AIDE-MÉMOIRE INSTRUCTIONS

PIC & PIC18

SIMULATOR IDE

de Oshon Software

SOMMAIRE

Conception de l'aide-mémoire

Cet aide-mémoire portant sur les simulateur/compilateur basic de Oshon Software, vise à obtenir une traduction de l'essentiel du support on line du site web.

Les éléments usuels sont triés par thèmes, et pour chacune des fonctions, est stipulée, la compatibilité avec les compilateurs (PIC et/ou PIC18). Attention toutefois, car excepté pour les fonctions ayant un rapport direct avec des tensions analogiques (tel le CAN), les pattes doivent être définies en digital.

Cet aide-mémoire possède, à la fin, des exemples de code, servant à différentes fonctions, avec éventuellement l'électronique à associée.

Si vous voyez des erreurs dans ce tutorial, et/ou des amélioration, ou encore si vous avez du code à proposez pour une petite fonction, n'hésitez pas à nous envoyer un mail.

Bonne lecture.

Communication

A - 1 fil (PIC&PIC18)

B - Série software (PIC&PIC18)

C - Série Hardware (PIC&PIC18)

D – SPI (PIC&PIC18)

E – I²C (PIC&PIC18)

F – USB (PIC18) (en cours d'écriture)

A - 1 fil

INSTRUCTIONS:

- 1WIRE_REG
- 1WIRE_BIT
- 1WIREINIT
- 1WIRESENDBIT
- 1WIREGETBIT
- 1WIRESENBYTE
- 1WIREGETBYTE

EXPLICATIONS:

Comme vous l'aurez deviné, les 4 dernières instructions servent à recevoir et émettre un bit, ou un octet. Nous ne nous attarderons alors que sur les 3 premiers mots clé. Sachez tout de même, qu'il est possible d'envoyer et recevoir plusieurs octet à suivre. Il suffit d'utiliser une virgule en séparation.

1WIRE_REG et 1WIRE_BIT servent à préciser quelle patte est utilisé pour la communication (reg=port, bit=patte). La configuration est initialisée avec l'instruction 1WIREINIT.

EXEMPLE:

```
1WIRE_REG = PORTA
1WIRE_BIT = 1
1WIREINIT

DIM i as BYTE

main:
    1WIRESENDBYTE 0xCC, 0xFF
    WAITMS 1
    1WIREGETBYTE i
goto main
end
```

B - Série software

INSTRUCTIONS:

- SERIN
- SERININV
- SEROUT
- SEROUTINV
- SEROUT_DELAYUS

EXPLICATIONS:

SERIN sert à recevoir des données.
SEROUT sert à recevoir des données.
Dans les deux cas, le niveau logique peut être inversé (SERININV & SEROUTINV).

SEROUT_DELAYUS sert à configurer la tempo de départ. Par défaut, cette dernière vaut 1000, et suffit dans énormément de cas, vous n'avez donc pas à vous en souciez spécialement.

Le CARRIAGE RETURN et LINE FEED sont programmé en mémoire. Vous pourvez les appelés en écrivant CR, LF, ou CRLF. Dans le cas, ou vous avez des problème de transmissions (par exemple, vous ne recevez pas ce que vous envoyer), vous pouvez agir sur OSCCAL, qui est le registre permettant d'afiner la fréquence d'oscillation de l'oscillateur interne. Par défaut à 128 (valeur max), vous pouvez le rabaisser par palier de 20 pour améliorer la transmission.

EXEMPLE:

```
DIM I AS BYTE
OSCCAL = 88
loop:
    SERIN PORTC.7, 9600, I
    SEROUT PORTC.6, 9600, "Number: ", #I, CrLf
GOTO loop
END
```

Tout comme le 1 fil, plusieurs donnée peuvent être envoyées, séparées d'une virgule et les phrases sont entre guillemets. On remarquera le # devant la variable I dans la ligne d'envoi. Cette configuration permet d'envoyer la valeur décimale.

Attention, car selon la fréquence du quartz utilisé (ou oscillateur interne), les hauts débits engendre plus d'erreurs que les bas débits.

C - Série Hardware

INSTRUCTIONS:

- HSERGET
- HSERIN
- HSEROUT
- HSEROPEN

EXPLICATIONS:

La première instruction à utilisée est HSEROPEN. Ce mot permet d'activer le module série du PIC, ainsi que de définir le débit de communication. En l'absence de débit, la valeur par défaut sera de 9600. HSERGET, lui permet de récupéré une éventuelle donnée restée dans le buffer série, dans le cas éventuel ou ce dernier ne serait pas plein. En effet, HSERIN ne lit le buffer qu'une fois ce dernier remplit. HSERIN et HSEROUT, fonctionne comme SERIN et SEROUT, à l'exception, qu'il n'y a plus d'arguments de pattes, ni de débit.

EXEMPLE:

```
DIM I AS BYTE

HSEROPEN 9600

loop:
    HSERIN  I
    HSEROUT "Number: ", #I, CrLf
GOTO loop
END
```

D - SPI

INSTRUCTIONS:

- SPI_CS_REG
- SPI_CS_BIT
- SPI_SCK_REG
- SPI_SCK_BIT
- SPI_SDI_REG
- SPI_SDI_BIT
- SPI_SDO_REG
- SPI_SDO_BIT
- SPICS_INVERT
- SPICLOCK_INVERT
- SPICLOCK_STRETCH
- SPICSON
- SPICSOFF
- SPIPREPARE
- SPISEND
- SPISENDBITS
- SPIRECEIVE

EXPLICATIONS:

Bien, maintenant, vous avez du remarquer une certaine redondance au départ. Si vous avez lu les pages précédentes, vous aurez deviné que ces instructions(_REG & _BIT) servent à configurer les pattes de communications. Les _INVERT servent à inversé les niveaux logiques des données, pour s'adapter aux matériels.

