie RS232. Dans ce cas vous ne pourrez pas télécharger le graphcet par la liaison série, ni observer le comportement de l'automate. Pour programmer le graphcet, il suffit alors de flasher le UC avec icprog et JDM grâce au fichier .hex fournit par progapi.

Il serait également possible d'utiliser un seul transistor à la place du MAX232 pour les communications du PC vers le PIC. Mais sans retour, il est impossible de savoir si le graphcet est bien programmé. Je ne détaille donc pas cette solution.

De toute façon, l'utilisation de la RS232 rend l'API tellement convivial que vous ne pourrez plus vous en passer.



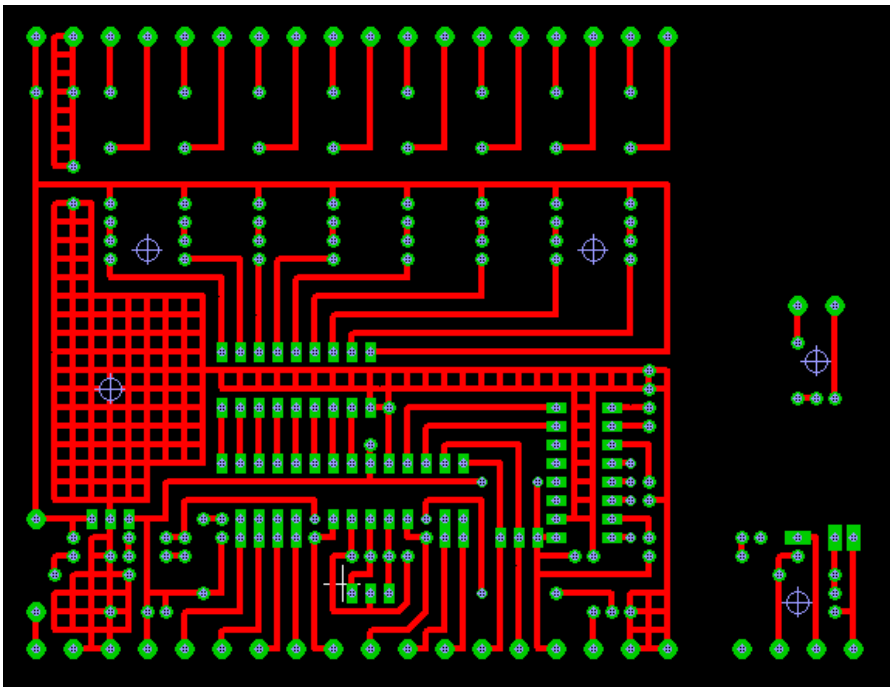
C6. Réalisation

Pour réaliser les automates, vous pouvez utiliser une des méthodes suivantes :

- La plus simple est d'utiliser une plaque percée avec des pastilles de cuivre. Il ne reste alors plus qu'à souder les composants et à les relier entre eux par des fils. Mais le résultat n'est pas très propre.
- La plus belle est la méthode photographique qui passe par l'impression des typons sur transparent, puis insolation d'une plaque de cuivre/époxy pré-sensibilisée, puis gravure au perchlorure de fer III … Certains magasins d'électronique proposent de réaliser des CI avec vos typons.
- Une méthode intermédiaire est d'utiliser une mini perceuse avec une fraise à graver le verre pour graver les inter pistes. Le résultat dépend bien entendu de la dextérité de l'artiste mais devrait être satisfaisant.

Pour vous aidez, voici les typons de l'automate 10E/8S. La taille est sûrement mauvaise, lors de l'impression veiller à bien redimensionner. Si vous ne pouvez pas redimensionner à l'impression alors utiliser la fonction Zoom In/Out d'une photocopieuse. Le moyen le plus sûre de vérifier les dimensions est de mesurer les 2 trous extrêmes de CN1 qui doivent être espacés de 17*2*2.54mm=86mm=8.6cm

Vous pouvez également vous servir du fichier .tci de l'archive et de l'excellent freeware TCI de Bruno Urbani (http://b.urbani.free.fr)



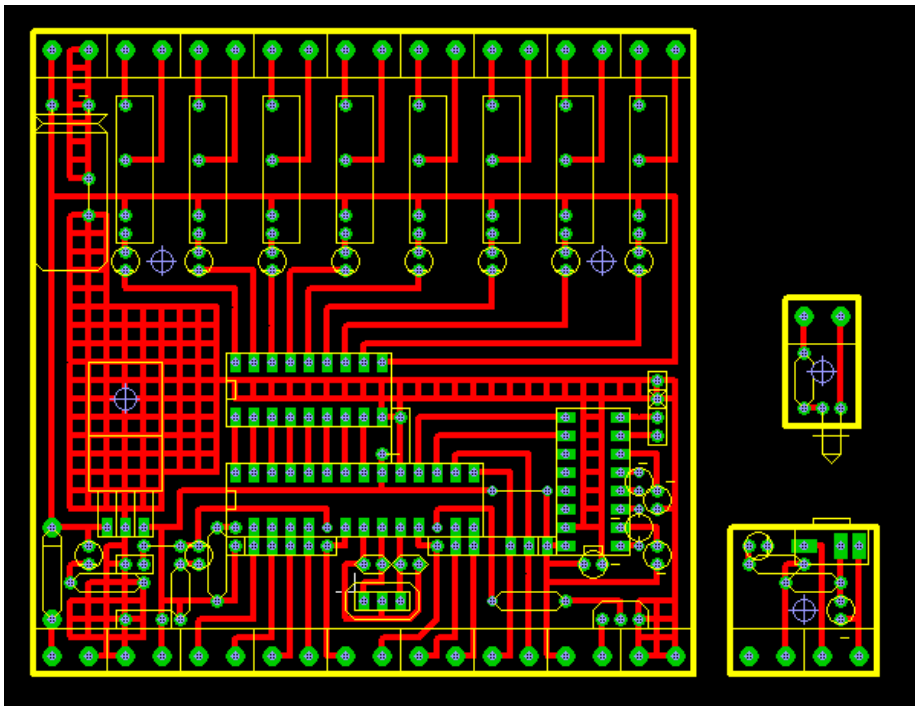
Afin de rendre la réalisation aussi simple que possible, j'ai réalisé un typon simple face avec de larges pistes qui ne passent pas entre les pattes des composants. Ceci à un prix : Une taille plus grande qu'en double face et le recours à 3 straps.

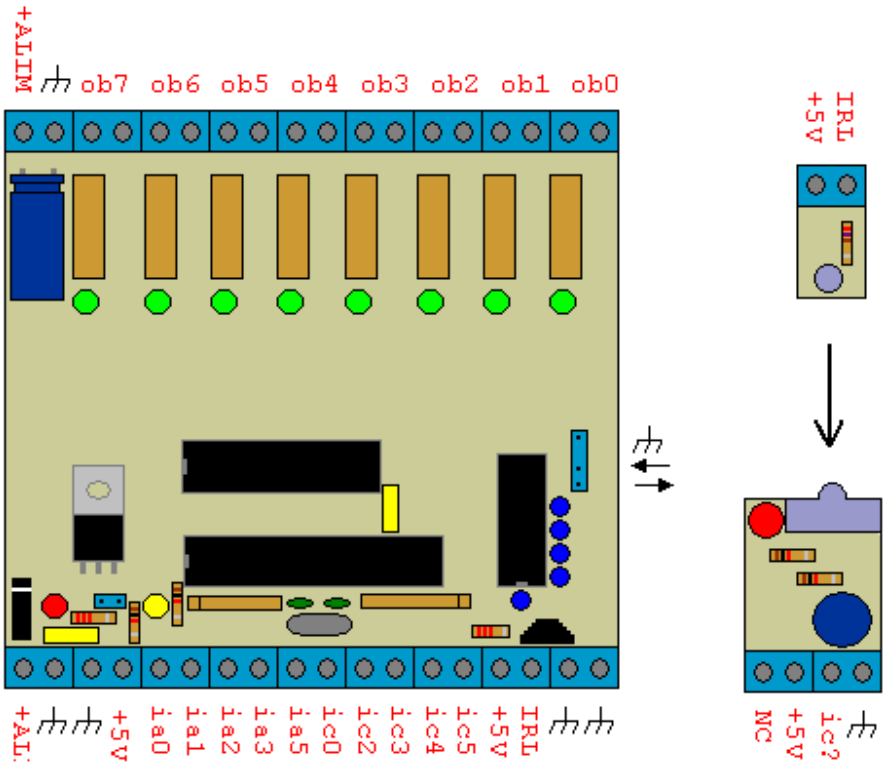
Pour l'étape de soudage, commencer par les plus petits composants afin de ne pas être gêné par la suite. Ne soudez pas les composants à l'envers. Les pattes – des condensateurs sont repérées par un – sur la sérigraphie. Celles des leds par un trait qui symbolise le méplat. Je n'ai pas mis les numéros des composants sur les typons, mais ils sont facilement repérables.

Pour tester le montage, procéder comme suit :

- vérifier l'absence de court circuit entre les pistes avec un ohmmètre.
- enlever tous les circuits et brancher l'alimentation. La led 8 doit s'allumer. Vérifier la présence du 5V et de la masse sur les supports des circuits.
- Brancher un fil entre la patte 9 de l'ULN (masse) et les pattes 18-11 tour à tour afin de vérifier les relais.
- Mettre l'ULN et brancher un fil entre la patte 20 du support du PIC (+5V) et les pattes 21-28 afin de vérifier le fonctionnement de l'ULN.
- Mettre le MAX232 et mettre un fil entre les pattes 17 et 18 du support du PIC afin de reboucler le lien série en passant par le MAX. Lancez l'hyper terminal sur le PC et regarder si vous recevez les caractères tapés.
- Flasher le PIC avec JDM / icprog puis le mettre sur son support. La led 9 doit clignoter

Attention, il est impératif de couper l'alimentation, lorsque vous mettez en place ou retirez le PIC ou le MAX232.





Vous pouvez acheter les composant électronique dans la boutique électronique de votre ville ou commander par correspondance chez de grande enseignes comme Selectonic (C'est ici que j'ai trouvé mes relais basse consommation) ou Conrad. Notez que vous pouvez demander des échantillons gratuits pour le PIC sur le site de Microchip (http://www.microchip.com) et pour le MAX232 sur le site de Maxim (http://www.maxim-ic.com)

D. Le graphcet

Soyez patient ! Je sais que vous voulez voir si l'automate fonctionne, mais comme je l'ai dit l'automate se programme avec le graphcet et il faut donc apprendre le graphcet .

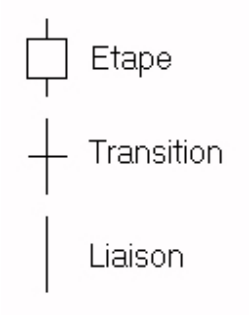
D.1 Présentation

Le graphcet pour Graphe de Commande Etapes Transition est un outil extrêmement puissant et bien adapté pour représenter la commande d'un système. C'est une adaptation des réseaux de Pétris à la commande des systèmes. De plus il à l'avantage d'être normalisé et bien utilisé dans l'industrie.
Comme son nom l'indique, il s'agit d'un graphe. Cette représentation est donc agréable à utiliser, intuitif et facilement accessible au non initié.

Les 3 éléments graphiques de bases sont :

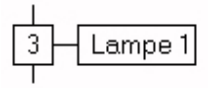
- les étapes
- les transitions
- les liaisons

Les liaisons permettent de joindre les étapes et les transitions (à angle droit d'après la norme) . La figure suivante montre ces éléments.

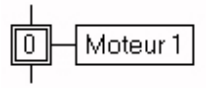


D.2 Les étapes

Les étapes représentent une position dans laquelle est le système. Une étape peut être active ou inactive. L'ensemble des étapes actives indique l'état global du système. On associe généralement les sorties aux étapes, il s'agit alors d'un automatisme de Moore. De plus les étapes portent des numéros.
La figure suivante indique que lorsque l'étape 3 est active alors il faut allumer la lampe 1.

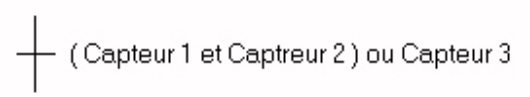


Ainsi lorsqu'une étape est active, la ou les sorties associées sont activées.
En fait il existe deux sortes d'étapes, les étapes "normales" et les étapes "initiales" qui s'activent à l'initialisation, les "normales" étant bien entendu inactives.
La figure suivante montre une étape initiale.



D.3 Les transitions

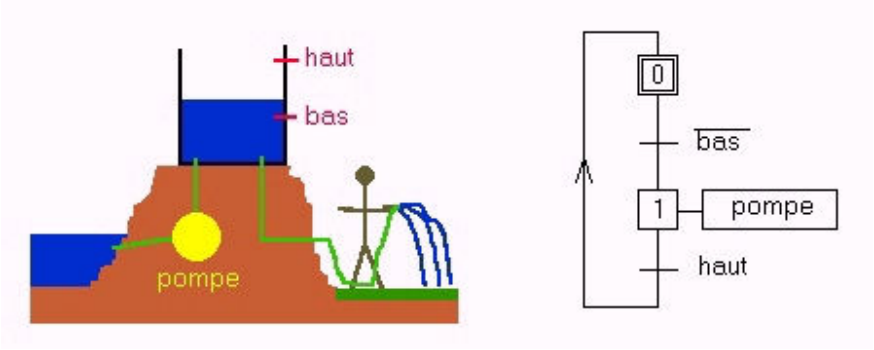
Les transitions permettent de passer d'une étape à une autre.
Les transitions sont reliées aux entrées le plus souvent et possèdent une équation logique. Si cette expression logique est vérifiée (vrai) alors la transition est dite réceptive ou active.
La figure suivante montre une étape avec son équation logique.



Si une transition et l'étape qui la précède sont actives alors l'étape précédente devient inactive et l'étape suivante active.

D.4 Un exemple simple

Pour illustrer ces notions, prenons un exemple simple. Il s'agit de remplir une réserve d'eau avec une pompe. Lorsque le capteur bas n'est plus recouvert alors la pompe se met en route, cette dernière s'arrête lorsque le capteur haut est recouvert . Un capteur recouvert vaut 1, sinon 0.



