République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abderrahmane MIRA- Bejaia

Faculté de Technologie

Département de Génie Electrique

Mémoire de fin de cycle

En vue de l'obtention du diplôme MASTER en Electrotechnique

Option : Automatisme industriel

Thème

Conception d'une Station Météo à Base de Microcontrôleur

Présenté par :

Mr BENMANSOUR Lahlou

Mr KHENTICHE Sofiane

Encadré par :

Mr TAIB.N

Mr METIDJI.B

Promotion 2011-2012

Remerciements

Qu'il nous soit d'abord permis de remercier et d'exprimer notre gratitude envers le Bon Dieu ,qui nous a donné la patience et le courage pour que nous puissions achever ce travail.

Nous adressons tous nos sincères et respectueux remerciements à nos promoteurs, Mr N. TAIB et Mr B. METIDJI pour leur dévouement et leur disponibilité durant la préparation de ce mémoire.

Nos remerciements s'adressent aux membres du jury d'avoir accepté d'évaluer notre travail.

Nous tenons également, à remercier tout le corps enseignant du département Génie électrique qui a contribué à notre formation.

Enfin, nous tenons à remercier, tous ceux qui ont contribué de prêt ou de loin pour l'élaboration de ce présent travail.

Dédicaces

Je dédie ce modeste travail:

A mes chères parents

A mes frères et sœurs

A tous mes amis (es) et toutes les personnes qui m'ont encouragé tout au long de mon cursus

A tous les êtres chères dont le soutient m'a été indispensable

Lahlou

Je dédie ce modeste travail:

A mes chères parents

A mes frères et sœurs

A tous mes amis (es) en particulier Lahlou et Halim

A toutes les personnes qui m'ont encouragé tout au long de mon cursus.

spécial es dédicaces pour le D313 , H06 et E405

A tous les êtres chères dont le soutient m'a été indispensable

Sofiane

Chapitre IV : Développement de l'application graphique sous DELPHI et tests.

Introduction générale

Notre tenue vestimentaire, nos déplacements, et un nombre de réflexes de notre vie quotidienne sont conditionnés par la connaissance du temps qu'il fera ou son estimation. Mais son véritable indispensabilité réside dans l'agriculture, le tourisme, la navigation surtout aérienne et maritime et la production d'électricité par des sources d'énergies renouvelables.

C'est pour cela que les scientifiques n'ont jamais cessé leurs recherches pour comprendre les lois qui régissent les phénomènes météorologiques. Et cela en perfectionnant les instruments de mesure, en exploitant au maximum l'avancée technologique qui leur apporte des progrès considérables. Car ils permettent d'effectuer des mesures avec une grande précision, tout en simplifiant les montages, pour ensuite les enregistrer dans le but d'exploiter ces données. Mais surtout exploiter ces phénomènes pour générer un éventuel profit où prévenir des dangers.

Notre projet est de concevoir une mini station météo piloté par un microcontrôleur (PIC16F876). Son rôle est de mesurer la température, l'humidité, la pression atmosphérique et la vitesse de vent, ces grandeurs vont être affichées sur un écran LCD, et au même temps seront transmises au PC via un port RS232.

Dans le premier chapitre, on a abordé les généralités sur les capteurs météorologiques, pour comprendre les types et les variantes de ces derniers pour choisir les plus adaptés.

Le deuxième chapitre traite les informations relatives au microcontrôleur (PIC16F876), et ses périphériques. On a pu apprendre à agir sur ses registres pour le configurer à notre volonté. Et aussi les méthodes qui vont nous permettre de configurer et d'afficher sur l'afficheur LCD.

Le troisième chapitre est consacré à la conception de la station météo sous le logiciel ISIS PROTEUS. Cela va nous permettre de voir le comportement des différents composants électronique utilisés, y compris le microcontrôleur PIC16F876 lors de la simulation. Où on a pouvoir faire des tests adéquats.

Dans le dernier chapitre qui est dédié à l'application de visualisation via le PC, on a d'abord présenté le principe de la programmation orientée objets. Puis on a décrit le langage de programmation qui nous a permis de la réaliser, qui n'est autre que DELPHI. Ensuite, on a expliqué les étapes importantes pour sa réalisation. Et enfin on a présenté son interface, et effectué des tests et des simulation, pour faciliter son exploitation à l'utilisateur final, et pouvoir apporter des améliorations souhaitées.

I.1 Introduction

Dans ce chapitre, nous avons d'abord cité l'intérêt d'une station météo, ses différents rôles et ses domaines d'exploitations. Puis on s'est focalisé sur les capteurs, car ils sont les éléments qui permettent la traduction des grandeurs physiques (température, humidité,…etc.) en grandeurs électriques mesurables (tension, courant, impédance, fréquence,…etc.).

I.2 Domaine d'exploitation

Les intérêts que rapporte la connaissance des facteurs climatiques grâce à la station météo sont divers, mais on s'intéresse à quelques domaines où son utilisation est plus répandue :

I.2.1 Dans le domaine des énergies renouvelables

a) Energie solaire

La station météo est utilisée pour choisir les endroits adéquats pour produire l'énergie solaire, que ce soit:

- ✓ Énergie solaire thermique, qui consiste à produire de la chaleur, par conversion de l'énergie contenue dans le rayonnement solaire.
- ✓ Énergie photovoltaïque, qui consiste à produire l'électricité à partir de la lumière à l'aide de panneaux solaires,
- ✓ Energie solaire thermodynamique ou thermosolaire, qui est la production de la vapeur à partir de la chaleur du soleil par concentration, puis conversion de la vapeur en électricité.
- ✓ Énergie solaire passive, utilisation directe de la lumière pour le chauffage.