Le CS étant un signal optionnel, car pas toujours utilisé en communication SPI, les instructions SPICSON & SPICSOFF servent à activer et désactiver ce signal, qui permet de sélectionner le périphérique avec lequel on désire dialoguer.

Afin d'optimiser le fonctionnement des matériels SPI, la fréquence de l'horloge peut être modifiée avec l'instruction SPICLOCK_STRETCH. Par défaut, le facteur est de 1. Ce dernier peut être augmenter afin d'obtenir une période plus élevée, et donc une fréquence moindre.

SPIPREPARE sert à activer la configuration créée. Pour envoyer un octet, on utilise l'instruction SPISEND, et SPIRECEIVE pour recevoir un octet. Enfin, dans le cas, ou moins d'un octet serait à envoyer, il faut utiliser l'instruction SPISENDBITS, en précisant le nombre de bits, le format, et les bits à envoyer.

EXEMPLE:

```
Define SPI_CS_REG = PORTC
Define SPI_CS_BIT = 0
Define SPI_SCK_REG = PORTC
Define SPI_SCK_BIT = 3
Define SPI_SDI_REG = PORTC
Define SPI_SDI_BIT = 5
Define SPI_SDO_REG = PORTC
Define SPI_SDO_BIT = 4
SPIPrepare

DIM data AS BYTE

main:

    SPICSON
    SPISENDBITS 6, %000110

    SPISEND 0xFF
    SPIRECEIVE data
    SPICSOFF

GOTO main
END
```

E - I²C

INSTRUCTIONS:

- I2CWRITE
- I2CREAD
- I2CREAD_DELAYUS
- I2CCLOCK_STRETCH
- I2CWRITE1
- I2CREAD1
- I2CPREPARE
- I2CSTART
- I2CSTOP
- I2CSEND
- I2CRECA
- I2CRECEIVEACK (identique à I2CRECA)
- I2CRECN
- I2CRECEIVENACK (identique à I2CRECN)

EXPLICATIONS:

L'I²C utilise seulement deux fils de communication. Un fil d'horloge (SCL) et un fil de donnée (SDA) bidirectionnel. La structure est simple: un maître et un ou plusieurs esclaves. L'horloge, elle, est générée par le maître. Chaque esclaves possèdent une adresse propre, équivalent de notre numéro de sécurité sociale. Cette adresse permet d'éviter les confusions (Plus d'infos sur l'I²C sont disponible sur la doc en ligne http://diablotronic.bzh.bz).

Commençons par I2CREAD & I2CWRITE. Ces deux instructions permettent de lire et écrire sur le périphérique I²C (souvent une mémoire EEPROM). La structure d'argument est la même pour les deux instructions. D'abord la patte servant au SDA, puis celle du SCL, suivit de l'adresse du périphérique I2C, l'adresse ou lire/écrire la donnée, et enfin, la donnée à écrire/lire.

L'I2CWRITE1 & l'I2CREAD1 sont identique mais n'ont pas besoin de l'argument d'adressage. En effet, certains périphérique I²C ne gèrent pas l'adressage.

I2CREAD_DELAYUS permet de régler une éventuelle tempo d'attente entre la demande de lecture et la réception de donnée, afin que le périphérique soit fonctionnel à la demande effectuée. Par défaut la valeur est de 0 µs.

Tout comme pour le SPI, l'I2CCLOCK_STRETCH permet de modifier la fréquence d'horloge.

I2CPREPARE permet d'initialiser la configuration I²C, entrainant deux configuration possible. Le start et le stop de la trame I²C sont générés par I2CSTART & I2CSTOP. L'acknoledge et le no acknoledge sont générés par I2CRECEIVEACK & I2CRECEIVENOACK. Une bonne compréhension du fonctionnement de l'I²C est nécessaire pour sa mise en oeuvre. En effet, le principe du ACK & NOACK est déroutant.

Deux cas, sont possible. Dans le premier cas, l'initialisation prépare les pattes SDA et SCL. Par conséquent, l'utilisation du I2CSEND est différent, car il nécessite l'I2CSTART et l'I2CSTOP. Dans le deuxième cas, l'I²C est géré en temps réel (initialisation fermeture, ack et noack). La deuxième solutions étant plus simple, c'est celle qui sera utilisée dans l'exemple. Pour voir l'autre solution, rendez vous sur le site de Oshon.

EXEMPLE:

```
DIM DATA AS BYTE
DIM DATA2 AS BYTE
DIM ADDR AS BYTE
DIM AADR_PERIPH AS BYTE

SYMBOL SDA=PORTC.1
SYMBOL SCL=PORTC.2

DATA=0xFF
ADDR=0x2C

main:
	I2CWRITE SDA, SCL, ADDR_PERIPH, ADDR, DATA
	WAITMS	500
	I2CREAD SDA, SCL, ADDR_PERIPH, ADDR, DATA2
GOTO main
END
```

F - USB

INSTRUCTIONS:

- USBSTART
- USBSERVICE
- USBSTOP
- USBIOBUFFER
- USBFTBUFFER
- USBSETVENDORID
- USBSETPRODUCTID
- USBSETVERSIONNUMBER
- USBSETMANUFACTURERSTRING
- USBSETPRODUCTSTRING
- USBSETSERIALNUMBERSTRING
- USBONIOINGOSUB
- USBONIOOUTGOSUB
- USBONFTINGOSUB
- USBONFTOUTGOSUB

EXPLICATIONS:

Commençons une petite explication sur l'usb. Lors de la 1ere connection d'un peripherique au PC, ce périphérique transmet certaine informations au PC, avant d'être opérationnel. Certaines instructions ici servent à cette fin (toutes les instructions avec "SET").. Parmis ces informations, dans l'ordre des instructions énumérés précédemment, il y a le numero d'identification du fabricant (attribué par l'usb.org), le numéro d'identification de produit (65535 possibles par vendorID), la version du produit, le nom du fabricant, le nom du produit , le numéro de série du produit.