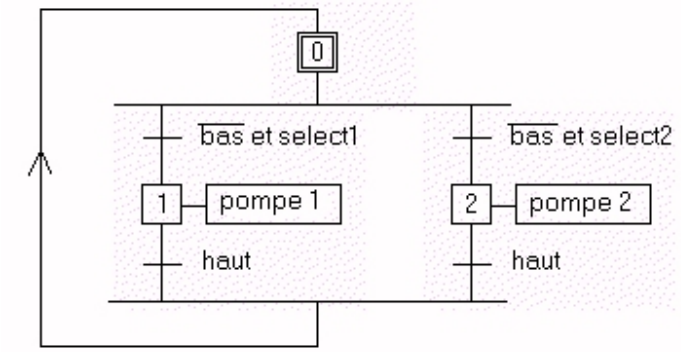
A l'initialisation, l'étape 0 est active et la 1 inactive. La pompe est arrêtée car aucune étape ne la commandant (ici la 1) est active.
Lorsque le niveau d'eau passe sous le capteur bas, alors celui ci n'est plus recouvert, donc la transition "/bas" est vérifiée. Il s'en suit une désactivation de l'étape 0 et une activation de l'étape 1 d'ou la mise en route de la pompe. Grâce à la pompe, le niveau d'eau monte et lorsqu'il recouvre le capteur haut, alors la transition "haut" devient active d'ou la désactivation de l'étape 1 et de la pompe et l'activation de l'étape 0. Et le cycle recommence. Ainsi le jardinier peut arroser son jardin tranquillement. J'entend les murmures ironiques du genre "Y à t'il besoin d'un UC la ou une simple bascule RS suffit ?", certe non mais c'est un exemple simple pour comprendre.

D.5 Le parrallélisme

Un des grands avantages du graphcet est le parallélisme, en effet dans un graphcet il peut y avoir plusieurs étapes actives en même temps. Il peut même y avoir plusieurs graphcets qui tournent simultanément ! Ainsi vous pouvez commander la pompe précédente en même temps que la porte du garage, les lampes de la maison et une alarme, et ceci avec le même API ! Intéressant non ? Tout ceci sans que les graphcets ne se gênent mutuellement. On peut aussi les faire interagir entre eux (exemple : l'alarme s'arme quand on ferme la porte du garage) ...

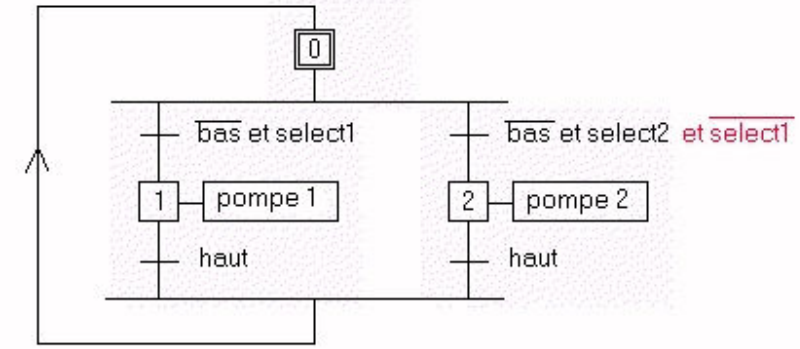
D.6 La divergence en "OU"

Le premier graphcet était linéaire, mais comme le graphcet s'apparente à un programme, il doit être capable d'effectuer des choix. En effet à partir d'une étape, on doit pouvoir aller à d'autres étapes suivant un choix.
Par exemple, on utilise maintenant 2 pompes et les entrées sélect1 et sélect2 sélectionnent la pompe à engager.



Mais attention, le parallélisme fait que si les deux transitions sont réceptives en même temps alors les étapes 1 et 2 s'activent simultanément. Ici ce n'est pas grave, le bassin se remplira plus vite, de plus la condition d'arrêt étant la même ... Mais cela peut poser des problèmes graves dans certains cas, en effet imaginons que l'étape 1 fasse monter un ascenseur et la 2 le fasse descendre ! De plus si les conditions d'arrêt ne sont pas les mêmes alors il se pourrait que l'étape 1 termine, ce qui active l'étape 0 alors que la 2 est encore active d'ou un désordre grave et dangereux ...

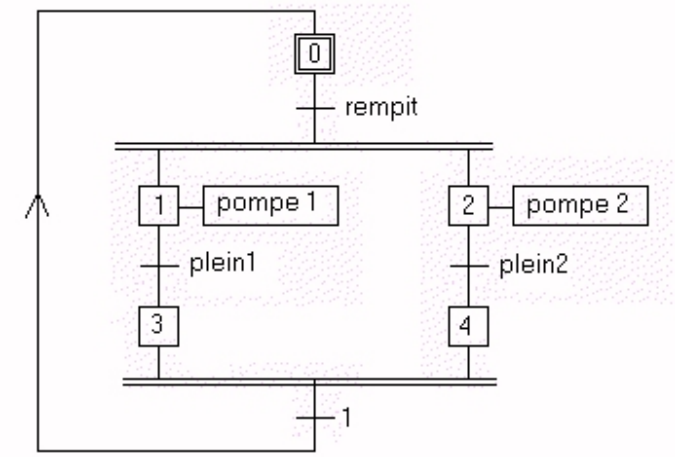
Bref si on veut vraiment faire un choix, alors il faut que les transitions soient exclusives, c'est à dire qu'elles ne puissent pas être réceptives en même temps. La figure suivante montre le graphcet modifié.



Comme vous le voyez la divergence en "OU" est représentée par une simple barre horizontale.

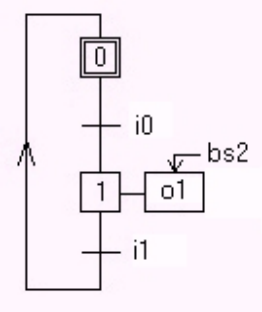
D.7 La divergence en "ET"

La divergence en "ET" permet de lancer plusieurs actions en parallèle, il ne faut donc pas oublier de les synchroniser à la fin afin de ne pas perdre des étapes actives en chemin.
Dans l'exemple suivant, il y a maintenant deux réservoirs, chacun ayant un capteur plein qui passe à "1" lorsqu'il est plein . Lorsque l'on appuie sur le bouton remplit alors les deux réservoirs se remplissent. Le graphe ci dessous montre ce fonctionnement.



A l'initialisation, l'étape 0 est active et donc les pompes sont à l'arrêt. Lorsque l'on appuie sur le bouton remplit, les étapes 1 et 2 s'activent en même temps alors que la 0 se désactive.
Quand le réservoir 1 est plein, le capteur plein1 rend la transition active d'ou une désactivation de l'étape 1, de la pompe1 et une activation de l'étape 3 qui attend que la 4 soit active.
De même , quand le réservoir 2 est plein , le capteur plein2 rend la transition active d'ou une désactivation de l'étape 2, de la pompe2 et une activation de l' étape 4 qui attend que la 3 soit active .
Le franchissement de la double barre ne peut se faire que si les étapes directement au dessus sont actives et si la condition est vérifiée. Ici la condition est toujours vérifiée car égale à 1, donc dès que les étapes 3 et 4 sont actives , on retourne à l'étape initiale .
La divergence en "ET" est représentée par une double barre horizontale.

D.8 Sorties conditionnelles



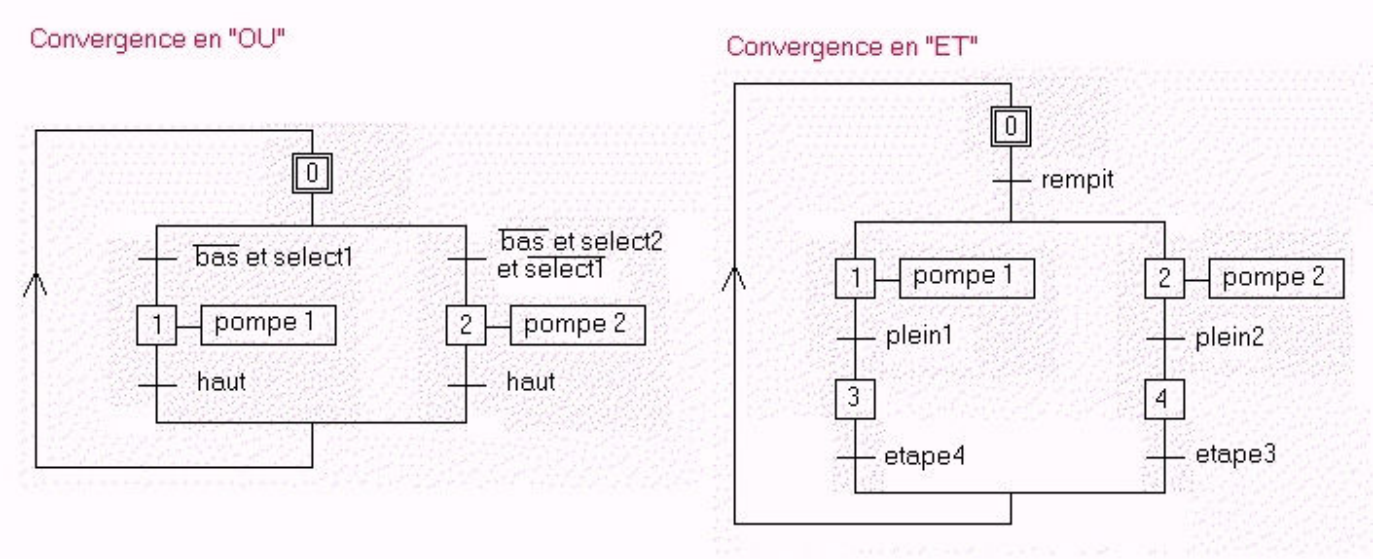
La flèche au dessus de l'action de l'étape 1 indique que la sortie est à activer lorsque l'étape 1 est active si bs2 est vrai. (bs2 peut être n'importe quelle expression, comme le plus souvent un bit clignotant, ce qui à pour effet de faire clignoter la sortie très simplement lorsque l'étape 1 est active).

D.9 Le reste

La norme ne s'arrête pas là, il y a encore des pages et des pages, mais vous en savez maintenant suffisamment. N'hésitez pas à consulter des ouvrages sur ce sujet. Un outil complémentaire et intéressant est le GEMMA ou Graphe d'Etude des Modes de Marche Arrêt qui est un tableau avec des cases à remplir. Dans ce tableau, tout les problèmes pouvant arriver au système sont recensés, à vous de décider si vous les traitez ou pas et comment. Exemple, coupure d'alimentation, faut il réinitialiser, Une machine de la chaîne est en panne, faut il produire en dégradé ou tout arrêter, comment traiter le bouton d'arrêt d'urgence ... Dans cette approche on construit d'abord le graphcet en fonctionnement normale puis on rajoute des bouts de graphcet au graphcet normale afin de réagir aux problèmes. Notez que la plupart des automatismes fonctionnent avec une association graphcet / GEMMA.

D.10 Particularités du graphcet utilisé

Soyez rassurés, tout ce que l'on a vu auparavant, l'automate peut le faire et le fait même très bien. Afin de simplifier mon travail pour la réalisation des logiciels, vous fabriquerez vous même vos convergences en "OU" et en "ET" , les notions de simple et double barres n'existe pas chez moi , mais vous pourrez bien entendu faire le même comportement . La figure suivante reprend les graphes avec ma notation.



Avouez que cela ne change pas grand chose.

E. Le firmware ou interpréteur du UC

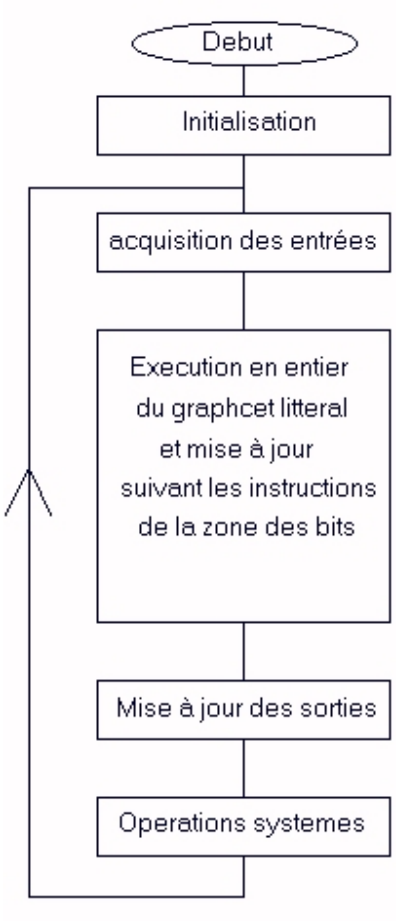
Les logiciels enfouis dans les UC sont souvent appelés firmware. Celui de l'API est constitué de 2 éléments.

- Le premier « bmon » précédemment décrit permet au PC de lire et écrire dans les différentes mémoires du PIC pour par exemple télécharger le graphcet ou indiquer les étapes actuellement actives ou le résultats des conversions du CAN pour chacune des pattes du port A.

- Le second est « l'interpréteur » de graphcet chargé comme sont nom l'indique d'interpréter le graphcet stocké dans sa mémoire flash.

E.1 Fonctionnement global de l'interpréteur

Ce programme est chargé d'exécuter le graphcet au niveau de l'API. Il dispose d'une table où est rangé le graphcet littéral à interpréter et d'une zone en RAM appelée zone des bits qui indique l'activité des entrées , sorties , étapes , bits internes et bits systèmes . Ces 2 derniers termes seront expliqués au niveau du graphcet littéral.
Son fonctionnement est le suivant.



Les opérations systèmes consistent à mettre à jour différents bits spéciaux comme le bit de démarrage uniquement actif durant la première interprétation, les bits constants « 0 » et « 1 », le bit de verrouillage de l'horodateur indiquant qu'il est à l'heure ainsi que diverses bits de clignotement.

