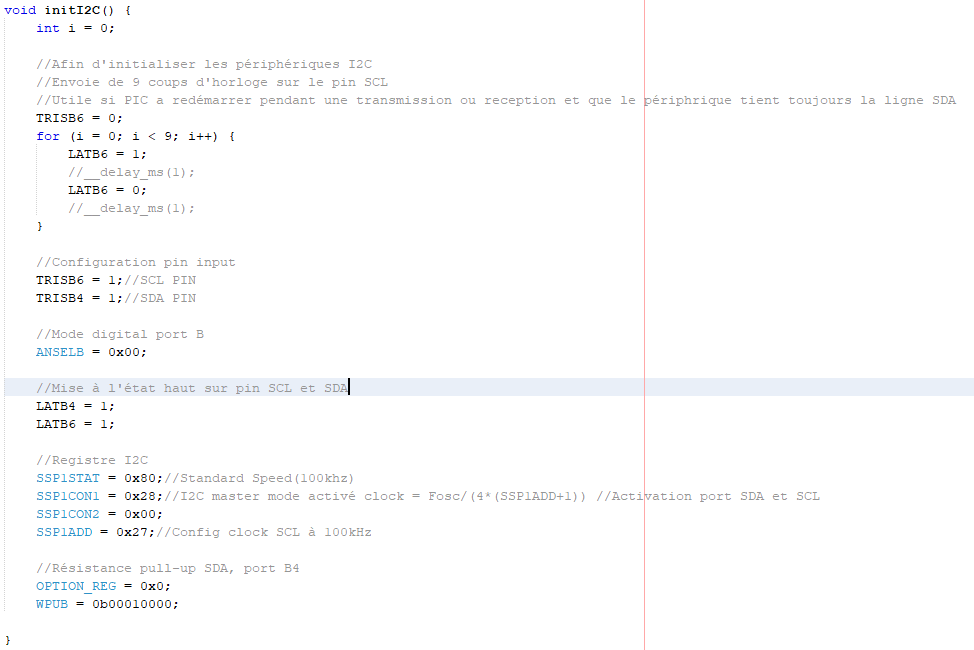
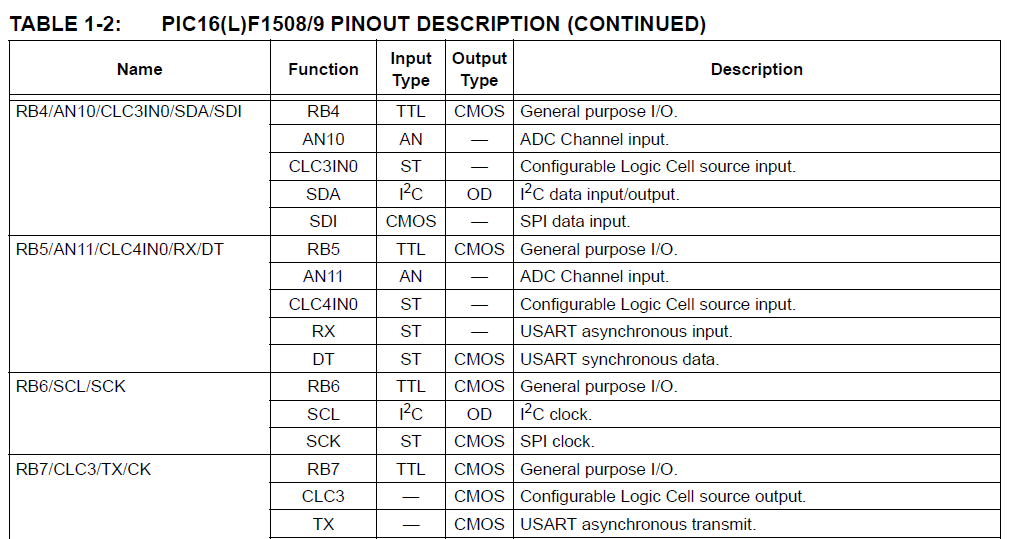
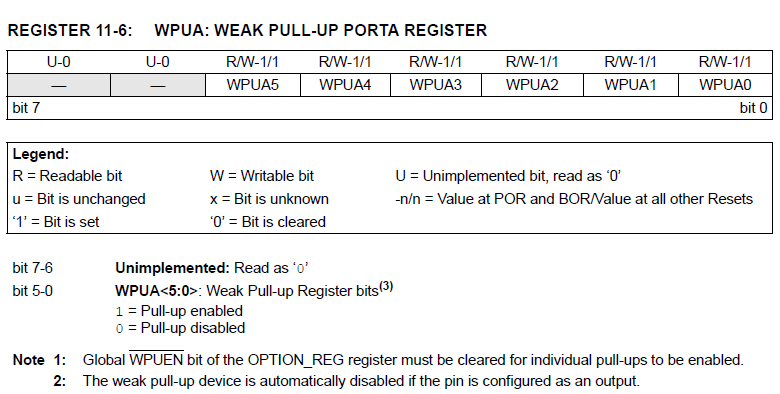
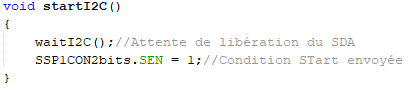
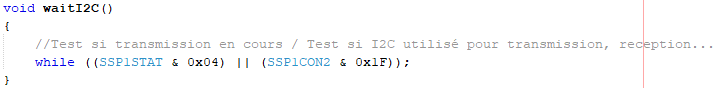
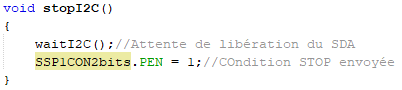
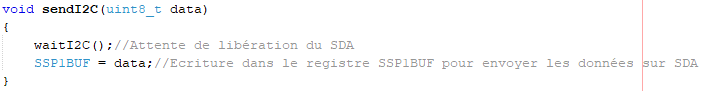
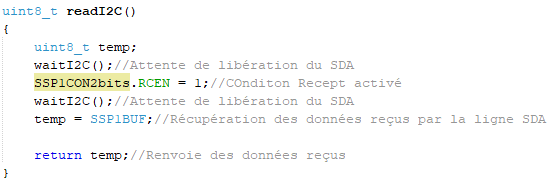
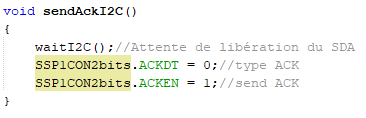
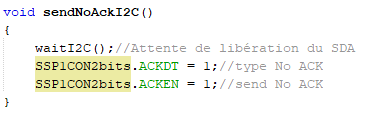
## Code des fonctions de la liaison I2C