Deux types de données peuvent êtres échangées via 2 buffers (USBIOBUFFER & USBFTBUFFER): des données pure, propre (IO), et des rapports de données (FT). Chaque buffer est composé de 8 octets.

Les instructions finissant par "GOSUB" servent aux interruptions permettant l'échange de données via les buffers.

Les instructions USBSTART & USBSTOP servent à activer et

désactiver le port USB du microcontrôleur.

Enfin, l'instruction USBSERVICE, elle, sert à pratiquer l'échange éventuel des buffers entre le PC et le microcontrôleur (c'est l'instruction qui gère les événement USB). ATTENTION, cette instruction doit être utilisée le plus souvent possible (au moins 1 fois / seconde). Si vous avez un problème dans la communication, cela peut en être la cause.

Le PIC ainsi programmé sera configuré en HID, vous permettant ainsi un débit de maximum 64 KO/s. Le HID est gérable par Visual Basic (voir après l'exemple), et par java avec certaines API, trouvable sur le NET.

EXEMPLE:

```
UsbSetVendorId 0x1234
UsbSetProductId 0x1234
UsbSetVersionNumber 0x1122
UsbSetManufacturerString "Diablotronic"
UsbSetProductString "Generic USB HID Device"
UsbSetSerialNumberString "1111111111"
UsbOnIoInGosub usbonioin
UsbOnIoOutGosub usbonioout
UsbOnFtInGosub usbonftin
UsbOnFtOutGosub usbonftout

AllDigital
ADCON1 = 0x0e          'activation des CAN
TRISB = 0
PORTB = 0xff
UsbStart
PORTB = 0

Dim an0 As Byte
Dim i as Byte

loop:
        Adcin 0, an0
        If an0 < 50 Then
                PORTB = 0
                UsbStop
                While an0 < 100
```

```
                Adcin 0, an0
            Wend
            PORTB = 0xff
            UsbStart
            PORTB = 0
        Endif
        UsbService
Goto loop
End

usbonftout:
Toggle PORTB.7
Return

usbonftin:
For i=0 to 7
UsbFtBuffer(i) = UsbFtBuffer(i) - 1
Next i
Return

usbonioout:
Toggle PORTB.6
Return

usbonioin:
For i=0 to 7
UsbIoBuffer(i) = UsbIoBuffer(i) +1
Next i
Return
```

MOT DE CONFIGURATION:

Si vous avez acheté l'option pour communiquer en USB, vous avez peut être été confronté à des problèmes de mise en œuvre. Ce problème vient principalement du mot de configuration. En effet, si pour un PIC12 ou un PIC16, ce mot est simple à configurer, c'est une tout autre affaire pour un PIC18. Reprenant le code en ligne de Oshon, j'ai reconstitué la configuration fonctionnelle, en précisant toutefois 3 paramètre n'ayant pas de réelle influence sur l'USB. Pour le reste, libre à vous de tester des configurations dérivées. Respecter le schéma électronique du site de OSHON.

PLL PRESCALER SELECTION: à 5, quartz de 20 Mhz
SYSTEM CLOCK POSTCALER SELECTION: PLL2, sans post diviseur
USB CLOCK SELECTION: USB_DIV2, avec PLL
OSCILLATOR SELECTION: HS, quartz externe 20 Mhz
FAIL SAFE CLOCK MONITOR: OFF
INTERNAL/EXTERNAL SWITCHOVER SELECTION: OFF
POWERUP TIMER: ON
BROWN OUT RESET: ON
BROWN OUT RESET VOLTAGE: 21
USB INTERNAL VOLTAGE REGULATOR: ON
WATCHDOG TIMER: OFF
WATCHDOG TIMER POSTSCALE: MODE 1
\MCLR PIN SELECCTION: au choix
LOW POWER TIMER 1 OSCILLATOR: OFF
PORT B A/D SELECTION: au choix
CCP2 MULTIPLEX SELECTION: au choix, de préférence à ON
STACK FULL/UNDERFLOW RESET ENABLE: OFF
SINGLE SUPPLY ICSP ENABLE: OFF
DEDICATED IN CIRCUIT DEBUG-PROGRAMMING PORT: OFF
EXTENDED INSTRUCTION SET ENABLE: OFF
BACKGROUNG DEBUGGER: OFF
BLOCK x PROGRAM FLASH PROTECTION: OFF
BOOT BLOCK CODE PROTECTION: OFF
DATA E²PROM CODE PROTECTION: OFF
BLOCK x PROGRAM FLASH WRITE PROTECTION: OFF
CONFIGURATION REGISTER WRITE PROTECTION: OFF
BOOT BLOCK WRITE PROTECTION: OFF
DATA E²PROM WRITE PROTECTION: OFF
BLOCK x TABLE READ PROTECTION: OFF
BOOT BLOCK TABLE READ PROTECTION: OFF

NIVEAU LOGICIEL PC, VISUAL BASIC:

Au niveau du logiciel PC, créé sous visual BASIC, peu d'instructions sont nécessaires. Nous allons voir ici le strict minimum d'instructions nécessaire pour pouvoir communiquer avec le PIC, et géré la communication.

Pour des raisons de commodité, nous appellerons la fenetre principale f1. Vous devez également installer un composant supplémentaire: un contrôle active x pour gérer la communication HID. Ici, deux solutions: le HIDTerm de Oshon Software, livré avec l'extension USB de PIC18.

Il existe aussi le HIDCOMM créer par Microchip (sur notre site).

Bien entendu, en tant que périphérique USB à part entière, vous pouvez débranchez ce dernier du PC via la petite icône dans la zone de notification.

Bien, maintenant voyons les principales commandes de l'activeX de chez Oshon. Par principales, j'entend qu'avec ces commandes, on est capables de gérer correctement une communication.