C'est également à ce niveau que la zone des bits est copiée dans l'ancienne zone des bits afin de détecter par la suite des fronts et travailler sur une base qui ne change pas car la zone des bits est mise à jour par les instructions.

Ce programme à été fait en assembleur et est optimisé pour la vitesse, la programmation du PIC en assembleur est un vrai régal !

E.2 Le graphcet littéral

Le programme progapi coté PC à convertit le graphe sous forme d'un tableau composé d'instructions graphcet et d'arguments. Ce langage intermédiaire est connu sous le nom de grapchet littéral.

Le graphcet littéral est un petit langage qui est sensé représenter un graphcet.

L'interpréteur travaille sur une zone mémoire appelée zone des bits ZB. Cette zone de 32 octets permet de coder l'état du système sur 256 bits. Chaque bit représente quelque chose.

- Les bits 0 à 79 servent généralement pour stocker l'état des étapes
- Les bits 80 à 175 peuvent servir pour stocker des résultats intermédiaires.
- Les bits 176 à 223 reflètent l'état des E/S
- Les bits 224 à 255 sont des bits spéciaux de clignotement, démarrage, constants (0 et 1), verrouillage de l'horodateur …

La plupart des instructions graphcet manipulent ces bits à travers un bit spécial appelé indicateur. Il est ainsi possible de charger ce bit avec un bit de la ZB et inversement de le sauver. Les opérations logiques et, ou, ou exclusif sont également possible entre lui et les bits de ZB. Il est aussi possible de travailler avec les compléments des ZB. La liste suivante résume ces instructions :

- l bit : chargement de l'indicateur avec la valeur de bit
- ln bit : chargement de l'indicateur avec le complément de la valeur de bit
- a bit : et logique entre l'indicateur et le bit (indicateur = indicateur et bit)
- an bit : et logique entre l'indicateur et le complément du bit (indicateur = indicateur et /bit)
- o bit : ou logique entre l'indicateur et le bit (indicateur = indicateur ou bit)
- on bit : ou logique entre l'indicateur et le complément du bit (indicateur = indicateur ou /bit)
- x bit : ou exclusif entre l'indicateur et le bit (indicateur = indicateur xor bit)
- xn bit : ou exclusif entre l'indicateur et le complément du bit (indicateur = indicateur xor /bit)
- = permet d'affecter un bit de la valeur de l'indicateur.
- s bit : met à le bit à 1 si l'indicateur est à 1
- c bit : met le bit à 0 si l'indicateur est à 1

Bien entendu un code correspond à chaque instruction, par exemple l'instruction ln est codée par 96.

Afin de pouvoir détecter les fronts et travailler sur une base qui ne bouge pas, l'interpréteur dispose d'une ancienne zone des bits OLD_ZB. Il est possible d'utiliser ces bits, mais pas de les modifier.

Les étapes se représentent par * bit pour les initiales et – bit pour les autres. Lorsque l'interpréteur rencontre ces instructions, il met à 1 un bit appelé bit de saut si l'étape indiquée est active dans l'ancienne zone des bits et inversement. Pour changer d'étape, on utilise > bit. Le changement survient si l'indicateur est à 1 et si le bit de saut est à un. Dans ce cas, la nouvelle étape est activée dans ZB et la courante désactivée dans la ZB.

L'étape est représentée par le bit indiqué. Il est d'usage d'utiliser les 80 premiers bits pour les étapes, mais ce n'est pas une obligation.

Une dernière instruction représentée par ! indique la fin du graphcet

Pour résumer :

- * bit : étape initiale (met à jour le bit de saut)
- - bit : étape standard (met à jour le bit de saut)
- > bit : changement d'étape (si l'indicateur et le bit saut sont à 1)
- ! : fin du graphcet. L'argument n'est pas utilisé dans ce cas.

Ces nouveaux API sont également capables de travailler sur des variables 8 et 16 bits contenus dans la zone des variables ZV. Il est possible de mettre à jour l'indicateur grâce à des instructions de comparaison entre une variable et une constante appelée littéral. Bien entendu, ces instructions occupent 2 fois plus de mémoire.

- l/ln/a/an/o/on/x/xn var8bits ==/>/</>=/<=/<> lit(0..255)
- l/ln/a/an/o/on/x/xn var16bits ==/>/</>=/<=/<> lit(0..65535)

Il est également possible d'effectuer les opérations suivantes sur ces variables si l'indicateur vaut 1

- =op var8/16 = lit (affectation)
- =op var8/16 + lit (addition pour faire des compteurs par exemple)
- =op var8/16 – lit (soustraction. La soustraction ne descend pas sous 0 donc 2-10 donne 0)

Actuellement, les variables suivantes sont disponibles :

- Les 5 résultats des comparaisons du can
- Les valeurs des 10 temporisateurs qui sont des variables 16 bits qui se décrémentent automatiquement toute les 100ms. Il est ainsi possible de faire des temporisations jusqu'à 6553 secondes soit 1h49 avec une résolution de 100ms.
- La valeur de l'horodateur horo qui indique hh:min et celle de l'horodateur horod qui indique jjjhh:min
- Des variables libres pour l'utilisateur, pour faire des compteurs par exemple …

Actuellement, il n'est plus possible de programmer en littéral, mais les équations permettent de faire la même chose. Le résultat de la compilation du graphcet en graphcet littéral est néanmoins disponible afin de voir le résultat de la compilation.

E.3. Modification du firmware

Si vous souhaitez adapter le firmware à votre application, il est possible de rajouter des bouts de code en assembleur grâce aux sources puis de re-assemblez afin de créer un fichier hex qui sera le nouveau firmware. Par exemple si vous souhaitez adapter l'API à la robotique ludique il faudra rajouter la gestion de servos moteurs et la détection de lignes blanches. Les positions des servos peuvent être contrôlées par des variables. Quand à la détection des lignes blanches, la zone des bits peut refléter la présence ou non de ces lignes. Il faudra néanmoins étudier sérieusement les sources du firmware en assembleur. Vos ajouts ne doivent pas toucher à certaines variables afin de ne pas faire n'importe quoi à l'interpréteur. Ils doivent également être assez rapide pour ne pas gêner l'interprétation. Cependant tout ne peut pas être piloté par un API, les API ne sont pas du tout adaptés pour faire des calculs. C'est pourquoi la programmation classique ne doit pas être oubliée. Il faudra également écrire pour progapi, un nouveau fichier de définition reflétant les caractéristiques du nouveau firmware.

E.4. Flashage du firmware

Pour programmer vos PIC, je vous conseille le programmateur icprog + JDM
Le soft sur pc est icprog et la carte de programmation JDM.
Je recommande ces 2 produits car le soft est simple d'utilisation et d'installation et la carte facile a réaliser et fiable.

icprog

icprog est un freeware créé et maintenu gratuitement par Bonny Gijzen. Il qui permet de programmer facilement entre autre les PICs. Vous pouvez télécharger la dernière version et trouver des informations sur http://www.ic-prog.com.

Pensez à configurer correctement le soft :

- Device = PIC 16F873 (convient aussi pour les autres PIC et programme uniquement les 4 premiers kw de flash ce qui divise par 2 le temps des UC avec 8kw).

- Hardware = JDM (sans inversions)

- Connexion : Indiquez votre port, souvent COM1 ou COM2 et choisissez Windows API pour que le programmateur fonctionne sans problèmes sous Windows 98, 2000 et XP car les registres ne sont pas directement accessibles sous ces OS. Laissez le délai à 10.

- Vérification : Durant la programmation afin de détecter un problème le plus rapidement possible.

Pour programmer un firmware (fichier .hex) :

- Ouvrir le fichier.

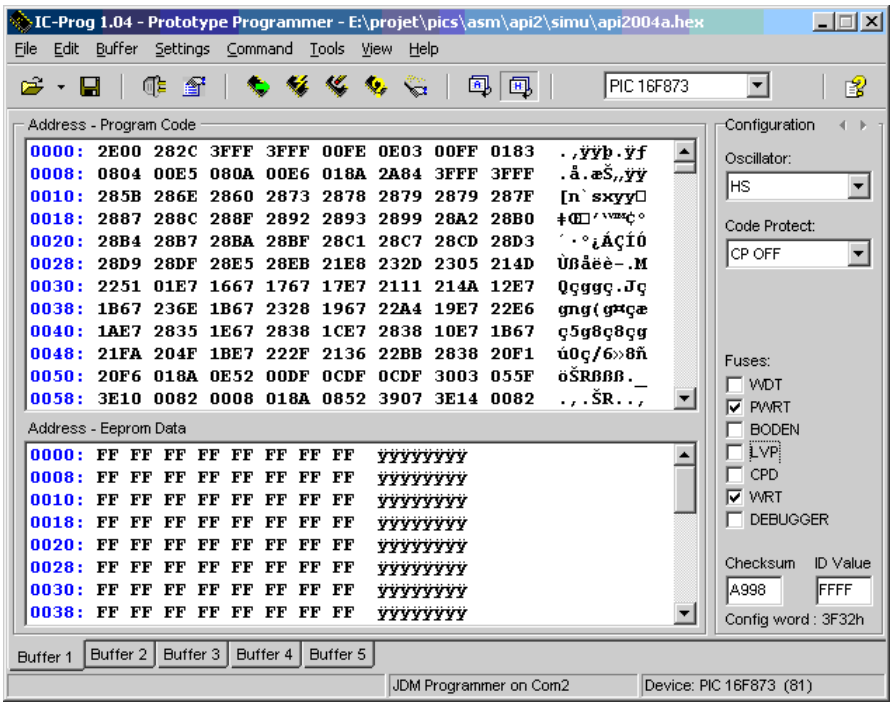
- Sélectionner les options de configuration du PIC comme suit :

- Oscillator : HS pour Hight Speed (grande vitesse). Necessaire pour les quartz > 4MHz

- Code Protect: OFF

- Options à cocher: PWRT pour fiabiliser le démarrage et WRT pour autoriser l'écriture de la flash par le firmware pour télécharger le graphcet.

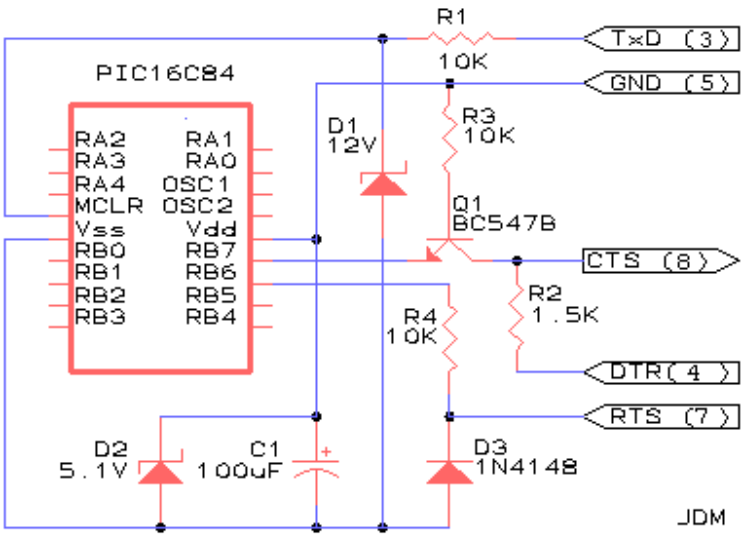
La copie d'écran suivant montre icprog



Le firmware actuel est api2004a.hex. Il convient pour les API 10E/8S ou 15E/14S avec les 4 UC.

JDM

Cette carte de programmation se branche sur un port série et ne connaît pas les problèmes des autres programmateurs sur port série car la masse du microcontrôleur est mise a -5V afin de garantir plus de 12V sur la broche reset (qui permet la programmation).
La figure suivante montre la carte JDM pour un PIC16F84 avec un connecteur série DB9



Pour utiliser un vieux connecteur DB25, changer TXD 3-->2, GND 5-->7, CTS 8-->5, DTR 4-->20, RTS 7-->4.
Pour info, les numéros des broches sont gravés sur les connecteurs. Les ports séries sont des connecteurs DB9 ou DB25 mâles a l'arrière des PC.

Pour programmer les UC qui nous intéressent, il suffit juste d'adapter le support.
Par exemple pour les microcontrôleurs flash a 28 broches (16F873 16F876) ou à 40 broches, adapter le connecteur au brochage suivant (et mettre une résistance de 270Ohms entre RB3 et la masse pour ne pas avoir de problème avec la programmation basse tension).

```
    PIC 18 broches          PIC 28 broches          PIC 40 broches
    PIC 16F84 16F628        PIC 16F873 876          PIC 16F874 877

    RA2 -## ##- RA1     ==>MCLR -## ##- RB7<===  ==>MCLR -#### ####- RB7<===
    RA3 -######- RA0        RA0 -######- RB6<===     RA0 -##########- RB6<===
    RA4 -######- OSC1       RA1 -######- RB5         RA1 -##########- RB5
==>MCLR -######- OSC2       RA2 -######- RB4         RA2 -##########- RB4
===>VSS -######- VDD<===    RA3 -######- RB3<===     RA3 -##########- RB3<===
    RB0 -######- RB7<===    RA4 -######- RB2         RA4 -##########- RB2
    RB1 -######- RB6<===    RA5 -######- RB1         RA5 -##########- RB1
    RB2 -######- RB5        VSS -######- RB0         RE0 -##########- RB0
    RB3 -######- RB4       OSC1 -######- VDD<===     RE1 -##########- VDD<===
                           OSC2 -######- VSS<===     RE2 -##########- VSS<===
                            RC0 -######- RC7         VDD -##########- RD7
                            RC1 -######- RC6         VSS -##########- RD6
VSS=GND                     RC2 -######- RC5        OSC1 -##########- RD5
                            RC3 -######- RC4        OSC2 -##########- RD4
                                                     RC0 -##########- RC7
                                                     RC1 -##########- RC6
                                                     RC2 -##########- RC5
                                                     RC3 -##########- RC4
                                                     RD0 -##########- RD3
                                                     RD1 -##########- RD2
```

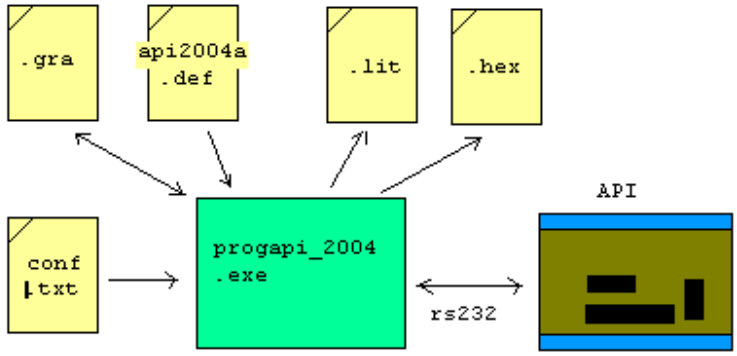
F. Le logiciel coté PC, « progapi »

F. 1. Présentation

Le logiciel coté PC « progapi_2004 » permet :

- d'éditer le graphcet
- de le compiler en graphcet littéral et générer le fichier hex pouvant être programmé par icprog + JDM
- de télécharger directement le graphcet littéral dans la mémoire flash du UC
- d'observer l'API (les étapes actives, la zone des bits, les variables, des chronogrammes …) et le mettre à l'heure.