Dans le fichier *i2c.c*, nous initialisons notre liaison i2c grâce à *initI2C()*:

Nous avons pu sélectionner toutes ces configurations grâce au MCC mais aussi grâce à la datasheet du PIC. En effet, les captures qui suivent prouvent les sorties des pins de SCL et SDA. Mais aussi l’explication de la résistance de pull-up sur SDA.



Ensuite, nous avons définis plusieurs petites fonctions ayant des rôles bien précis :

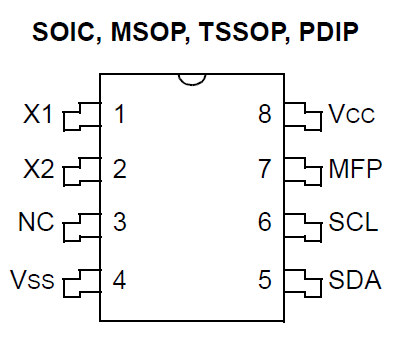
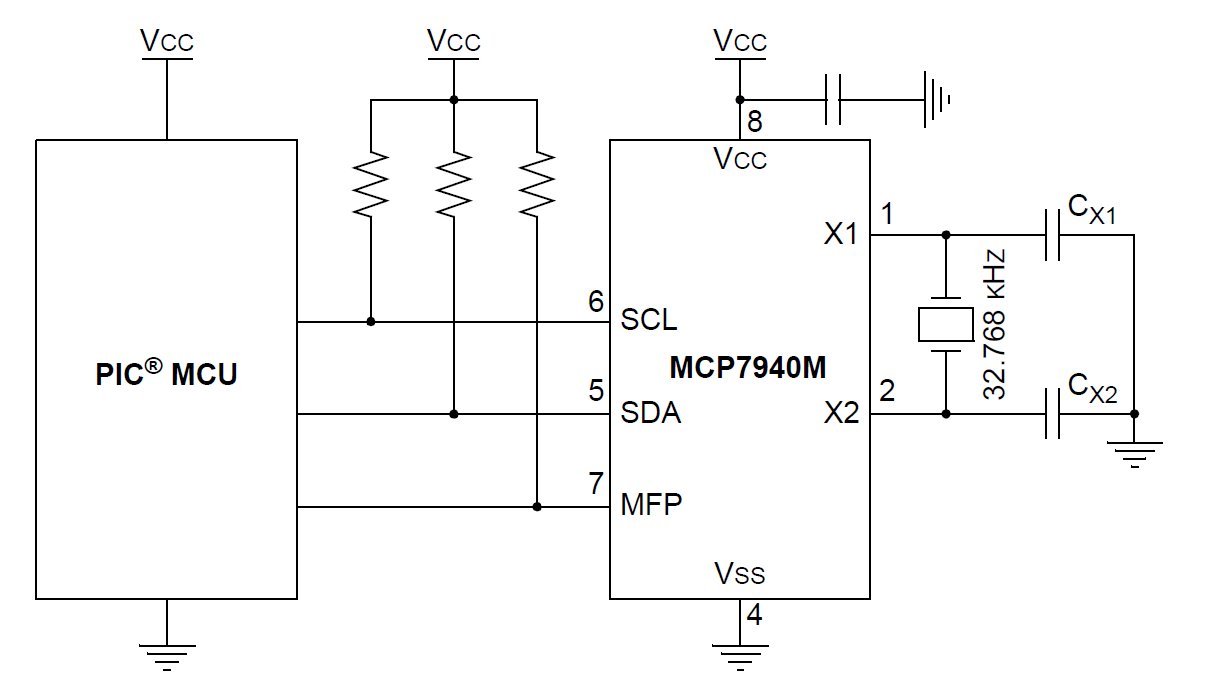
1.  **startI2C()** : ici, nous cherchons à prévenir le début de la transmission de données
2. **waitI2C()** : cette fonction sert à laisser du temps pour que la transmission de données se fasse
3. **stopI2C()**: nous cherchons à prévenir que la transmission de données est finie
4. **sendI2C()**: la valeur que nous souhaitons utilisée plus tard est stockée dans le registre SSP1BUF
5. **readI2C()**: la donnée est récupérée et retournée
6. **sendAckI2C()**: nous vérifions que la donnée a bien été reçue
7. **sendNoAckI2C()**: nous annonçons que la transmission touche à sa fin