Tout d'abord, initialisé les numéros de VID & PID auquel on veut se connecter.

-f1.hid_vendorid
-f1.hid_productid

Voici la commande, à utilisée telle quelle, permettant de se connecter au périphérique, identifié par les ID précédemments définis.

-call f1.hidconnect

Ensuite la commande d'accusé de connection, renvoyant TRUE ou FALSE selon le cas

-f1.hid_detected

Enfin, les instructions permettant d'envoyer et de recevoir, via 2 buffers de 8 octets, des données. Chaque commandes retourne TRUE ou FALSE, témoignant ainsi de l'échec ou de la réussite de l'opération. Les 8 octets sont séparés par des virgules.

-f1.hidreadfeature(8 variables)
-f1.hidsendfeature(8 données)
-f1.hidreadreport(8 variables)
-f1.hidsendreport(8 données)

Face à la simplicité d'emploi de cet activeX comparé à celui de Microchip, celui de Microchip ne sera pas explicité ici. De Toute façon, l'aide livré avec l'activeX permet, avec un minimum d'étude de tout gérer, mais de manière un peu plus compliqué. L'activeX de Microchip est disponible dans la section téléchargement/électronique de notre site.

Opto

A – LCD Graphique (PIC&PIC18)

B – LCD classique (PIC&PIC18)

A – LCD GRAPHIQUE

Nous verrons ici, les instructions permettant de gérer un écran LCD graphique 128*64 avec interface KS017 ou compatible.

INSTRUCTIONS:

- GLCD_DREG
- GLCD_RSREG
- GLCD_RSBIT
- GLCD_EREG
- GLCD_EBIT
- GLCD_RWREG
- GLCD_RWBIT
- GLCD_CS1REG
- GLCD_CS1BIT
- GLCD_CS2REG
- GLCD_CS2BIT
- GLCDINIT
- GLCDCLEAR
- GLCDPSET
- GLCDPRESET
- GLCDCLEAN
- GLCDPOSITION
- GLCDWRITE
- GLCDOUT
- GLCDIN
- GLCDCMDOUT

EXPLICATIONS:

Comme d'habitude, les premières lignes permettent de définir les lignes utilisés pour communiquer avec le LCD, les _REG permettant de définir le port utilisé, et les _BIT la patte utilisée, sur ce port.

GLCDINIT doit être placé immédiatement après la config, et au début du programme. GLCDCLEAR sert à effacer l'écran. Un argument peut être utilisé. GLCDCLEAN sert à effacer qu'une partie de l'écran, et utilise au moins un argument.

GLCDPSET et GLCDPRESET servent à activer ou éteindre un pixel de l'écran. Les deux arguments sont les coordonnées, X et Y, du pixel.

GLCDPOSITION sert à indiquer des coordonnées ou placer un caractère. Cette commande est à utilisée avant d'envoyer le caractère (commandes suivantes).

GLCDWRITE sert à écrire du texte sur l'écran LCD. GLCDOUT sert à afficher une valeur de variable à la position courante (on peut aussi utiliser GLCDWRITE avec "#"), GLCDIN sert à lire ce qui est affiché sur l'écran à la position courante, GLCDCMDOUT sert à envoyer des commandes diverses à l'écran LCD.

EXEMPLE:

```
'DEFINITION DES PATTES UTILISÉES NON EXPLICITÉES

GLCDINIT

Dim i as Byte

main:
    GLCDPOSITION 60,32
    GLCDWRITE "BONJOUR, numéro:", #i
    WAITMS 1000
    GLCDCLEAR
    GOTO main
END
```

B – LCD classique

INSTRUCTIONS:

- LCD_BITS
- LCD_DREG
- LCD_DBIT
- LCD_RSREG
- LCD_RSBIT
- LCD_EREG
- LCD_EBIT
- LCD_RWREG
- LCD_RWBIT
- LCD_COMMANDUS
- LCD_DATAUS
- LCD_INITMS
- LCD_READ_BUSY_FLAG
- LCD_LINES
- LCD_CHARS
- LCDINIT
- LCDOUT
- LCDCMDOUT
- LCDCLEAR
- LCDHOME
- LCDLEFT
- LCDRIGHT
- LCDSHIFTLEFT
- LCDSHIFTRIGHT
- LCDLINE1HOME
- LCDLINE2HOME
- LCDLINE3HOME
- LCDLINE4HOME
- LCDLINE1CLEAR
- LCDLINE2CLEAR
- LCDLINE3CLEAR
- LCDLINE4CLEAR
- LCDLINE1POS
- LCDLINE2POS
- LCDLINE3POS
- LCDLINE4POS

- LCDDEFCHAR
- LCDDISPLAYON
- LCDDISPLAYOFF
- LCDCURBLINK
- LCDCUROFF
- LCDCURUNDERLINE
- LCDCURBLINKUNDERLINE

EXPLICATIONS:

LCD_BITS sert à définir le nombre de bits de donnée (4 ou 8), par défaut à 4.

LCD_DREG sert à définir le port sur lequel sont connectés les lignes de données (port b par défaut).

LCD_DBIT sert à définir la patte de la 1ere ligne de donnée (0 ou 4) pour 4 fils de donnée, par défaut à 4. Inutil pour 8 fils de données.

LCD_RSREG est par déaut sur le port B. LCD_RSBIT par défaut est à 3.

LCD_EREG par défaut sur le port B. LCD_EBIT par défaut est à 2.

LCD_RWREG sur le port B par défaut. LCD_RWBIT par défaut à 1. Si cette instruction n'est pas utilisée, la valeur par défaut pour les RW sera alors de 0. LCD_READ_BUSY_FLAG s'utilise avec le RW.

LCDINIT doit être placé juste après la configuration des pattes, au tout début du programme. Un argument optionnel peut être utilisé afin de configurer le curseur (0: sans curseur (par défaut), 1: clignote, 2: souligné, 3: clignote et souligné).