La figure suivante montre le synoptique de progapi :



L'utilisateur dessine sont graphcet et écrit ses équations dans un fichier texte d'extension .gra

J'ai choisit un fichier texte même pour dessiner le graphcet en ASCII car cela permet de dessiner plus rapidement qu'en mode graphique comme pour les versions précédentes mais aussi de le modifier et l'imprimer avec n'importe quel éditeur.

Progapi permet naturellement d'éditer les fichiers .gra

Le fichier api2004a.def est un fichier de définition qui décrit les API. Il spécifie surtout le nom de chaque bit et de chaque variable. Vous pouvez en créer d'autre et modifier les noms pour faciliter pour augmenter la lisibilité. Il renferme également le fichier api2004a.hex qui contient le firmware avec un graphcet qui ne fait rien.

La compilation du fichier gra crée un fichier lit pour contrôler le résultat et un fichier hex qui peut être programmé dans un UC en utilisant icprog et JDM. Le résultat de la compilation en mémoire de progapi peut également être téléchargé dans le PIC par la liaison série.

Le fichier conf.txt permet de spécifier quelques options de configuration comme les paramètres de communication, le dernier fichier de définition utilisé afin de le recharger automatiquement au prochain lancement…

Progapi permet également d'observer le fonctionnement de l'API via la RS232.

F.2. Règles d'édition d'un graphcet

Comme indiqué précédemment, les graphcets s'éditent dans des fichiers texte avec l'extension .gra. Il est possible de les ouvrir avec progapi ainsi qu'avec n'importe quel éditeur de texte tel notepad ou edit …

Les fichiers gra sont divisés en diverses sections qui commencent par @<nom de la section> et durent jusqu'à la prochaine section ou à la fin du fichier. Les sections actuellement utilisées sont :

- @PRE : (pour prétraitement) pour y écrire les équations à exécuter avant le graphcet

- @GRA : (pour graphcet) pour dessiner le graphcet

- @POST : (pour post traitement) pour écrire des équation à exécuter après le graphcet.

Il est possible de créer d'autres sections qui ne seront pas utilisés par progapi comme @NOTES…

Dans une section, les commentaires sont à insérer entre /* et */ ou entre // et la fin de la ligne du commentaire.

Le listing suivant montre le fichier gra qui nous servira d'exemple (lisez la description).

```
Fichier graphcet progapi 2004

-projet     : exemple
-description: montre l'utilisation des fichiers gra
-version    : 1
-revision   : 0
-historique : 13 nov : création
-notes      :

Une LDR est connectée sur ia0. Lorsque la luminosité devient suffisamment faible (conversion >= 200),
il devient possible de lancer une temporisation d'une minute (600*100ms) en appuyant sur un bouton
poussoir connecté sur ic0. Lorsque la tempo est inactive et peut être utilisée, la sortie ob1 clignote
à 1Hz afin par exemple de signaler l'interrupteur par une led dans la nuit.
La sortie ob2 commande une électrovanne chaque jour entre 18h00 et 20h00 pour arroser la pelouse.
La sortie ob3 active une pompe d'une fontaine décorative tous les dimanches entre 14h00 et 16h00.


@PRE

can0>=200 = bi0

@GRA

  +----+        // voici le graphcet
  |    |
  |  [[0]]      /* etape initiale 0 */
  |    |
  |    - ic0 . bi0
  |    |
  |   [1] t0=600 ob0
  |    |
  |    - t0==0
  |    |
  +----+

@POST

x0 . clisec . bi0 = ob1
horo>=18:00 . horo<20:00 = ob2
horod>=dim14:00 . horod<=dim16:00 = ob3
```

Les graphcet se dessinent avec les caractères + - | [ ]. Il est impératif de mettre les liaisons ou les transitions sous le caractère suivant le dernier [de l'étape précédente. Les étapes portent uniquement un numéro. C'est le seul endroit ou il est autorisé et obligatoire d'utiliser le numéro des bits de la zone des bits. Dans les équations, il faudra utiliser x0 pour l'étape 0

Il est également possible d'utiliser des ancrages et d'éliminer certaines liaisons. Le graphe suivant montre l'utilisation d'un ancrage nommé pos0:

```
    Y pos0
  [[0]]
    - ic0 . bi0
   [1] t0=600 ob0
    - t0==0
    V pos0
```

Afin de manipuler les bits et les variables facilement, ces derniers portent des noms évocateurs au lieu de leurs numéros. Cette correspondance est faite dans le fichier de définition sélectionné.

- Noms des bits

Les noms des bits sont spécifiés dans le fichier de définition. Par exemple api2004a.def définit:

```
x0  .. x79 : pour les étapes
bi0 .. bi95: pour les bits intermédiaire pour stocker des résultats d'équation
irc0, irc2 .. irc5 :  pour les capteurs des barrières infrarouges.
ia0 .. ia3, ia5, ic0, ic2 .. ic5, ie0 .. ie2, id0, id1 : pour les entrées
ob0 .. ob7, od2 .. od7 : pour les sorties
cli10 : clignotement à 10Hz, cli5 : 5Hz, cli3 : 3Hz, cli1 : 1Hz, cli 06 : 0.6Hz, cli03 : 0.3 Hz, cli
015 : 0.15Hz, cli 008 :0.08Hz
cli01 : 0.1 Hz (change d'état toute les 50ms, la période dure donc 100ms)
clisec: 1Hz (la période dure 1 seconde), climin (la période dure 1 minute), clihr (la période dure 1
heure)
eq0..7: Devrait être utilisé dans le futur par le compilateur pour mettre des prioritées dans les
équations. Ne pas utiliser.
Start : bit à 1 uniquement lors de la première interprétation. Il permet certaines initialisations.
horook: indique que l'horodateur est à l'heure
cst0  : bit toujours à « 0 »
cst1  : bit toujours à « 1 »
```

- Noms des variables

Les noms des variables sont spécifiés dans le fichier de définition. Par exemple api2004a.def définit:

```
16 bits  t0..t9       : temporisateurs qui décrémentent de 1 chaque 100 ms
8  bits  can0,1,2,3,5 : résultats des conversions sur le port A
8  bits  v8_0..v8_5   : variables libres pour l'utilisateur
16 bits  v16_0..v16_5 :
      ou cpt0..cpt5   : variables 16 bits libres pour l'utilisateur
8  bits  sec          : secondes 0-59
8  bits  min          : minutes 0-59
8  bits  hour         : heures 0-23
8  bits  dayhour      : jour (1 pour lundi, 7 pour dimanche, 0 pour non initialisé)*32 + heures
16 bits  horod        : horodatage jour_heure_minutes (0-7)*32*256+(0-23)*256+(0-59)
16 bits  horo         : horodatage heure_minutes (0-23)*256+(0-59)
```

Il existe 3 types d'équations :
- Les équations complètes dans PRE ou POST qui sont constituées d'une partie conditionnelles et d'une partie affectation.
- Les équations de transitions qui sont uniquement constituées d'une partie conditionnelle.
- Les équations de sorties qui contiennent uniquement une partie affectation simplifiée.

- Les équations de sorties

Ces équations sont les plus simples, en effet il suffit d'indiquer les noms des bits à activer (souvent des sorties) pour l'étape concernée. Il est également possible de précéder ces noms par set/s et clr/c/clear afin de mettre à 1 ou à 0 le bit concerné à l'étape spécifié. Il ne faut pas mélanger des opérations set/clr sur un bit et des opérations d'affectation car le résultat est alors imprévisible.
Pour les variables, il est possible d'affecter, d'ajouter ou de retrancher une valeur (un littéral). Ces instructions sont uniquement exécutées lorsque l'étape passe d'inactive à active.
Il est possible mais non obligatoire de séparer les instructions par des virgules.
Example : `ob0 ob1 , set ob2, clr ob3, to=1000 cpt0+1 cpt2-3`

- Les équations de transitions

Ces équations contiennent uniquement une partie conditionnelle. Si la condition est vraie alors le changement d'étape opère. Pour les bits, les équation sont logiques : Le et est représenté par un point, le ou par une barre | et le ou exclusif par un chapeau. Il est également possible de travailler sur le complément d'un bit (son état contraire) en utilisant un ! devant son nom. Il est aussi possible de travailler sur l'ancienne valeur du bit avec un § après son nom ou sur un front montant ou descendant en plaçant un / ou \ après son nom. Les équations sont interprétées de la gauche vers la droite sans tenir compte des priorités habituelles qui veulent que le et soit prioritaire. Pour utiliser les priorités, il faudra alors utiliser des bit temporaires connus sous le nom de bit interne.
Pour les variables, il est possible de les comparer ==/>/</>=/<=/<> avec des constantes. Comme les constantes de l'horodateur sont assez difficiles à calculer, il est possible d'utiliser des constantes sous la forme hh :mm pour horo et <lun/mar/mer/jeu/ven/sam/dim>hh :mm pour horod
Bien entendu il peut y avoir des bits et des variables dans la même équation
Exemples :

```
ia0 . ia1 | ia2 ^ ia3
ia0 . ia1 = bi0, ia2 . ia3 = bi1, bi0 | bi1
!ic0 . ic2/ | ic3\ ^ !ic4$
cpt0 == 2
horo >=   18:00 . horo < 20:00
horod>=dim12:00 . horod<dim18:00
can0>200 . ic0
```

- Les équations complètes

Les équations complètes concentrent les 2 équations précédentes. Le résultat de la condition sert à mettre à jour un bit ou une variable.
Pour les bits, il est possible d'utiliser =, =set ou =clr
Pour les variables, il faut d'utiliser =op
Exemples :

```
 horo >= 18:00 . horo < 20:00 = ob2                 // exemple avec l'horodateur
ic3  =op t0=1000, t0<>0 = ob3                        // une tempo toute simple
ic4/ =op cpt0++ , ic5/  =op cpt0--, cpt0>=10 = ob4  // les compteurs
```

- Les équations d'étapes

Pour les allergiques au graphcet, il est également possible de le décrire sous forme d'équations.
* représente une étape initiale, - une étape normale et > un changement d'étape. Si la dernière équation et l'étape courante sont valides.
Exemples :

```
* 0
ic0 . bi0 > 1
* 1  t0==0 > 0
x1 =op t0=600 = ob0
```

Enfin, signalons que l'instruction de fin de graphcet est automatiquement ajoutée.

Après une compilation, vous pourrez examiner et comprendre facilement le fichier de graphcet littéral.

F.3. Utilisation de progapi

Ce chapitre montre comment installer et utiliser progapi pour éditer, compiler, télécharger et observer le graphcet précédent.

Installation

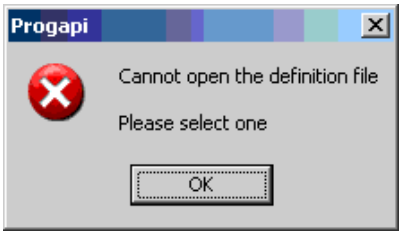
Pour installer progapi, il suffit de :

- créer un répertoire par exemple c:\progapi
- ouvrir l'archive api2004.zip et copier les 2 fichiers progapi_2004.exe et api2004a.def dans le nouveau répertoire.
- Pour plus de facilité d'utilisation vous pouvez faire un raccourci sur progapi_2004.exe et le mettre sur le bureau.
- Vous pouvez aussi copier ce manuel um_api2004a.doc dans le répertoire afin de l'ouvrir directement avec l'aide.

1er lancement

Lancer progapi_2004.exe

Vous êtes sur le magnifique écran vert d'accueil et vous vous voyez gratifier de la belle erreur suivante !



progapi à besoin du fichier de définition des automates qui contient le nom des bits et variables et le code hex du firmware. Ce fichier est nécessaire aussi bien pour compiler que pour afficher correctement les informations lors du dialogue avec l'API.

Appuyer sur OK.

Une fenêtre de sélection s'ouvre et vous permet de sélectionner ce fichier.

Sélectionner le menu File | Exit pour fermer progapi

Si vous regardez dans le répertoire, vous trouverez un nouveau fichier conf.txt qui contient les paramètres de configuration de progapi, comme les paramètres de communication et le dernier fichier de définition ouvert.

Ainsi grâce à ce fichier, progapi chargera automatiquement le fichier de définition sélectionné et vous n'aurez plus l'erreur.

Si vous souhaitez un autre fichier de définition, il sera toujours temps de le sélectionner avec le menu File | Select Definition File.

Vous remarquerez que la langue du logiciel est un mélange d'Anglais et de Français. Ceci est du à mon habitude à programmer en Anglais. Cela reste néanmoins facilement compréhensible par un Français. D'autant que quelques informations difficilement traduisibles en Anglais sont en Français.

Configuration

Relancez progapi. Cette fois vous vous trouver sur l'écran d'accueil sans erreur. Vous serez sur cet écran à chaque lancement et après chaque erreur RS232.

Notez que la version affichée dans la barre des titres est très importante lorsque vous me communiquez des bogues.