## Module RTC + Oscillateur

## Présentation

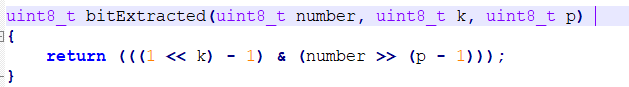
Notre module RTC a pour but de nous fournir une horloge temps réel plus précise que l’oscillateur interne de notre PIC. Pour ce faire, notre PIC et le module RTC dialoguent à l’aide d’une liaison I2C.

Tout d’abord, pour le montage de notre module RTC nous avons procéder de la manière suivante :

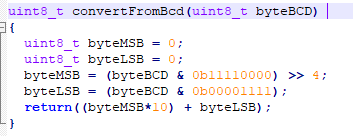


En effet, grâce à la datasheet de notre composant nous pouvons suivre le schéma de pin lay-out afin de pouvoir souder les pins de notre composant sans commettre d’erreur. Puis, nous devons activer notre oscillateur externe car nous utilisons un quartz externe brancher sur les pin X1 et X2 de 32Khz pour cela nous envoyons d’abord l’adresse de notre module suivis ensuite par l’adresse du registre que nous voulons modifier et pour finir les bits du registre que nous désirons, nous avons juste besoin d’initialiser le bit 7 à 1 pour activer notre oscillateur et suite a cela ce dernier va compter tout seul.

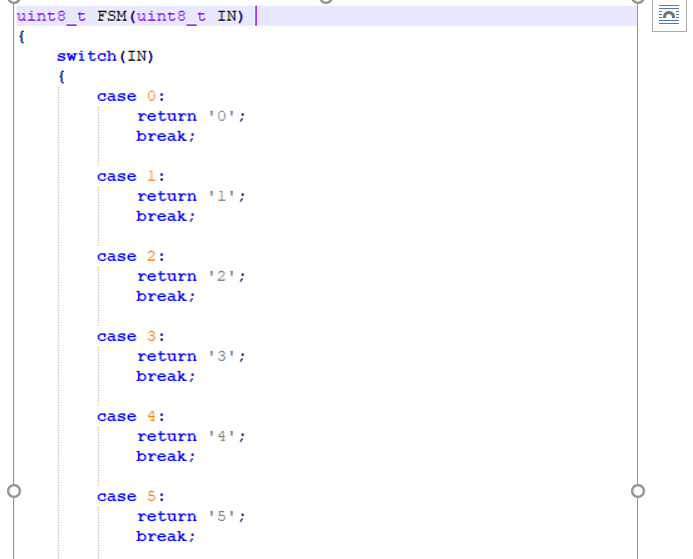
1. Fonction liée au module RTC
2. **bitExtracted()** permet d'extraire un certain nombre de bits depuis un certain point dans l'octet

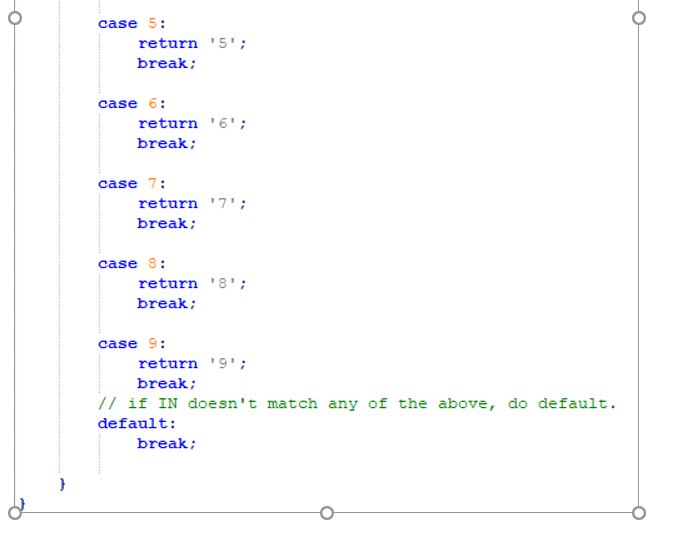


1. **convertFrombcd()** permet de convertir un binary coded décimal en décimal classique, plus pratique pour voir les valeurs au débugger, inutile dans le programme en lui-même