Le type d'écran est défini avec LCD_LINES (nombre de lignes) et LCD_CHARS (nombre de caractères).

LCDOUT sert à envoyer des données, et LCDCMDOUT sert à envoyer un mot de commande. Les mot de commandes sont l'effacement de lignes ou d'écran (LCDCLEAR, LCDLINE1CLEAR, LCDLINE2CLEAR, LCDLINE3CLEAR, LCDLINE4CLEAR), position d'écran (LCDLINExPOS, avec pour argument un chiffre entre 1 et 40, et les instructions avec HOME

permettant de revenir à la case 0).

Des caractères peuvent être défini par vos soins via la commande LCDDEFCHAR, 8 sont possibles généralement. Ce sera le 1er argument (0 à 7). Un caractère d'écran LCD est composé de 5 pixels de large par 8 pixels de haut, et sont décrit du haut vers le bas. Ce seront les arguments suivants: 8 fois 5 bits (les bits non utilisés sont remplacés par des 0).

```
LCDDEFCHAR 0, %11111, %00000, %00000, %11111, ...
LCDOUT 0
```

EXEMPLE:

```
'DÉFINITION DES PATTES UTILISÉES NON EXPLICITÉS

LCDINIT

main:
    LCDCMDOUT LCDLINE1HOME
    LCDOUT "ligne 1"
    LCDCMDOUT LCDLINE2HOME
    LCDOUT "ligne 2"
    LCDCMDOUT LCDLINE1CLEAR
    LCDCMDOUT LCDLINE1HOME
    LCDOUT "ligne 1 effacée"
    LCDCMDOUT LCDCLEAR
    LCDCMDOUT LCDLINE2HOME
    LCDOUT "ECRAN EFFACEE"
    GOTO main
End
```

Interface

A – PWM (PIC&PIC18)

B – ADC (PIC&PIC18)

C – Servo moteur (PIC&PIC18)

D – DS18S20

E – Moteur pas à pas

A – PWM

INSTRUCTIONS:

- PWMON
- PWMDUTY
- PWMOFF

EXPLICATIONS:

La PWM, ou Pulse Width Modulation, repose sur un principe simple: la variation du rapport cyclique du signal. Ce faisant, il est possible de faire varier la luminosité d'une ampoule par exemple.

Comme l'évidence l'indique, PWMON et PWMOFF servent à allumer et éteindre la PWM. PWMON nécessite deux argument, le module PWM utilisé, et le second le mode (voir ci après).PWMOFF nécessite un argument, le numéro du module PWM.

PWMDUTY sert à varier la position haute du rapport cyclique. Par défault, ce dernier est à 0. Il peut varier jusqu'à 1023 (10bits), 511 (9 bits), 255 (8 bits), 127 (7bits). Les fréquence possibles pour une fréquence de 4 Mhz sont:

mode 1: 10-bit, 244Hz
mode 2: 10-bit, 977Hz
mode 3: 10-bit, 3906Hz
mode 4: 9-bit, 488Hz
mode 5: 9-bit, 1953Hz
mode 6: 9-bit, 7813Hz
mode 7: 8-bit, 977Hz
mode 8: 8-bit, 3906Hz
mode 9: 8-bit, 15625Hz
mode 10: 7-bit, 1953Hz
mode 11: 7-bit, 7813Hz
mode 12: 7-bit, 31250Hz

Pour les autres fréquence, les changements sont proportionnels. Maintenant, nous pouvons voir le PWMDUTY. Son 1er argument est le module PWM, et son second, son rapport cyclique (état haut sur état bas).

EXEMPLE:

```
PWMON 1, 9
PWMDUTY 1, 150
PWMOFF 1
```

B - ADC

L'utilisation de ADC presuppose certaines contraintes: configuration des pattes concernée en entrée, et configuration de ADCON1.

INSTRUCTIONS:

- ADCIN
- ADC_CLOCK
- ADC_SAMPLEUS

EXPLICATIONS:

ADCIN utilise deux arguments: le numéro du CAN utilisé, puis la variable de stockage (de destination).

ADC_CLOCK permet de sélectionner l'horloge de convertion (0 à 3, ou 0 à 7 selon le pic). Par défaut, la valeur est de 3.

ADC_SAMPLEUS sert à modifier le nombre d'acquisition par µs (de 0 à 255) réglé par défaut à 20.

EXEMPLE:

```
Dim i as Byte
ADCON=0x00

main:
        ADCIN 0,i
        GOTO main
End
```

C – Servo moteur

La possibilité de commander des servomoteurs est fort intéressant pour la robotique, et ses dérivés. Un servomoteur est piloté par 15-20 impulsions par seconde.

INSTRUCTIONS:

-SERVOIN
-SERVOOUT

EXPLICATIONS:

Ces instructions nécessitent seulement 2 arguments. Le 1er est la patte de communication utilisée. Le second arguments dépend de l'instruction.

Pour SERVOIN, il s'agit de la variable dans laquelle sera stockée la longueur de l'impulsion. La valeur récupéré, de type BYTE, est un multiple de 10 µs et est comprise entre 10µs et 2,55 ms.

Pour SERVOOUT, il s'agit de la variable contenant la valeur du multiple, avec généralement un max de 200, ce qui fait de 10 µs à 2 ms.

EXEMPLE:

```
main:
    SERVOOUT PORTB.1, 100
    GOTO main
End
```

D – DS18S20

Le DS18S20 est un capteur numérique de température, communiquant sur un fil (de type 1 fil).

INSTRUCTIONS:

- DS18S20START
- DS18S20READT

EXPLICATIONS:

DS18S20READT nécessite deux argumets. Le 1er récupère la température, le deuxième, le signe. Un delais de 1 s est nécessaire entre l'établissement de la communication et la réception des données.