Ouvrez le fichier de configuration avec le menu Windows | Edit Config File.

```
RS232_PORT      = COM1
RS232_BOARD_ADR = 3
RS232_USE_SUM   = 1
LIT_PUT_EQ      = 1
DEF_FILE        = C:\progapi\api2004a.def
```

Régler les paramètres de communication. (Pour l'emplacement du fichier de définition, il est plus aisé d'employer le menu File | Select Definition File.)

`RS232_PORT` spécifie le nom du port série sur lequel est conne té l'API (généralement COM1 ou COM2, mais il peut en être tout autrement avec un câble USB-série)

`RS232_BOARD_ADR` est l'adresse de l'API car bmon est adressable. 3 est l'adresse configurée dans le firmware, ne changez donc pas cette valeur

`RS232_USE_SUM` active l'utilisation de checksum dans les communications afin de les sécuriser. La contrepartie est qu'elles sont un peu plus lentes. Pendant le transfère du graphcet le checksum est automatiquement activé afin de détecter la moindre erreur. Pendant le transfère de diagramme, il est automatiquement désactivé afin de disposer de la plus grande bande passante possible. Autrement, c'est une affaire de goût qui n'a que peu d'importance. 1 pour activer, 0 pour désactiver.

`LIT_PUT_EQ :` Lors de la compilation progapi génère un fichier littéral afin d'en vérifier le résultat. Mettre ce paramètre à 1 insère dans le fichier littéral les équations à convertir. Sinon il n'y a que le résultat des conversions.

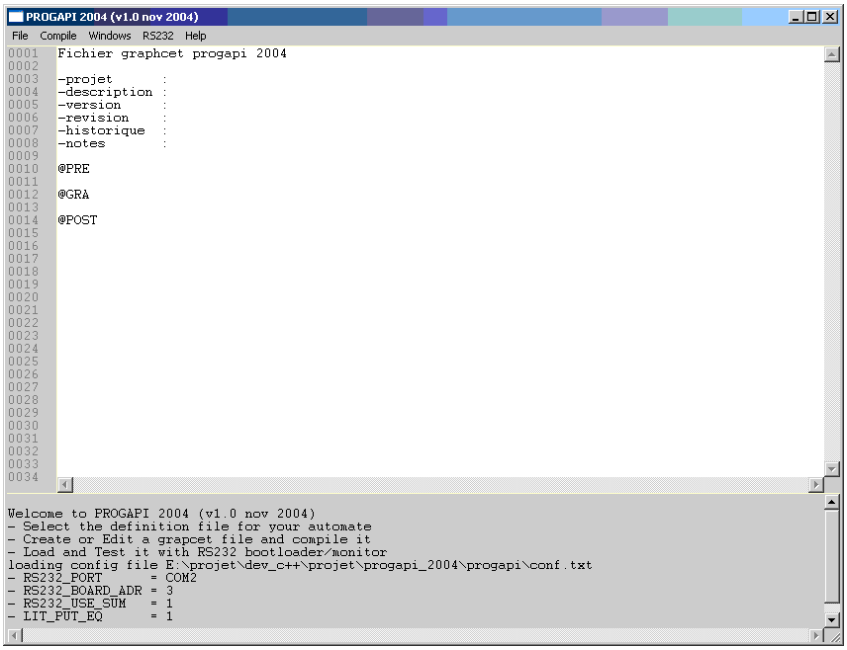
`DEF_FILE` spécifie l'emplacement absolu du fichier de définition a utiliser.

Le fait d'utiliser n'importe quel menu sauve vos modifications.

Saisie du graphcet

Nous allons créer notre premier graphcet avec le menu File | New Graphcet.

La fenêtre d'édition des graphcets s'ouvre avec un fichier texte pré rempli



Cette fenêtre dispose d'une zone d'édition centrale pour saisir votre graphcet et vos équations.
D'une zone à gauche qui identifie le numéro des lignes. Les lignes sont très utiles lorsque le compilateur signale une erreur à talle ligne.
La zone du bas affiche divers messages, de la banale information aux erreurs de compilation, lisez la, avec le plus grand soin.

Vous pouvez revenir à tout moment dans cette fenêtre avec le menu Windows | Edit Graphcet Window

Saisissez l'exemple précédent.
Si vous souhaitez voir fonctionner l'horodateur, changer les constantes d'heures et de dates.

```
Fichier graphcet progapi 2004

-projet     : exemple
-description: montre l'utilisation des fichiers gra
-version    : 1
```

```
-revision   : 0
-historique : 13 nov : création
-notes      :

Une LDR est connectée sur ia0. Lorsque la luminosité devient suffisamment faible (conversion >= 200),
il devient possible de lancer une temporisation d'une minute (600*100ms) en appuyant sur un bouton
poussoir connecté sur ic0. Lorsque la tempo est inactive et peut utilisée, la sortie ob1 clignote à 1Hz
afin par exemple de signaler l'interrupteur par une led dans la nuit.
La sortie ob2 commande une électrovanne chaque jour entre 18h00 et 20h00 pour arroser la pelouse.
La sortie ob3 active une pompe d'une fontaine décorative tous les dimanches entre 14h00 et 16h00.


@PRE

can0>=200 = bi0

@GRA

  +----+        // voici le graphcet
  |    |
  |  [[0]]      /* etape initiale 0 */
  |    |
  |    - ic0 . bi0
  |    |
  |   [1] t0=600 ob0
  |    |
  |    - t0==0
  |    |
  +----+

@POST

x0 . clisec . bi0 = ob1
horo>=18:00 . horo<20:00 = ob2
horod>=dim14:00 . horod<=dim16:00 = ob3
```

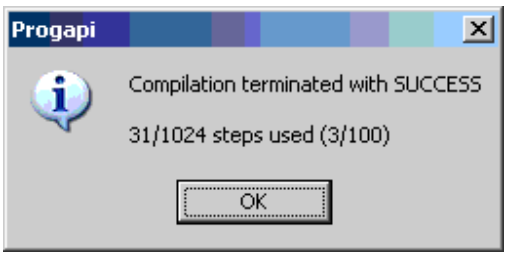
Sauvez votre travail avec le menu File | Save Graphcet As. Choisissez un nom pour votre fichier par exemple exemple.gra
Vous venez de créer votre premier fichier GRA. Le nom du fichier apparaît maintenant dans la barre des titres.
Vous pourrez à partir de maintenant faire File | Save Graphcet pour le sauver.
Il est à noter que le fichier est automatiquement sauvegardé en début de la phase de compilation.
Lors de la prochaine ouverture, vous pourrez récupérer votre fichier avec File | Open Graphcet.

Compilation

Le moment de compiler est enfin arriver. La compilation consiste à transformer le graphcet et les équations en un langage intermédiaire interprétable par l'API appelé graphcet littéral.
Pour compiler, utilisez simplement le menu Compile | Compile
Une fenêtre résume les fichiers mis en jeu :

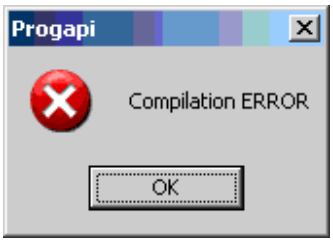
- En entrée du compilateur : Le fichier GRA et celui de définition DEF
- En sortie le fichier littéral LIT et le fichier HEX

Faites OK, la compilation commence pour se terminer avec succès (cette fois du moins)



La quantité de mémoire FLASH utilisée est indiquée.
31pas sur les 1024 disponibles pour le graphcet. Un pas comprend une instruction et un argument soit 2 mots de Flash. Notre graphcet utilise donc 62mots des 2048mots réservés au graphcet. Ce qui fait 3%. Ceci permet de se rendre compte de l'étonnante compacité du graphcet littéral.

Ici, tout finit bien. Nous allons maintenant rajouter une erreur. Pour ce faire transformer la variable can0 par can 0 et recompilez. Vous aurez cette fois une tout autre fenêtre



La zone des messages situe l'erreur, il ne vous reste alors plus qu'à corriger :

```
Line  20: Error in equation: can 0>=200 = bi0
```

Vous pouvez regarder dans le fichier DEF avec Windows | See Definition File que la variable can 0 n'existe pas, contrairement à can0. (De toute façon, il est interdit d'utiliser des blancs dans les noms des variables).
Corriger l'erreur puis recompiler jusqu'à obtenir un beau Compilation SUCCESS.

Examiner maintenant le résultat de la compilation en ouvrant le fichier LIT avec le menu Windows | See Compiled Lit File.

```
@ BEFORE

@ PRE

can0(ZV8,,,20) >(OP,,,62) =(OP,,,61) 200(LIT,,,200) =(OP,,,61) bi0(ZB,,,80)
l    can0>=200            [d4,14][00,c8]
=    bi0                  [04,50]
@ GRA-TRANS

*(OP,,,42) x0(ZB,,,0)
*    x0                   [01,00]

ic0(ZB,,,200) .(OP,,,46) bi0(ZB,,,80)
l    ic0                  [40,c8]
a    bi0                  [41,50]

>(OP,,,62) x1(ZB,,,1)
>    x1                   [03,01]

-(OP,,,45) x1(ZB,,,1)
-    x1                   [02,01]

t0(ZV16,,,0) =(OP,,,61) =(OP,,,61) 0(LIT,,,0)
l    t0==0                [d0,00][00,00]

>(OP,,,62) x0(ZB,,,0)
>    x0                   [03,00]
@ GRA-OUT

t0(ZV16,,,0) =(OP,,,61) 600(LIT,,,600) ob0(ZB,,,208)
l/   x1                   [50,01]
=op  t0=600               [80,00][02,58]
l    x1                   [40,01]
=    ob0                  [04,d0]
@ POST

x0(ZB,,,0) .(OP,,,46) clisec(ZB,,,233) .(OP,,,46) bi0(ZB,,,80) =(OP,,,61) ob1(ZB,,,209)
l    x0                   [40,00]
a    clisec               [41,e9]
a    bi0                  [41,50]
=    ob1                  [04,d1]

horo(ZV16,,,46) >(OP,,,62) =(OP,,,61) 18:00(LIT,,,4608) .(OP,,,46) horo(ZV16,,,46) <(OP,,,60)
20:00(LIT,,,5120) =(OP,,,61) ob2(ZB,,,210)
l    horo>=4608           [d4,2e][12,00]
a    horo<5120            [c9,2e][14,00]
=    ob2                  [04,d2]

horod(ZV16,,,44) >(OP,,,62) =(OP,,,61) dim14:00(LIT,,,60928) .(OP,,,46) horod(ZV16,,,44) <(OP,,,60)
=(OP,,,61) dim16:00(LIT,,,61440) =(OP,,,61) ob3(ZB,,,211)
l    horod>=60928         [d4,2c][ee,00]
a    horod<=61440         [d9,2c][f0,00]
=    ob3                  [04,d3]
@ AFTER
!                         [07,00]
```

Vous voyez ainsi votre graphcet et vos équations compilés en graphcet littéral. Pour voir uniquement le résultat sans les équations sources alors mettez à 0 la variable `LIT_PUT_EQ`. Vous obtiendrez alors uniquement le résultat :

```
@ BEFORE

@ PRE
l    can0>=200            [d4,14][00,c8]
=    bi0                  [04,50]
@ GRA-TRANS
*    x0                   [01,00]
l    ic0                  [40,c8]
a    bi0                  [41,50]
>    x1                   [03,01]
-    x1                   [02,01]
l    t0==0                [d0,00][00,00]
>    x0                   [03,00]
@ GRA-OUT
l/   x1                   [50,01]
=op  t0=600               [80,00][02,58]
l    x1                   [40,01]
=    ob0                  [04,d0]
@ POST
l    x0                   [40,00]
a    clisec               [41,e9]
a    bi0                  [41,50]
=    ob1                  [04,d1]
l    horo>=4608           [d4,2e][12,00]
a    horo<5120            [c9,2e][14,00]
=    ob2                  [04,d2]
l    horod>=60928         [d4,2c][ee,00]
a    horod<=61440         [d9,2c][f0,00]
=    ob3                  [04,d3]
@ AFTER
!                         [07,00]
```

progapi ne se sert pas par la suite de ce fichier, il est juste produit pour vérifier la compilation. Il stocke le résultat de la compilation dans sa mémoire de travail. Ceci explique que si vous fermez progapi, vous devrez recompilez avant de pouvoir télécharger le graphcet dans l'API.

Vous pouvez également visualiser le fichier HEX produit avec le menu Windows | See Hex File. Ce fichier comprend le firmware et le graphcet littéral. Il peut être programmé dans le PIC en utilisant JDM et icprog pour ceux qui ne peuvent pas télécharger par la liaison série. Ne vous inquiétez pas, vous n'avez pas besoin de comprendre ce fichier `-;`

```
:020000040000fa    <-- vecteur de reset
:04000000002e2c287a    <-- vecteur d'it
:08000800fe00030eff0083015e
:0c0010000408e5000a08e6008a01842ac2
:100020005b286e2860287328782879287f280b    <-- interpreteur
:1000300087288c288f28922893289928a228b028ce
:10004000b428b728ba28bf28c128c728cd28d32864
...
:1006c0000530ed0061309f006b2bf2000030ed0033
:1006d00041309f006b2b08001f150800d4308400a8
:1006e0006e088000840a6f088000840a7008800009
:1006f000840a71088000840a72088000840a080055
:100c00000726051a01281526012e1526052ec801ce    <-- bmon
:100c1000cd018316bf3087001930990024309800 29
:100c20008312903098008c1208008c1e5d2f1a08d9
...
:100eb000cc008030cd00431ecd014d080319872f93
:100ec0008316981c872f8312cd1f712f4908990014
:100ed0004030cd004919872f1030cd00c91dcd01fc
:100ee000872f4d1f782f4a0899002030cd00872f7b
:100ef000cd1e812f4b0899001030cd00c91dcd01aa
:100f0000872f4d1e872f4c089900cd01872f831204
:020f10000800d7
:04100000D43414349C    <-- graphcet littéral
:041004000034C834B8
:041008000434503428
:04100C000134003477
...
:04106C00D9342C3413
:04107000F034003424
:041074000434D33439
:041078000734003405
:00000001FF    <-- fin
```

Téléchargement

Il est maintenant temps de télécharger Le graphcet dans l'API. Pour ce faire branchez l'API et connectez le au PC.
Attention, le firmware api2004a.hex doit avoir été chargé préalablement avec icprog et JDM.
Si tout se passe bien vous obtiendrez au bout de quelques secondes la fenêtre suivante :

Mais vous pouvez aussi rencontrer une des erreurs suivantes :

- Impossible d'ouvrir le port série (create file) : Cela indique soit que le port série sélectionné n'existe pas sur votre ordinateur, ou soit qu'il est déjà utilisé par une autre application, car sous Windows, une seule application à la fois peut accéder à un port série spécifique.