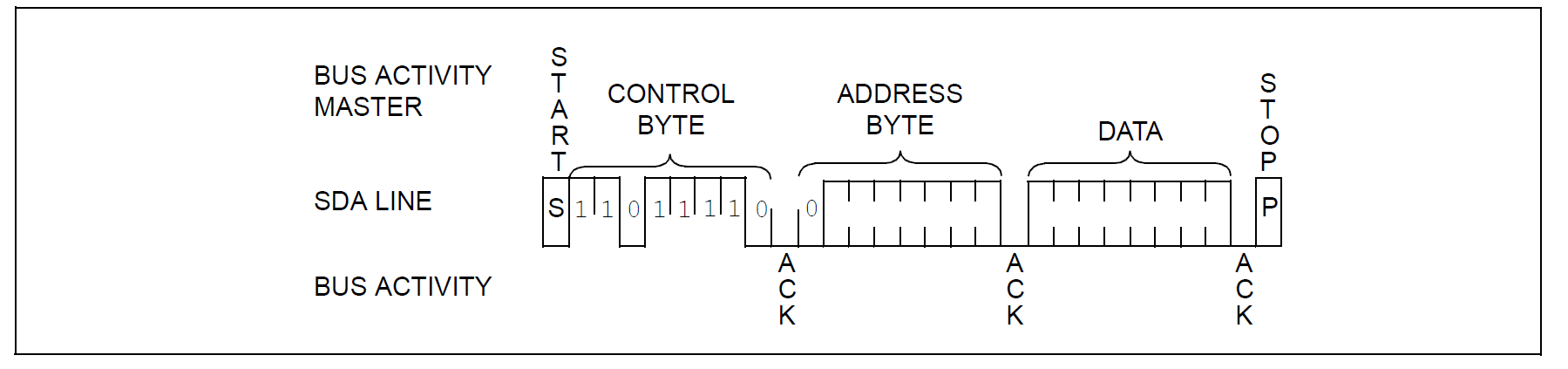


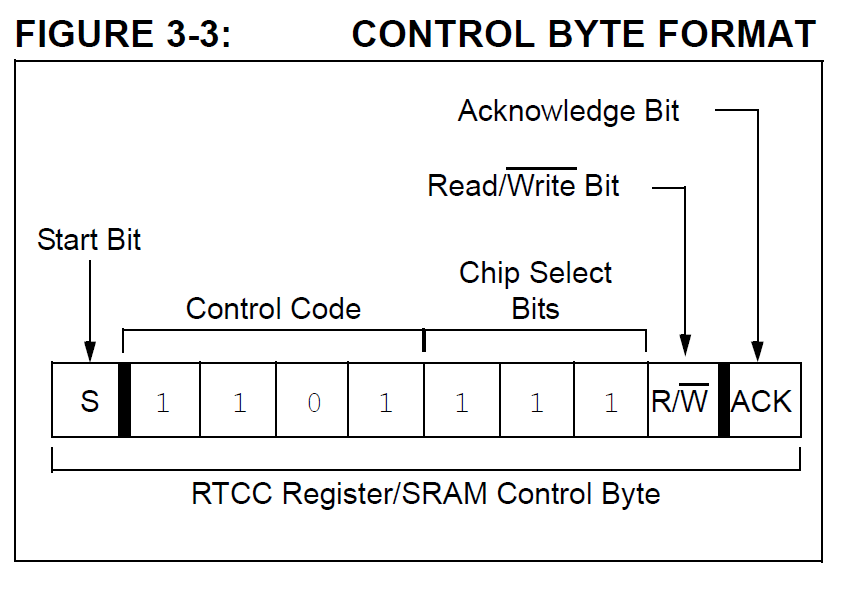
1. **FMS()** permet de convertir une valeur décimale de 0 à 9 et renvoyer un char de 0 à 9 affichable sur l'écran sans avoir à utiliser le code binaire correspondant au caractère