EXEMPLE:

```
DS18S20START
WAITMS 1000
DS18S20READT temp, sign
```

E – Moteur pas à pas

Il est possible de commander des moteurs pas à pas, via une interface adéquat (tel des transistors), à partir des microcontrôleurs.

INSTRUCTIONS:

-STEP_A_REG
-STEP_A_BIT
-STEP_B_REG
-STEP_B_BIT
-STEP_C_REG
-STEP_C_BIT
-STEP_D_REG
-STEP_D_BIT
-STEP_MODE
-STEPHOLD
-STEPCW
-STEPCCW

EXPLICATIONS:

A ce stade, vous aurez reconnu les classiques _REG et _BIT, afin de fournir les pattes ou sont connectées les fils moteurs (ou interface).

STEP_MODE sert lui à dire si le moteur est utilisé en pas (1, défaut) ou en demi pas(2).

STEPHOLD sert à appliquer les paramètres définit precedemment.

Enfin, STECW et STECCW servent à faire tourner le moteur dans le sens horaire et antihoraire. Le 1er argument est le nombre de pas de rotation, et le second le temps en µs entre chaque pas.

EXEMPLE:

```
Define STEP_A_REG = PORTB
Define STEP_A_BIT = 7
Define STEP_B_REG = PORTB
Define STEP_B_BIT = 6
```

```
Define STEP_C_REG = PORTB
Define STEP_C_BIT = 5
Define STEP_D_REG = PORTB
Define STEP_D_BIT = 4
Define STEP_MODE = 2

STEPHOLD

main:
        STEPCW 8, 300
        goto main
end
```

Utilitaire

A - Tempo (PIC&PIC18)

B - Fonctions mathématique (PIC&PIC18)

C - Fonctions logique (PIC&PIC18)

D - Fonctions condition (PIC&PIC18)

E - Fonctions pattes (PIC&PIC18)

F - Interruptions (PIC&PIC18)

G - Variables (PIC&PIC18)

H – EEPROM

I – COMPTEUR

J – GENERATION AUDIO

K – FONCTIONS ANNEXES

A – Tempo

INSTRUCTIONS:

- WAITMS
- WAITUS

EXPLICATIONS:

Les instructions, ici, sont très simple. La première doit être suivit d'un entier indiquant une valeur de millisecondes, et la secondes de microsecondes.

EXEMPLE:

```
main:
    portB.6=1
    WAITMS 1000
    PORTB.6=0
    WAITMS 1000
    GOTO main
ENDs
```

B - Fonctions mathématique

INSTRUCTIONS:

- MOD (modulo)
- SQR (racine carré)
- +
- -
- *
- /

EXPLICATIONS:

Les 4 dernières "Instructions" sont les opérations authorisées. Outre cela, le compilateur offre la possibilitée de calculer une racine carrée (SQR), et le modulo d'une division (MOD).

EXEMPLE:

```
Dim i as Byte

main:
        i=SQR(i)
        GOTO main
END
```

C - Fonctions logique

INSTRUCTIONS:

- AND
- NAND
- NOR
- NOT
- NXOR
- OR
- XOR

EXPLICATIONS:

Ici rien de particulier à expliquer. Il s'agit d'instruction logique classiques.

D - Fonctions condition

INSTRUCTIONS:

- IF X THEN
  ...
  ELSE
  ...
  ENDIF

- FOR X = Y TO Z
  ...
  NEXT X

- FOR X = Y TO Z STEP R
  ...
  NEXT X

- WHILE
  ...
  WEND

- SELECT CASE X
  CASE 1
  ...
  CASE <= 2
  ...
  CASE ELSE
  ...
  ENDSELECT

- GOSUB

- ASM

EXPLICATIONS:

Toutes les premières instructions sont assez classiques et se passes de toutes explications. Nous nous attarderons donc sur la 3ème. Basée sur une instructions FOR, l'intérêt se situe au niveau du STEP. En effet, le STEP permet de donner un pas d'incrémentation ou de décrémentation.

GOSUB, sert à appeler un sous programme. Les sous programmes sont écris après le END du main, et se finissent par l'instruction RETURN.

ASM, permet d'écrire du code assembleur. Par exemple, l'instruction sleep n'étant pas disponible en assembleur, il faut alors utiliser cette instruction.

EXEMPLE:

```
main:
        FOR i=0 to 10 STEP 2
                GOSUB latch()
        NEXT i
        ASM: SLEEP
        GOTO main
END
```

E - Fonctions pattes

INSTRUCTIONS:

- 0 logique
- 1 logique
- TOGGLE
- ALLDIGITAL
- SYMBOL

EXPLICATIONS:

Les pattes d'un microcontroleur, en mode digital, peuvent prendre deux états: 1 logique, ou 0 logique. Le passage d'un état à l'autre peut se faire via l'instruction TOGGLE.

ALLDIGITAL permet de passer toute les pattes d'un PIC en mode digital. Cette instruction est très pratique lorsque le PIC possède plusieurs CAN.

SYMBOL, enfin, permet de renommer une patte afin que le programme soit plus lisible.

EXEMPLE:

```
ALLDIGITAL

SYMBOL LED= PORTB.1

main:
        TOGGLE LED
        WAITMS 1000
        GOTO main
END
```

F – Interruptions

INSTRUCTIONS:

- ON INTERRUPT
- RESUME
- SAVE SYSTEM
- ENABLE
- DISABLE

EXPLICATIONS:

Créer une interruption permet de parcourir normalement un programme tout en surveillant un événement attendu. Si cet événement se produit, le programme normal est alors interrompu, afin d'exécuter un programme secondaire.

Les interruptions (dont plusieurs sources sont possibles, même si un seul programme d'interruption est possible), permettent par exemple une économie d'énergie. Plus clairement, il est possible d'endormir un PIC avec "asm: sleep". Cette commande, en assembleur plonge le pic en sommeil, et sa consommation chute au quasi nul (<1mA). Un changement sur la source d'interruption, le réveil. L'intérêt réside ici dans l'économie d'énergie pour les appareils nomade.