- Un nombre bizard s'il y a un problème de communication, soit le retour ne se fait pas, soit le timeout est atteint, soit le checksum n'est pas bon. Il faut alors vérifier la liaison. Essayer le test de rebouclage avec l'hyper terminal précédemment décrit dans les tests de la carte.

- Programmation Error : Pour vérifier la programmation, progapi écrit la Flash puis la lit et compare. Si la lecture est différente de l'écriture, alors vous obtiendrez cette erreur. Si vous reprogrammez la FLASH plus de 1000 fois, des signes de faiblesse peuvent commencer à apparaître. Essayer alors une seconde programmation ou changer de PIC. 1000 écritures est l'endurance minimale de la FLASH, mais avant d'en arriver la …

Vérification manuelle du fonctionnement

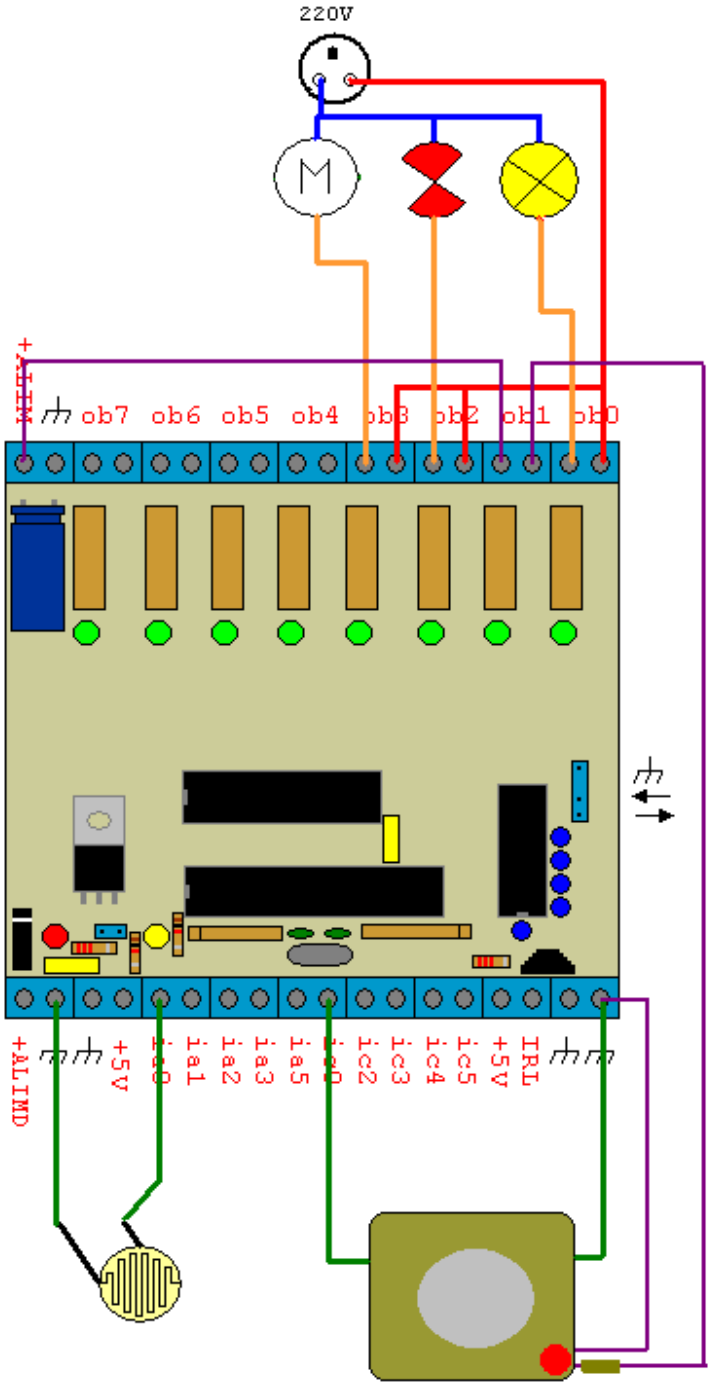
L'exemple téléchargé pourrait servir à automatiser les éléments suivants d'un jardin :

- La lampe extérieure fonctionne avec une temporisation afin d'en limiter le fonctionnement. Lorsque l'utilisateur appuie sur le bouton poussoir la lampe s'allume durant 1 minute. Une LDR autorise cependant l'allumage uniquement lorsque la luminosité du soleil est faible. Lorsque c'est le cas et que le bouton poussoir peut être utilisé pour allumer cette lampe, une led encastrée dans le bouton poussoir clignote. Elle sert également à repérer le bouton poussoir dans la nuit. Lorsque la lampe fonctionne la led s'éteint car elle ne sert plus à rien.

- L'API contrôle également une électrovanne 220V qui permet d'arroser le gazon chaque jour entre 18 et 20h.

- Afin d'agrémenter le jardin il alimente la pompe d'une fontaine décorative chaque dimanche de 14 à 16h.

La figure suivante montre les branchements de l'exemple :



Vous pouvez simplement utiliser un interrupteur et une LDR pour tester l'exemple. Les leds de l'automate indiquerons l'état des sorties. Si vous n'avez pas encore d'interrupteur, utilisez 2 fils avec lesquels vous simulerez l'interrupteurs. Si vous n'avez pas de LDR, ne mettez rien, en effet rien, équivaut à une résistance infinie, ce qui est assez proche d'une LDR dans le noir, vous pourrez donc allumer votre lampe !

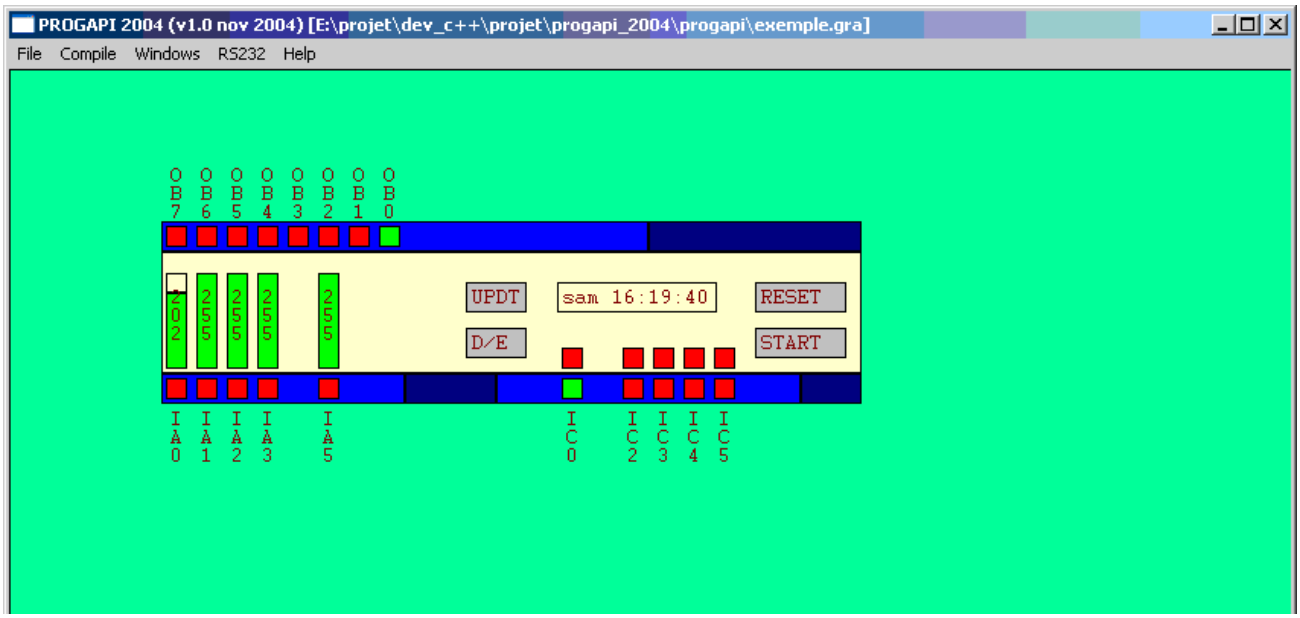
Essayer le fonctionnement de la lampe (virtuelle). Pour l'électrovanne et la fontaine il faudrait attendre la bonne heure et éventuellement le bon jour, mais il ne se passerait quand même rien car l'automate n'est pas à l'heure ! Lorsque l'API n'est pas à l'heure, il indique toujours 0h00 à un jour incorrecte 0 (alors que lundi=1 et dimanche=7)

Observation des entrées/sorties et mise à l'heure de l'horodateur

Ouvrer à présent l'écran des entrées/sorties avec le menu RS232 | Inputs/Outputs

Une représentation de l'automate s'affiche alors à l'écran. Vous pouvez voire l'état des entrées et des sorties, l'état des capteurs infrarouges (au dessus des entrées ic0,2,3,4,5), les résultats des conversion du CAN et le jour et l'heure courant de l'API qui devrait être à xxx xx :xx :xx

La figure suivante montre cet écran :



On voit sur l'exemple que la lampe (ob0) est allumée. Le bouton poussoir connecté à ic0 est appuyé et la lampe à pu être allumée car il fait assez sombre (la valeur de can est supérieure à 200). Il est 16h19 donc l'arrosage et la fontaine ne fonctionnent pas. La led de repérage de l'interrupteur ne clignote pas car la lampe est allumée. Les ports D et E normalement dans les rectangles bleu foncés sont cachés.

L'indicateur IR d'une entrée du port C n'a de sens que si un capteur IR est utilisé.
De même, si un capteur IR est utilisé, l'indicateur d'entrée normal n'a pas de sens.

Lorsque les can passent sous 128 (2.5V), les indicateurs d'entrées doivent s'activer.

Les 4 boutons gris permettent de réaliser les opérations suivantes :

- Le bouton UPDT (pour Update) permet de mettre l'API au jour et à l'heure du PC. Une fois que l'API est à l'heure, il devrait le rester jusqu'à la prochaine coupure d'alimentation.

- Le bouton D/E permet d'afficher ou de cacher les ports D/E disponibles uniquement sur l'automate api2004_29a. Sur un automate api2004_18a, les ports D et E n'existent pas et l'état des indicateurs est bidon.

- Le bouton Start/Stop, permet de démarrer ou stopper l'automate. Ceci peut être utile pour la mise au point afin d'examiner l'API à un moment particulier. Lors de la mise sous tension, l'API démarre automatiquement.

- Le bouton Reset permet de réinitialiser le graphcet. Cela peut également servir lors de la mise au point. Si l'automate était à l'arrêt, alors il redémarre.

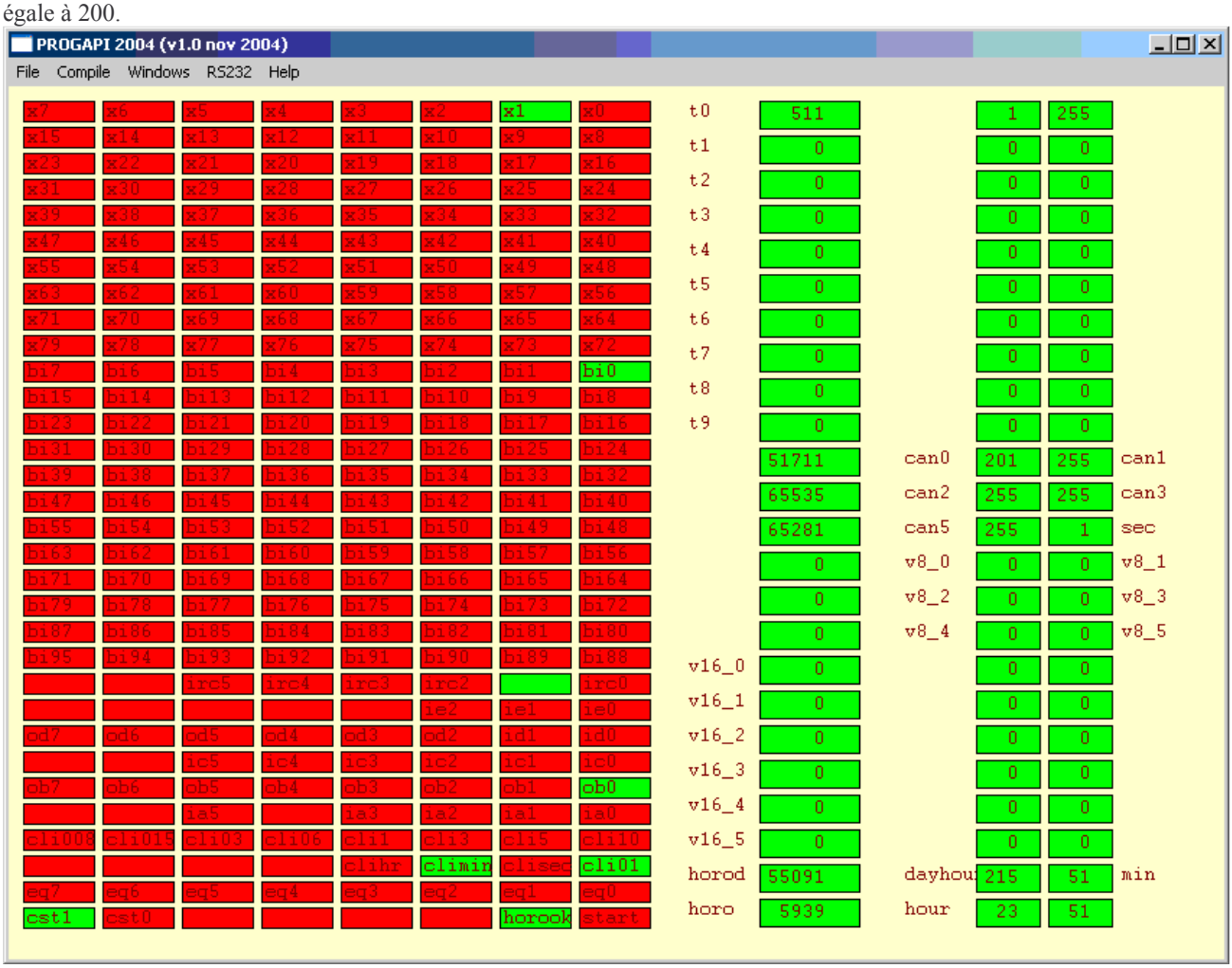
Vous pouvez jouer avec les entrées afin de voir changer les indicateurs. Les 4 CAN non utilisés devraient être à 255 car ils sont tirés au +5V par les résistances de pull-up. Plus il fait sombre et plus la valeur du can0 doit augmenter.