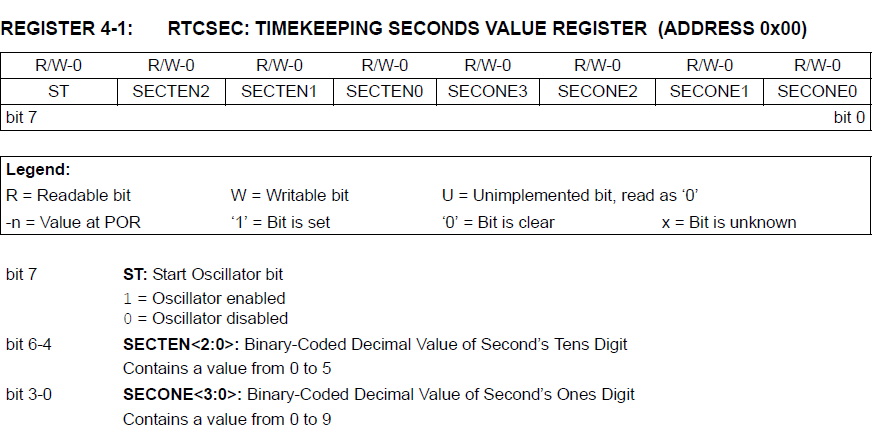




## Extrait de datasheet utile





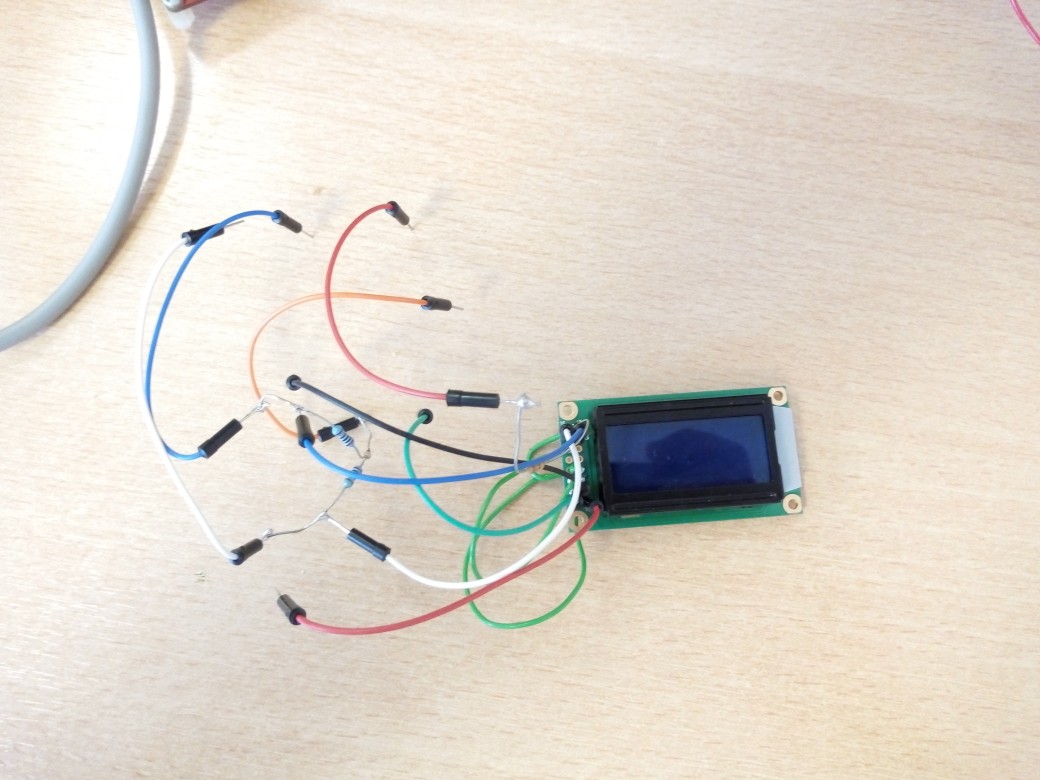


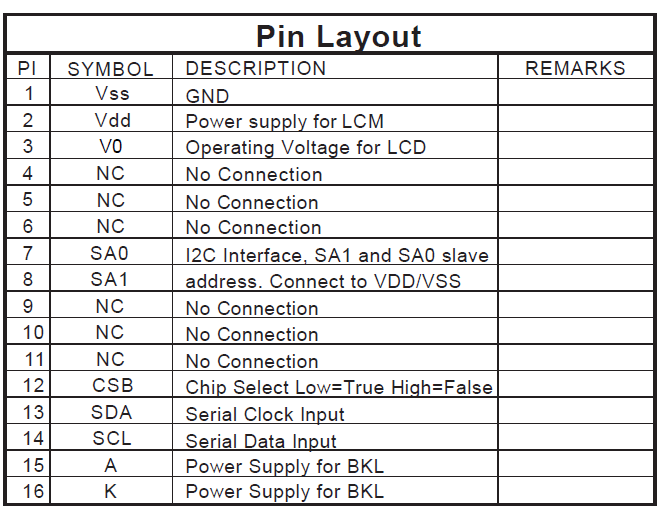
## Ecran LCD MIDAS

## Présentation

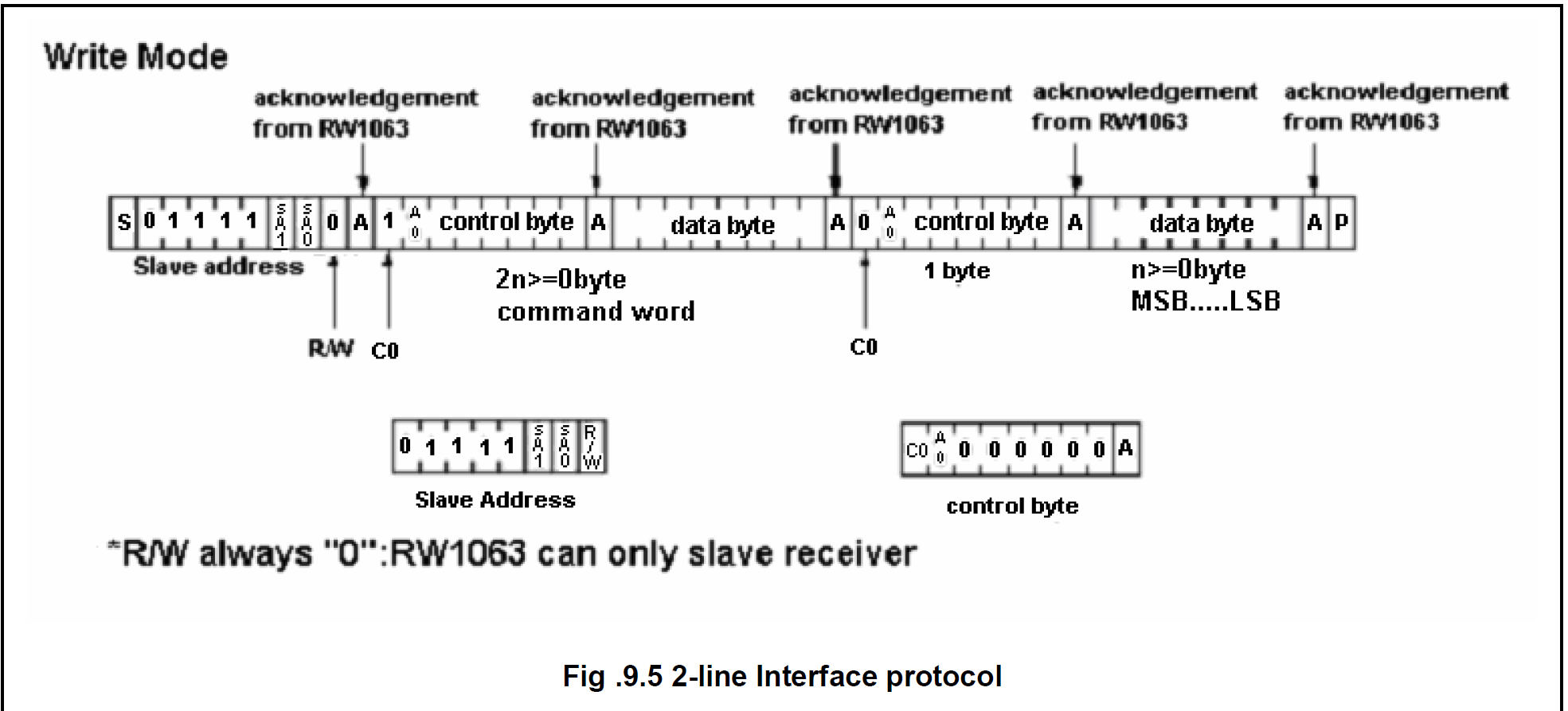