Le bit GIE du registre INTCON qui active ou non les interruptions, est commandé via les instructions ENABLE & DISABLE. Les différentes sources d'interruptions se gèrent via le registre INTCON, PIE1, PIR1, IOC (selon les modèles de PIC, tous ces registres ne sont pas disponibles).

ON INTERRUPT sert à décrire le programme d'interruption, alors que RESUME sert à retourner là où le programme normal s'est arrêter.

SAVE SYSTEM doit être utilisé afin de sauvegarder les éventuelles données en traitement lors de l'interruption.

EXEMPLE:

```
ENABLE
main:
        ADCIN 1, data
        GOTO main
END

ON INTERRUPT
        SAVE SYSTEM
        PORTA.1 = 0
RESUME
```

G – Variables

INSTRUCTIONS:

- BIT (1 bit)
- BYTE (1 octet)
- WORD (2 octets)
- LONG (4 octets) (Attention, option)
- .HB
- .LB
- .HW
- .LW
- TRUE
- FALSE
- CONST
- SHIFTLEFT
- SHIFTRIGHT

Un nombre décimal s'écrit normalement, le symbole 0x précède l'hexa, et % le binaire.

EXPLICATIONS:

Les 4 premieres instructions servent à déterminer le type de la variables. Il est possibles de créer un tableau en indiquant le nombre entre parenthèse (ex: *DIM A(10) AS BYTE).*

Dans le cas d'un WORD, il est possible de séparer l'octet haut de l'octet bas via .HB et .LB. De même dans le cas d'un LONG, il est possible de séparer le WORD haut du WORD bas via .HW et .LW.

Les valeurs logique peuvent êtres lues et remplacées par TRUE et FALSE.

Enfin, CONST permet de définir des constantes.

Attention, le compilateur ne gere pas les chiffres à virgules. Si vous avez besoin, pensez à multiplier par une puissance de 10.

SHIFTLEFT & SHIFTRIGHT permettent de faire des roulement de variables. Par roulement entendez le fait de décaler le code binaire vers la gauche ou la droite. Tout nombre sortant de la variable à un bout repasse à l'autre bout, tournant ainsi en boucle. Le 1er argument est le nom de la variable, le second argument est le nombre de décalage.

EXEMPLE:

```
DIM A(10) AS WORD
DIM i AS BYTE
CONST PI=314

main:
        i=A(0).LB
        GOTO main
END
```

H – EEPROM

INSTRUCTIONS:

- READ
- WRITE

EXPLICATIONS:

Rien de bien compliqué ici. Deux arguments: l'octet d'adresse, suivit de l'octet à lire/écrire. Attention, veuillez à désactiver les interruptions (DISABLE) durant une opération sur l'EEPROM.

EXEMPLE:

```
DIM i AS BYTE

main:
        READ 0x12, 0x23
        WRITE 0x14, I
        GOTO main
END
```

I – COMPTEUR

INSTRUCTIONS:

- COUNT_MODE
- COUNT

EXPLICATIONS:

Ces instructions se basent sur le TOCKI des PICs. Grâce à elles, vous êtes capables de compté des fronts montants (mode 1) ou descendant (mode 2), et ce sur une période donnée.

Les arguments de COUNT sont la patte de comptage, suivit de la période de comptage en ms, et enfin la variable ou va etre stocké le nombre calculé.

EXEMPLE:

```
COUNT_MODE=1
DIM i AS BYTE

main:
        COUNT PORTB.0, 1000, i
END
```

J – GENERATION AUDIO

INSTRUCTIONS:

- FREQOUT

EXPLICATIONS:

Sous cette instruction se cache la possibilité de réalisé de la génération audio. Le 1er argument est la patte ou est branché le HP, le second la fréquence en hertz du signal, le troisième, la durée du signal en ms.

A noter que vous pouvez définir (CONST) les notes classiques, dont voici les fréquences arrondies: DO(264), DO#(275), RE(297), RE#(317), MI(330), FA(352), FA#(371), SOL(396), SOL#(413), LA(440), LA#(475), SI(495), DO(528).

Grâce à cela, vous pouvez rendre votre code plus lisible.

EXEMPLE:

```
main:
        FREQOUT PORTB.0, LA, 1000
ENDs
```

K – FONCTIONS ANNEXES

INSTRUCTIONS:

- FUNCTION

- PROC
-CALL

EXPLICATIONS:

Le but, ici, est de définir des fonctions que l'on peut appelées à souhait. Imaginons par exemple qu'une même tache se réalise 5 fois dans le programme. Il est alors intéressant de réaliser une fonction de cette tache, et de l'appeler 5 fois, entrainant ainsi une économie de place et une optimisation du code.

La différence entre ces 2 instructions est fort simple: Lors de l'appel de la première, avec passage d'arguments éventuel, la fonction nous retourne une valeur. La seconde, elle, non.