Observation des bits et variables

Accédez maintenant à l'écran des bits et des variables avec le menu RS232 | See ZB/ZV

Vous voyez les 256 bits de la zone des bits courante ainsi que toutes les variables. Le rafraîchissement de l'ordre de 2secondes est assez lent et ne permet pas de voir les bits de clignotement clignoter à vitesse réelle. Cette lenteur est du à la grande quantité de donnée à transférer.

Vous pouvez par exemple regarder les étapes x0 et x1. Ou encore le bit bi0 qui est mis à 1 lorsque can0 est supérieure ou égale à 200.



La zone des variables est constituée de 48 octets. Dans ces 48 octets cohabitent des variables 16bits qui occupent donc 2 octets et des variables 8 bits qui en occupent un seul. Il est également possible d'utiliser un des 2 octets d'une variable 16 bits comme pour la variable 16 bits horo qui est constitué d'une variable 8 bits heure et une variable 8 bits min. Pour cette raison, l'écran affiche en décimal les valeurs 16 bits et 8bits. Notez que 5939=256*23+51.

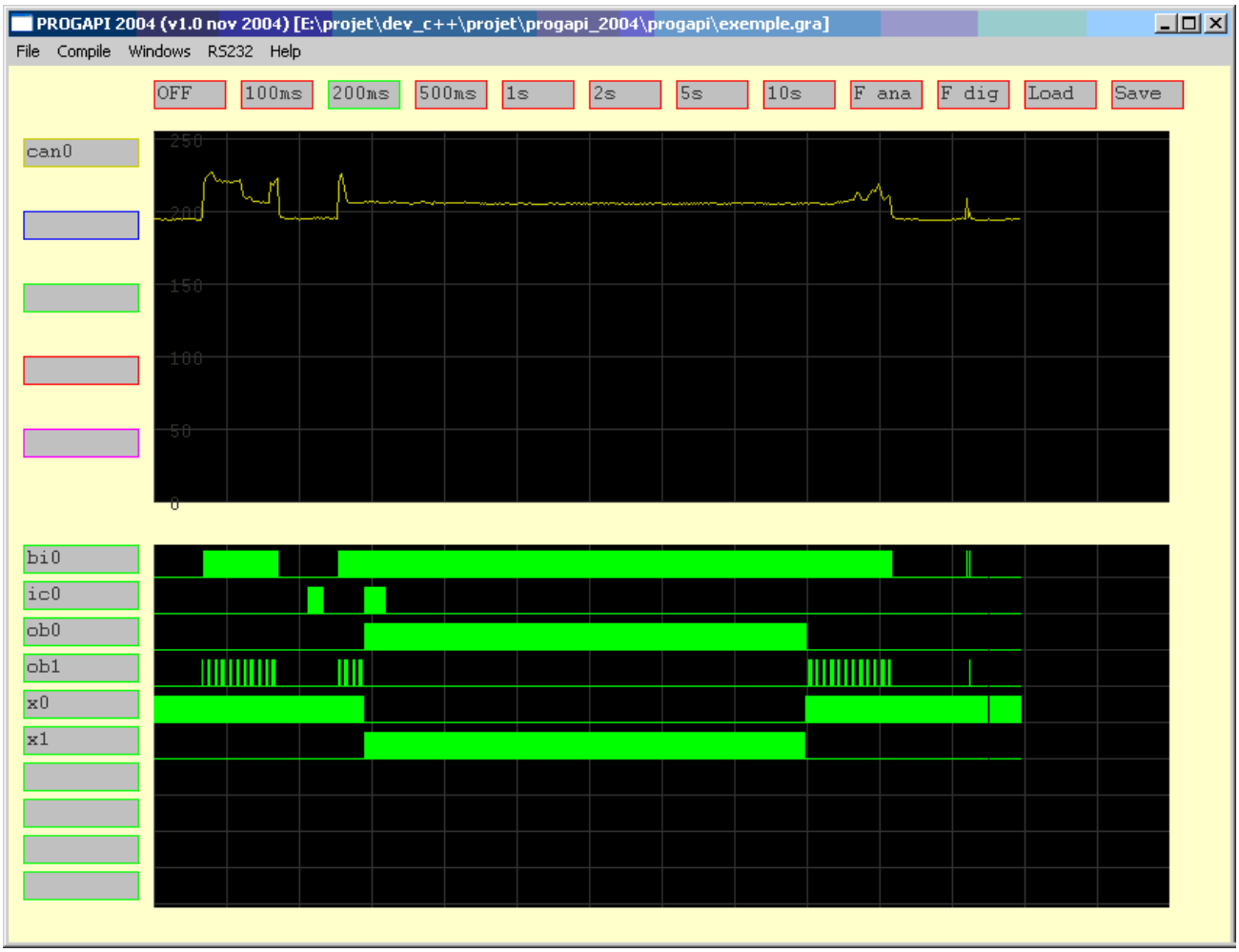
Vous remarquerez peut-être que les bits non utilisés 7,6 et 1 des capteurs infrarouges clignotent. Ceci est normal, car rappelez vous que le firmware mémorise à un moment particulier l'état du port C (En fin d'émission du signal IR). A ce moment, le bit 1 qui est la sortie 30kHz modulée peut très bien être à 1 ou a 0. Idem pour les bits 6 et 7 qui représente l'état des pattes dédiées à la communication RS232.

Pour l'automate 10E/8S, les bits des ports D et E ne sont pas à prendre en compte.

Génération de chronogrammes

Progapi permet également de tracer les chronogrammes des bits et des variables. Pour ce faire, rendez vous dans l'écran des chronogrammes avec le menu RS232 | See Graphics.

La figure suivante montre cet écran



La zone du haut représente l'évolution des variables. Ici seul le can0 a été sélectionné.
La zone du bas représente l'évolution de l'état des bits. Ici l'entrée ic0, le bit interne bi0, les sorties ob0 et ob1 et les étapes x0 et x1 ont été sélectionnées.

Il est extrêmement facile de vérifier le comportement. On voit par exemple que le bit interne bi0 passe à 1 lorsque can0 >=200. Lorsque c'est le cas la led de la sortie ob1 clignote et il est possible d'allumer la lampe ob0 avec l'interrupteur connecté sur ic0. Le chronogramme montre que la première tentative échoue car il ne faisait pas assez sombre. Par contre la seconde déclenche bien la temporisation durant 1 minute. Durant ce temps, la lampe s'allume et la led arrête de clignoter.

Le fonctionnement est le suivant :
1 – Entrez le nom des variables et bits dans les petites cases de gauche en appuyant sur Enter après chaque saisie
2 – Choisissez la période d'échantillonnage. (A ce moment, les mesures commencent).

Si la variable ou le bit n'existe pas, alors la case reste dans la couleur de saisie, sinon, elle retrouve son gris habituel.
La période d'échantillonnage représente l'intervalle de temps entre 2 mesures pour chaque bit et variable et correspond à une avancé d'un pixel sur l'écran. A raison de 50 pixels par carreau, un écran complet représente 700 mesures pour chaque bit et variable. S'il n'est pas possible de tenir le rythme spécifié, alors, le chronogramme est temporellement faux. Cela se produit souvent à 100ms s'il y a trop de variables et bits. Dans l'exemple, on voit que le chronogramme de 200ms est juste car la temporisation dure 6 carreaux soit 6*50*200ms=60s=1minute. En fait un petit peu plus car les temporisateurs utilisent en fait une base légèrement plus grande que 100ms.

Les boutons F_ana et F_dig pour Fast Analogique et Fast Digital lancent une acquisition de la première variable ou du 1er bit à la vitesse maximale. Cela peut être intéressant pour voire des phénomènes rapides.

Il est également possible de sauver la configuration dans un fichier avec le bouton Save et de la récupérer par la suite avec le bouton Load.

Observation des étapes du graphcet.

L'écran ZB/ZV permet déjà d'observer les étapes, mais il y a encore plus simple pour voir l'évolution de votre graphcet. Pour cela lancez l'écran d'observation des étapes actives du graphcet avec le menu RS232 | See Active Graphcet.

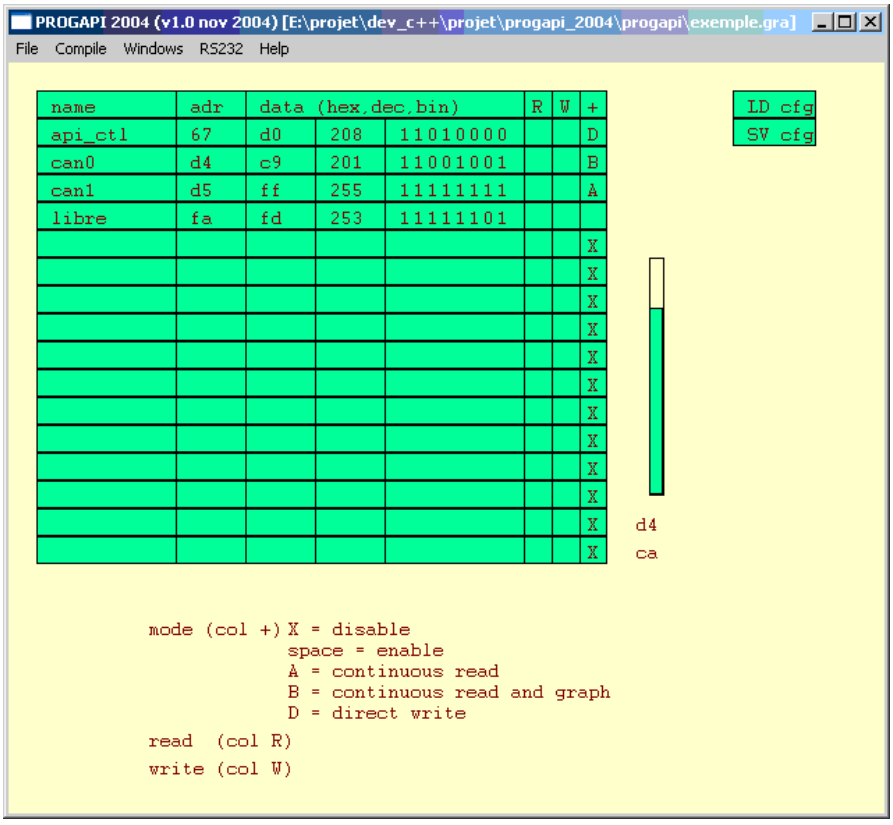
Vous verrez alors votre fichier GRA avec des étoiles qui, clignotent à la place des numéros des étapes actives.

Ceci est très utile pour les gros graphcets.

Utilisation du moniteur

Enfin le moniteur RS232 accessible par RS232 | Monitor permet de voir et modifier la RAM du PIC. Il ne tire pas toute la puissance de bmon qui permet aussi d'explorer la FLASH, l'EEPROM et de travailler en 16bits, mais il est largement suffisant pour déboguer le firmware. Il devrait donc juste intéresser ceux qui se lanceront dans la modification du firmware. Les autres peuvent sauter sa description.

L'écran suivant montre ce moniteur



Il est possible d'utiliser 16 emplacements RAM.

Nous allons en premier lieu, observer et modifier l'emplacement libre à l'adresse 0xfa (en hexa). Commencez par activer une ligne en cliquant dans la colonne + et en tapant sur la barre espace. Entrez ensuite l'adresse. Un appuie dans la colonne R (Read=lecture) lit la valeur de la RAM à cette adresse. Modifier ensuite la valeur en la rentrant en décimal ou en hexa ou cliquez sur les bits pour les faire changer d'état. Une fois que la valeur est rentée, cliquer dans la colonne W (Write=écriture).

Examinons maintenant une cellule RAM utilisée. Par exemple la variable can1. Le fichier de définition indique que can1 est la 21eme variable. Les variables sont stockées à partir de l'adresse 0xc0. Il faut donc rajouter cet offset d'où l'adresse 0xd5. Une lecture avec R donne la valeur attendu de 255 car l'entrée est non utilisée et sa pull-up la tire au +5V. Tapez maintenant A (pour Automatic) dans la colonne +. A ce moment, cette cellule est lue en permanence. Ca ne se voit pas ici car elle vaut toujours 255.

Faites de même pour le can0, mais tapez B (pour Barre). Comme dans le cas précédent, la lecture est continue, mais en plus un barre graphe permet de suivre les évolutions la cellule mémoire.

Faites pareil pour l'adresse 0x67 qui contrôle l'activation de l'API. Faites R pour récupérer la valeur actuelle puis tapez D (pour Direct). A ce stade, toute modification de la cellule sur l'interpréteur est automatiquement reproduite en RAM. Par exemple si vous cliquez sur le bit 1 isolé, il passe à 0, ce qui provoque l'arrêt de l'automate (Cela est identique au bouton Stop de l'écran des E/S).

La colonne name vous autorise à donner un nom à cette cellule RAM (ou registre). Afin de vous y retrouver plus facilement.

Vous pouvez ici aussi sauvez votre configuration pour la recharger plus tard.