Notre écran LCD a pour but d’afficher une horloge temps réel. Pour ce faire, notre PIC, le module RTC ainsi que l’écran LCD dialoguent à l’aide d’une liaison I2C.

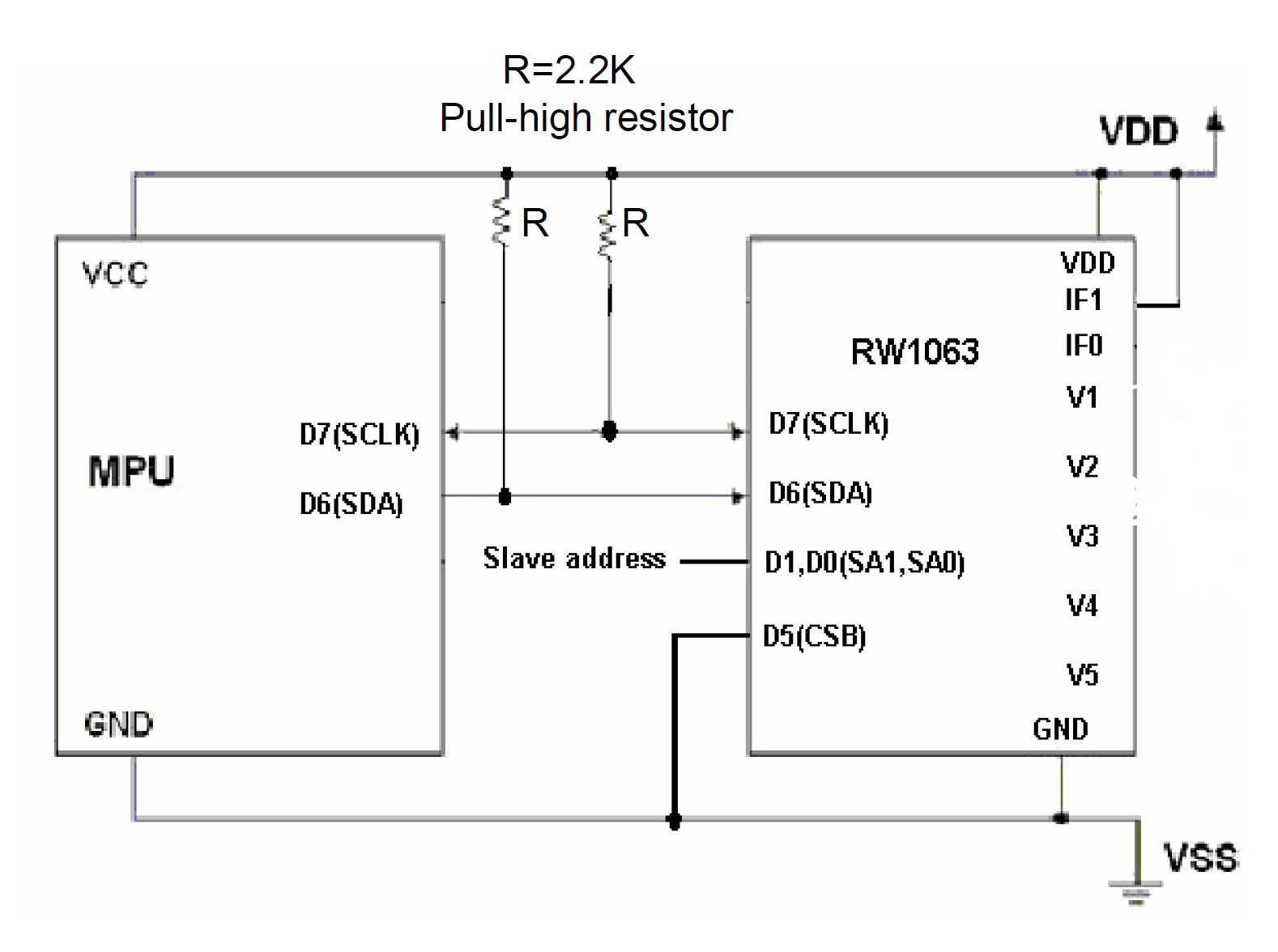
Tout d’abord, pour le montage de notre écran LCD nous avons procéder de la manière suivante :