EXEMPLE:

```
DIM i AS BYTE
DIM j AS BYTE

main:
        CALL temp (j)
        i=val(j)
END

PROC temp(j as BYTE)
        WAITMS j
END PROC

FUNCTION val(j AS BYTE) AS BYTE
        j=j*j
END FUNCTION
```

Code supplémentaire

A - DACOUT (PIC&PIC18)

B - ADKEY (PIC&PIC18)

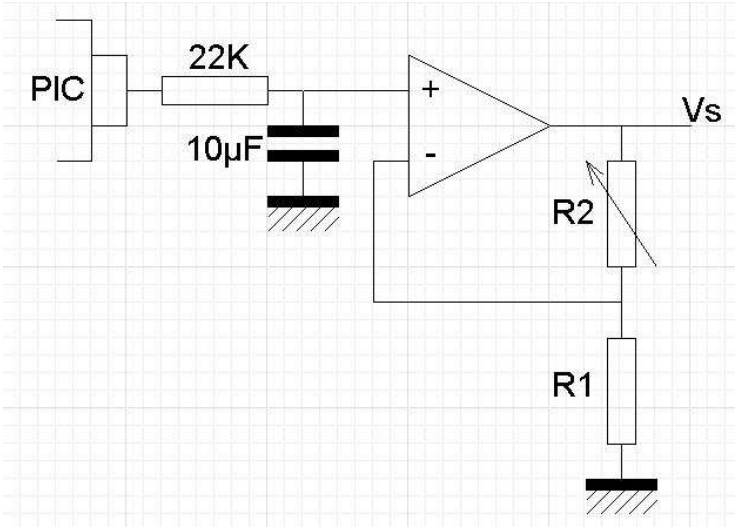
C – SDCARD (PIC&PIC18)

A - DACOUT

EXPLICATIONS:

Sous ce nom se cache un principe très simple. Le but ici n'est pas de réaliser un CNA ultra rapide mais par exemple, de traduire une variable numérique en constante analogique.

Cette fonction ne pourra être réalisé que sur les sorties PWM. Le principe est fort simple. Il s'agit de gérer la charge d'un condensateur via une résistance. Pour prévenir une éventuelle charge, un AOP suiveur sera nécessaire par la suite.

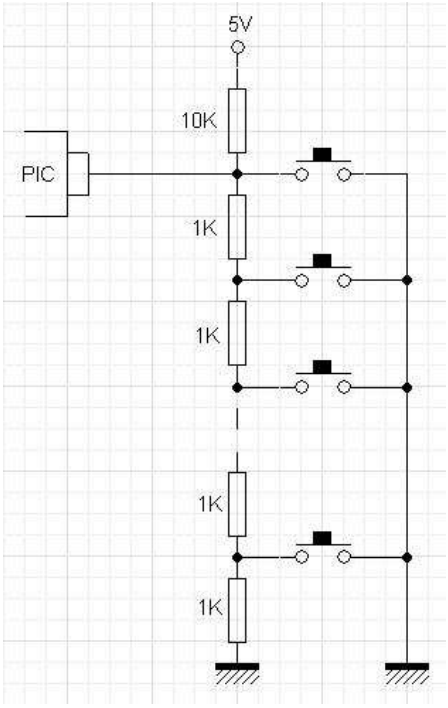


B - ADKEY

EXPLICATIONS:

Pour ceux qui ont déjà utiliser les PICBASIC de chez COMFILE, cette instruction leur est familière, puisqu'inspiré de la très pratique ADKEYIN. Au lieu de monopoliser des entrées/sorties sur un clavier matriciel, cette fonction permet de disposer d'un clavier de 10 touches sur une seule entrée analogique.

Basé sur le principe des pont diviseur, chaque touche du clavier équivaut à un niveau de tension. De fait, Juste en lisant la valeur de la tension d'entrée, il est possible de connaître la touche appuyée.



Sachant qu'un bouton correspond à une valeur de conversion, il suffit de lire l'entrée, et de faire des boucles "if x<=Y then" en commençant par la a valeur la plus basse et en montant. Ici, X est l'entrée et Y est la conversion.

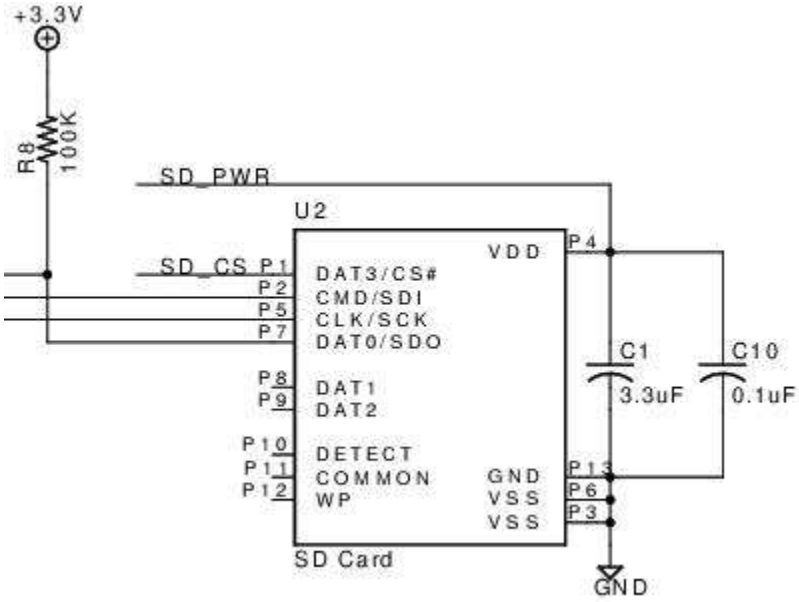
Un maximum de 10 touches est possibles pour une bonne lecture. Le taux d'erreur maximum avec des résistances à 5% est de 10%.

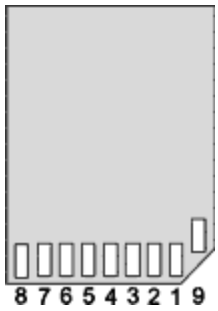
C – SDCARD

EXPLICATIONS:

Le principe utilisé ici est très simple. Il s'agit de l'utilisation de la communication SPI, car, aussi surprenant que cela puisse parraitre, les carte SD utilise la SPI comme interface.

EXEMPLE:





Comme nous pouvons le voir, le connecteur des SD card est simple, à connecter en SPI.

Le SD_POWER vaut 3,3V. La résistance de tirage reliée au 3,3V, sert à garanti l'alimentation en courant.

Attention toutefois, car certaine règle sont à respecter au niveau des fréquences d'horloge.