Aides

Le menu Help procure les aides suivantes :

Help | About : Indique la version de progapi, la date de sa compilation, mon email et l'adresse du site web.

Help | Help : Affiche une petite aide

Help | Personal Help : Affiche votre aide personnelle que vous pouvez écrire dans le un fichier texte appelé pershelp.txt dans le répertoire de progapi. Ce fichier est à créé.

Help | User Manual : Affiche ce manuel s'il est dans le répertoire de progapi sous le nom um_api2004a.doc et si Word ou un liseur de .doc est installé sur votre PC.

F.4. Notes

Ce chapitre donne quelques informations complémentaires, astuces …

Sorties conditionnelles

Le chapitre de la norme introduisait les sorties conditionnelles. La led clignotante de l'exemple précédent en est une parfaite démonstration. Au lieu de mettre la sortie directement dans les actions de l'étape, il suffit de mettre un bit interne à la place ou rien du tout si la sortie est seulement active durant cette étape. Après le traitement du graphcet, dans POST il suffit de faire un ET entre l'étape (ou le bit interne) et la condition, puis d'affecter la sortie. Par exemple l'équation suivante permet de faire clignoter la sortie ob2 à 1Hz lorsque l'étape 3 est active.

```
X3 . clisec = ob2
```

Réutilisation des temporisateurs

Les temporisateurs peuvent être utilisés à plusieurs endroits du graphcet à condition que chaque temporisateur ne puisse pas être utilisé au même moment par différents bouts du graphcet.

Il est ainsi possible de réaliser un séquenceur avec un simple temporisateur. Le graphcet suivant permet de faire un chenillard sur les 4 premières sorties. Elles s'activent toutes les 10 seconds.

```
  Y loop
[[0]] ob0 t0=100
  -   t0==0
 [1]  ob1 t0=100
  -   t0==0
 [2]  ob2 t0=100
  -   t0==0
 [3]  ob3 t0=100
  -   t0==0
  V loop
```

Il est ainsi possible de réaliser des animations lumineuses très simplement comme le K2000 suivant :

```
  Y loop
[[0]] ob0 t0=5
  -   t0==0
 [1]  ob1 t0=5
  -   t0==0
 [2]  ob2 t0=5
  -   t0==0
 [3]  ob3 t0=5
  -   t0==0
 [4]  ob2 t0=5
  -   t0==0
 [5]  ob1 t0=5
  -   t0==0
  V loop
```

Télérupteurs

Les télérupteurs sont nécessaires dès lors que l'on veut commander une lampe à partir de plus de 2 interrupteurs. Le principe est simple. A chaque appuie sur un des boutons poussoir, un relais change d'état et allume ou éteint l'ampoule qui lui est connectée. Les interrupteurs sont du type bouton poussoir. L'équation suivante suffit à faire un télérupteur :

```
ic0/ ^ ob0 = ob0
```

S'il y a un front sur l'entrée ic0 alors la sortie change d'état. Il est possible de branchez tous les interrupteurs sur une seule entrée. En effet en appuyant sur un des boutons poussoir, l'entrée se retrouve à la masse et devient active.

La lame d'un bouton poussoir, ne se ferme pas et ne s'ouvre pas immédiatement, mais elle rebondit. Ceci génère plusieurs fronts. Cependant l'automate scrute ces entées toutes les 50ms et ne devrait donc pas être gêné par ce phénomène plus rapide.

Temporisateur réarmable

Un temporisateur réarmable est un temporisateur qui se déclenche lorsque l'on appuie sur un bouton pour une durée déterminée. Mais à chaque appuie, il réinitialise l'attente. Ce fonctionnement est bien entendu différent d'un temporisateur non réarmable comme l'exemple du jardin ou un appuie sur ic0 durant la temporisation ne change rien. La toute simple équation suivante réalise un temporisateur réarmable:

```
ic0 =op t0=600, t0<>0 = ob0
```

Temporisateur non réarmable

Il suffit simplement de vérifier que la temporisation n'est pas en cours avant de la relancer.

```
t0==0 . ic0 =op t0=600, t0<>0 = ob0
```

Temporisateur de très longue durée

Mes temporisateurs standard se décrémente toutes les 100ms et sont des variables 16 bits. La durée maximale est donc de 65535*100ms=6553s soit environ 1h49.
Pour avoir de plus gros temporisateurs, il suffit d'utiliser une variable libre (par exemple de 16 bits) et de la décrémenter plus lentement à l'aide d'un indicateur de clignotement.
Cela fonctionne également avec les variables 8 bits.
Vous pouvez utilisez clisec, climin ou clihr, mais attention clihr ne fonctionne pas lorsque l'horodateur n'est pas à l'heure, je conseille donc de pas l'utiliser. En décrémentant une variable 16 bits toutes les minutes, il est donc possible de temporiser durant 65535/60=1092h=~45jours. L'exemple suivant montre une temporisation réarmable de 10h

```
ic0 =op v16_0=600, v16_0<>0 = ob0
climin/ =op v16_0 --
```

Tables de vérités des fonctions logiques

Tout le monde n'est pas électronicien et les opération logiques et, ou , ou exclusif ne sont pas naturelles pour tous le monde. Les tables de vérité suivantes donnent le résultat d'une opération logique à 2 entrées.

```
ET

a . b = c    (a et b égal c)

0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1 . 1 = 1

sortie c à 1 si les 2 entrées a et b sont à 1

OU

a | b = c    (a ou b égal c)

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

sortie c à 1 si au moins une des 2 entrées est à 1
Ce OU est aussi connu sous le nom de OU inclusif


OU EXCLUSIF

a ^ b = c    (a ou b égal c)

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

sortie c à 1 si une seule des 2 entrées est à 1
Il peut également servir à inverser un bit. Par exemple si b=0 alors c=a, sinon c=!a (complément de a)
Il peut également servir à comparer 2 bits. Par exemple si les bits sont identiques alors la sortie
vaut 0.
```

Divergences/Convergences en ET et OU.

L'exemple présentait un graphcet linéaire, mais comme présenté au chapitre norme du graphcet, il est possible de faire des divergences en ET et en OU. L'exemple suivant montre un OU à gauche et un ET à droite.

L'exemple en OU permet de lancer une tempo de 1 minutes sur la sortie ob0 lorsque l'on appuie sur ic0, tandis que l'appuie sur ic2 lance une tempo de 2 minutes sur la même sortie. Notez que la transition est sélective.

L'exemple ET lance une temporisation de 1 minute sur la sortie ob1 et une de 2 minutes sur ob2, lorsque l'on appui sur ic3. Avant de relancer une nouvelle séquence, les 2 temporisations doivent être terminées. Notez la synchronisation de fin utilise l'ancien état des étapes. Ceci est nécessaire pour travailler sur une base qui ne bouge pas. En effet lors de l'examen de l'étape 6, si 6 et 7 sont actives alors 3 s'active et 6 se désactive. Puis lors de l'examen de l'étape 7, 6 étant désactivée, 7 reste active ! Alors qu'en travaillant sur l'ancien état qui ne change pas en cours d'interprétation (cf ancienne zone des bits), tout se passe pour le mieux.

```
/* OU */

+-----+
|     |
|   [[0]]
|     |
|     +----------------+
|     |                |
|     - ic0            - ic2 . !ic0
|     |                |
|    [1] ob0, t0=600  [2] ob0, t0=1200
|     |                |
|     - t0==0          - t0==0
|     |                |
+-----+----------------+
```

```
/* ET */

+------+
|      |
|    [[3]]
|      |
|      - ic3
|      |
|      +-----------------+
|      |                 |
|     [4] ob1, t1=600   [5] ob2, t2=1200
|      |                 |
|      - t1==0           - t2==0
|      |                 |
|     [6]               [7]
|      |                 |
|      +-----------------+
|      |
|      - x6§ . x7§
|      |
+------+
```

Le portail.

L'exemple présenté était un petit peu simple mais présentait la plupart des fonctionnalités. Voici à titre d'exemple la gestion d'un portail qui s'ouvre lorsque l'on appuie sur une télécommande HF dont le récepteur est connecté sur ic0. Le portail s'ouvre, reste ouvert 1 minute, puis se referme. Lors de la fermeture, 2 barrières infrarouges connectées sur irc2 et irc3 et placées de part et d'autre du portail arrêtent sa fermeture lorsqu'un des rayons est coupé par un obstacle et commandent sa réouverture. Un capteur à galet connecté sur ic4 indique que le portail est fermé et un autre sur ic5 qu'il est ouvert. La sortie ob0 commande la fermeture et ob1 l'ouverture.

```
    [[0]]
      +----------+
      - ic4      - !ic4
+-----+          V ferme
|    [1]
|     |
|     - ic0     Y ouvre
|     +---------+
|    [2] ob1
|     |
|     - ic5
|     |
|    [3] t0=600
|     |
|     - t0==0   Y ferme
|     +----------+
|    [4] ob0
|     +----------------------+
|     - ic4 . irc2 .irc3     - !irc2 | !irc3
|     |                      V ouvre
+-----+
```

Je vous laisse le soin de comprendre le graphcet. Vous pouvez bien entendu rajouter de la fioriture, comme un flash qui clignote lorsque le portail bouge ou encore une lampe qui s'allume lorsque le portail est ouvert et qu'il fait nuit …

Horodateur

En fait, il y a 2 horodateurs,

- Le premier appelé horod (d pour day (jour)) donne le jour de la semaine et l'heure courante (heure + minute). Il est bien adapté pour réaliser des actions hebdomadaires.

- Le second appelé horo est en fait une copie du premier mais sans le jour. Il contient uniquement l'heure (heure + minute). Il est bien adapté pour réaliser des tâches quotidiennes qui ne dépendent pas du jour de la semaine.

Il ne faut donc pas se tromper en les employant.

La valeur de ces variables 16 bits est la suivante :

32*256*Jour + 256*heure + minute.

Jour est bien entendu uniquement utilisé par horod. Lundi est codé 1, mardi 2, … et dimanche 7. 0 indique que l'horodateur n'est pas à l'heure.

Vous pouvez utiliser cette formule, mais il est bien plus agréable d'utiliser des constantes sous la forme `jjjHH:MM` ou `HH:MM` que progapi convertit automatiquement.

Ex : `dim16:00` ou `23:55`

Pour utiliser l'horodateur, il suffit simplement d'utiliser les comparaisons.
La comparaison `horo==08:00 = ob0` active la sortie ob0 durant 1 minute entre `08:00:00 et 08:00:59`).

Pour utiliser une plage horaire, il suffit de faire 2 comparaisons
`horo>=08:00 . horo<=09:35 = ob1` active la sortie ob1 entre `08:00:00 et 09:35:59`).

Il convient d'utiliser le OU à la place du ET lorsque la plage horaire traverse minuit pour horo ou dimanche minuit, pour horod. Je vous laisse réfléchir quelques minutes à ce phénomène …

```
Horo>=23:00 | horo<=01:00 = ob2
Horod>=dim23:00 | horod<=lun01:00 = ob3
```

Lorsque l'automate n'est pas à l'heure (après la mise sous tension par exemple, les horodateurs indiquent 0). Le jour à 0 d'horod indique que l'horodateur n'est pas à l'heure. Le firmware se sert de cette information pour mettre à jour le bit horook. Lorsqu'il est à 1, cela indique que l'API est à l'heure.

Si vous utilisez un automate qui n'est pas à l'heure, le résultat de comparaisons devrait être faux.
Comme par exemple entre 12 et 15h : `horo>=12:00 . horo<=15:00 = ob0`

Sauf si

- vous faites une comparaison sur horo avec minuit : `horo==00:00 = ob0`

- vous tester une plage horaire avec horo qui traverse minuit : `horo>=23:00 | horo<=01:00 = ob0`

- vous testez une plage avec horod qui traverse dimanche minuit : `horod >=dim23:00 | horod<.lun01:00`

Dans ces cas, il est conseillé de faire un ET avec le bit horook

```
horo==00:00 . horook = ob0
horo>=23:00 | horo<=01:00 . horook = ob1
horod>=dim23:00 | horod<=lun01:00 . horook = ob2
```

Si vous n'utilisez pas la RS232 et désirez vous servir de l'horodateur, vous pouvez utiliser une entré pour mettre l'automate à une heure et jour spécifié. L'exemple suivant met l'automate à lundi 12:00 lorsque l'on appuie sur un bouton poussoir relié à ic0. Il faut mettre à jour horod et non horo car ce dernier est une simple copie d'horod. horod est l'horodateur maître.

```
ico =op horod=lun12:00
```

G. Conclusion

J'espère que cette documentation aura été agréable à lire et vous aura apprit de nouvelles choses. Vous savez maintenant tout sur ces API 2004. Ce sont des outils très puissants pour tout ce qui touche à l'automatisation et si vous les essayez, vous ne pourrez vraisemblablement plus vous en passer ! J'ai essayé de les rendre les plus simples et les moins chers possibles, afin que tout le monde puisse les réaliser et les utiliser.

Vous pouvez me contacter pour toute question (mais lisez bien, la documentation avant). Si vous décidez de partager vos réalisations, envoyez moi un lien. Je serais également très content de recevoir des photos de vos réalisations.

Ce projet a été développé uniquement avec des produits gratuits :

- mplab pour le firmware (éditeur, assembleur et simulateur)
- icprog et jdm pour les premières programmations des PIC
- dev c++ 4 pour le logiciel cote PC (éditeur et compilateur)
- tci pour le circuit imprime

Il est donc tout naturellement livré dans l'esprit GNU (gratuitement et avec les sources). Il ne peut cependant être utilisé commercialement sans mon consentement.

Et pour terminer, la célèbre phrase de couverture : Je ne peux être tenu pour responsable des problèmes ou dommages causés par l'utilisation de mes API ou liés aux informations de cette documentation. Evitez néanmoins d'utiliser ces automates dans des situations pouvant mettre des vies en jeux.

BONNES REALISATIONS !

Ulysse Delmas-Begue novembre 2004
Email : udelmas@chez.com / ulysse.delmas@laposte.net
Site web: http://www.chez.com/udelmas