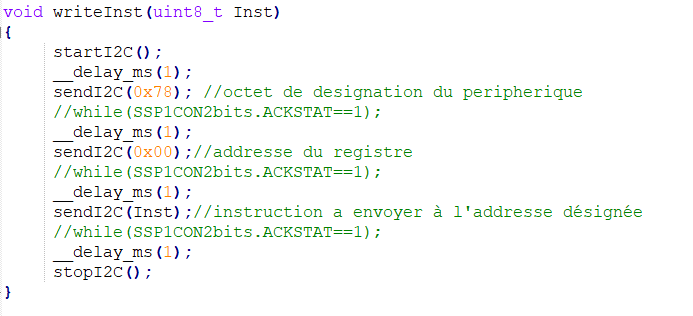
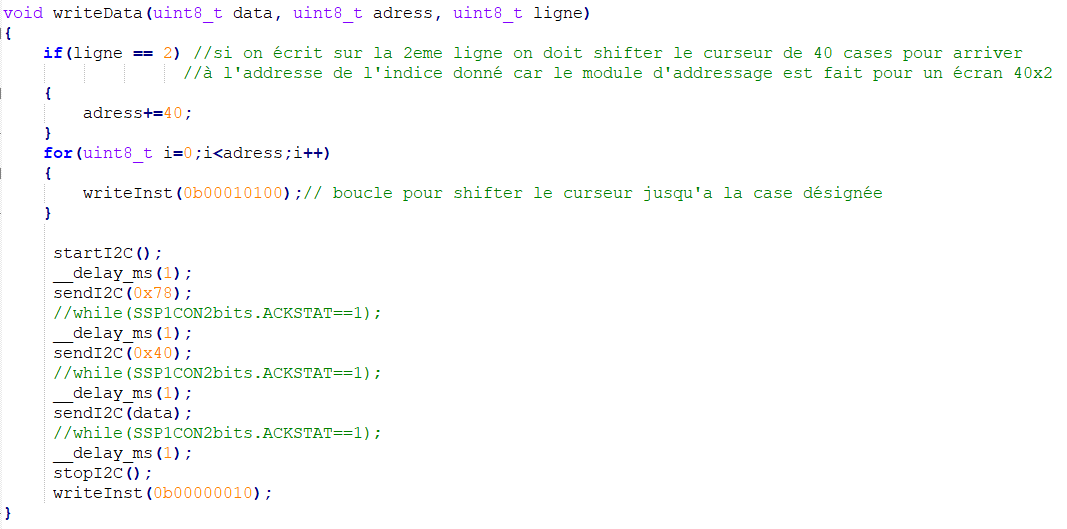






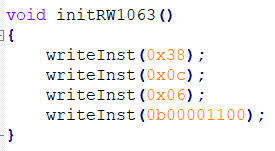


Programmation

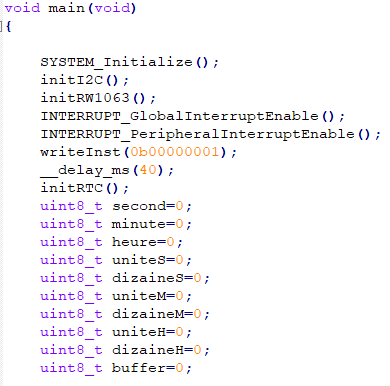
Cette fonction permet d'envoyer des Instructions à l'écran LCD tel que l'utilisation du curseur, du rétro-éclairage, allumer/éteindre l'écran

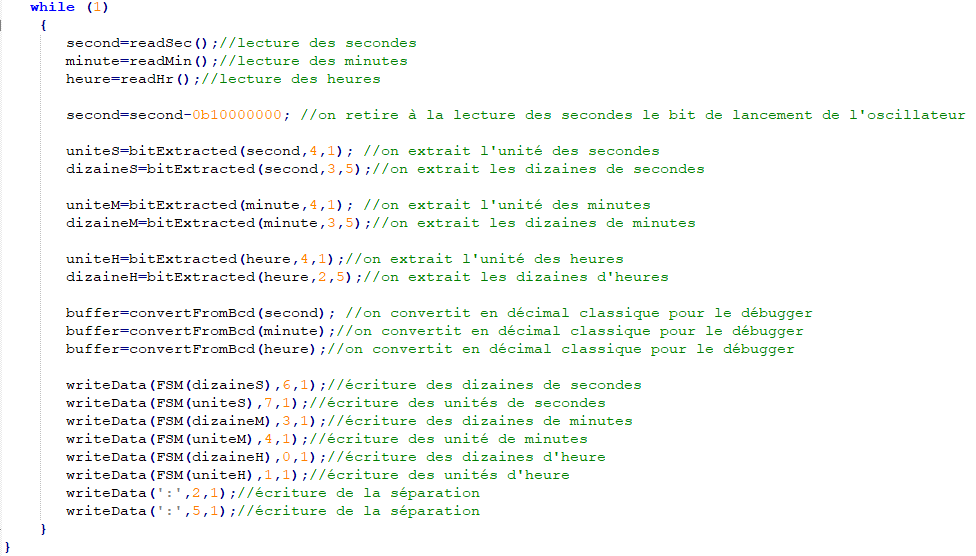
Celle-ci permet d'écrire un caractère ASCII a l'indice passé en paramètre et à la ligne passée en paramètre

Initialisation du module d'adressage interne de l'écran permettant de l'allumer, utiliser le rétro-éclairage…



Main





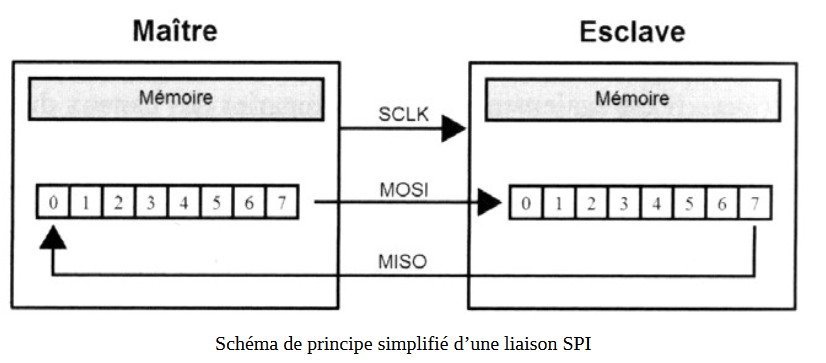
## Carte SD

## Présentation

Notre carte SD a pour but de contenir les valeurs des couleurs de rouge de vert et de bleu dans un fichier texte et ces dernières seront envoyées à notre EEPROM par Bluetooth pour éviter une latence programme lors de la lecture des couleurs, ce fichier pourra être rempli par l’utilisateur s’il le souhaite. Pour ce faire, notre PIC et notre carte SD dialoguent à l’aide d’une liaison SPI.

1. **Principe de la liaison SPI**

La figure ci-dessous correspond au schéma de principe simplifié d’une liaison de données SPI, avec ses principaux composants.



Les données échangées sont des octets. La transmission s’effectue sur 2 fils monodirectionnels

(nommés MOSI, MISO).

Une horloge indépendante fixée par le maître synchronise les échanges (en général sur front).

La fréquence de l’horloge de transmission est comprise entre 1 Mhz et 20 Mhz (selon les

performances des circuits reliés au bus).

Il n’y a pas d’adressage des esclaves (comme sur un bus i2C par exemple). L’esclave devient actif au moyen d’une ligne de sélection de boîtier dédiée (généralement active à l'état bas).

La ligne est constituée de 3 fils auxquels il faut ajouter les fils de sélection d'esclave.

* SCK :

Il s'agit de l'horloge de synchronisation des échanges, elle est générée par le maître et est

commune à tous les éléments du bus. Cela a pour avantage d’éviter que chaque composant

possède son propre quartz avec les problèmes de précision que cela impose.

* SDI, DI, SI, SDO, DO, SO :

Il s'agit de la patte de Serial Data In; dans ce cas de convention d'écriture il convient de

relier la SDI du maître à la SDO du ou des esclaves.

Il s'agit de la patte de Serial Data Out; dans ce cas de convention d'écriture il convient de

relier la SDO du maître à la SDI du ou des esclaves.

* nCS, CS ,nSS, STE :

Il s'agit de la patte /SS Slave Select générée par le maître