Toutes ces méthodes varient en fonction de la météo où la température, la pression, l'humidité et autres facteurs climatiques sont déterminants pour exploiter au maximum la source.

b) Énergie du vent

L'énergie éolienne est l'énergie tirée du vent au moyen d'un dispositif aérogénérateur comme une éolienne ou un moulin à vent, Elle peut être utilisée de deux manières :

- ✓ Conservation de l'énergie mécanique : le vent est utilisé pour faire avancer un véhicule), pour pomper de l'eau, ou pour faire tourner la meule d'un moulin.
- ✓ Transformation en énergie électrique : l'éolienne est accouplée à un générateur électrique pour fabriquer un courant continu ou alternatif.

Cela rend indispensable le rôle de la station météo qui permet de connaitre la direction et la vitesse du vent et autres facteurs qui influence le rendement du dispositif.

c) Énergie de l'eau

- ✓ Énergie des vagues : utilise la puissance du mouvement des vagues.
- ✓ Énergie marémotrice : issue du mouvement de l'eau créé par les marées (variations du niveau de la mer, courants de marée).
- ✓ Energie hydrolienne : Les hydroliennes utilisent les courants sous marins.

- ✓ Énergie maréthermique : produite en exploitant la différence de température entre les eaux superficielles et les eaux profondes des océans.

Pour exploiter cette énergie, nous avons intérêt à connaitre les facteurs climatiques des océans tels que la pression, direction et vitesse du vent et la température.

I.2.2 Dans le domaine des transports

Dans les aéroports, la pression, l'humidité, la direction et la vitesse du vent sont primordiale pour la sécurité des vols, les stations météo sont embarquées dans les avions pour qu'ils puissent connaitre les changements climatique à temps réel. Des vents très intenses provoquent parfois l'annulation des décollages ou des atterrissages car ils perturbent les processus. Aussi dans le navigation maritime, il provoque l'agitation de la mer ce qui compromet la sécurité des navires et des personnes à bord, c'est pour ces raisons qu'ils intègrent aussi des stations météo dans ces derniers.

I.2.3 Dans la vie quotidienne

Connaitre la pression atmosphérique et l'humidité relative sont des éléments essentiels pour l'être humain, car le fonctionnement de quelques organes et le corps en général est influencé par celles-ci. Dans les altitudes élevées, la pression est basse, ce qui provoque un comportement inhabituel pour le corps.

I.2.4 Le rôle préventif de la station météo

La station météo joue aussi un rôle pour la prévention des dangers potentiel de la nature, que ce soit les tempêtes, les cyclones ou autres dangers qui résulte du vent, on peut les éviter en prenons les précautions nécessaires, la température peut aussi engendrer des menaces telle que la canicule, mais une fois tous les facteurs climatiques connus grâce à la station météo, on peut empêcher des catastrophes ou du moins réduire leurs impactes.

I.3 Constitution de la station météo

I.3.1 Capteurs de températures

La température est l'une des grandeurs physiques dont la mesure est la plus fréquente. Ce n'est pas une grandeur directement mesurable (comme la longueur) mais une grandeur repérable à l'aide d'un phénomène associé tel que la variation de la résistance d'un conducteur électrique, la dilatation d'un fluide ou l'émission d'un rayonnement thermique.

Avant de choisir une méthode de mesure, il convient de bien connaître la nature du milieu. On distingue plusieurs capteurs :

- ✓ Capteurs mécaniques.
- ✓ Capteurs optiques.
- ✓ Capteurs électriques.

Mais nous on s'intéresse à des capteurs électriques.

I.3.1.1 Capteurs électriques

Les capteurs électriques qui suivent auront l'avantage d'une plus grande souplesse d'emploi (information transmissible, enregistrement) tout en gardant une précision suffisante pour les emplois industriels et beaucoup d'emplois de laboratoire.

a) Capteurs par résistances

La thermométrie par résistance utilise, comme son nom l'indique, la variation de la résistance d'un matériau en fonction de la température. Cette variation de résistance peut être faite aussi bien avec un métal (dans ce cas-là nous parlerons de résistance métallique) mais aussi avec des oxydes (dans ce cas-là nous parlerons de thermistances) [1].

b) Capteurs par thermocouples

Les thermocouples sont des capteurs actifs qui délivrent une f.é.m lorsque ceux-ci sont soumis à une modification de la température. Le principe de fonctionnement est basé sur l'effet Seebeck qui lorsque deux conducteurs métalliques sont reliés par deux jonctions soumises à des températures différentes, crée une différence de potentiel aux bornes du circuit. Une fois que nous avons la f.é.m, pour pouvoir remonter à la valeur de la température, il faut connaître l'une des deux jonctions et surtout sa température. Celle-ci se nommera jonction de référence. La nature des matériaux conducteurs utilisés définit le type du thermocouple. Il existe beaucoup de type des thermocouples qui sont pour la plupart repérés par une lettre[1].

c) Capteurs par diodes et transistors

La diode ou le transistor au silicium (base et collecteur reliés comme le montre la figure I.1) sont alimentés dans le sens direct à courant constant. La tension à leurs bornes est fonction de la température peut donc être la grandeur électrique de sortie du capteur de température qu'ils constituent.

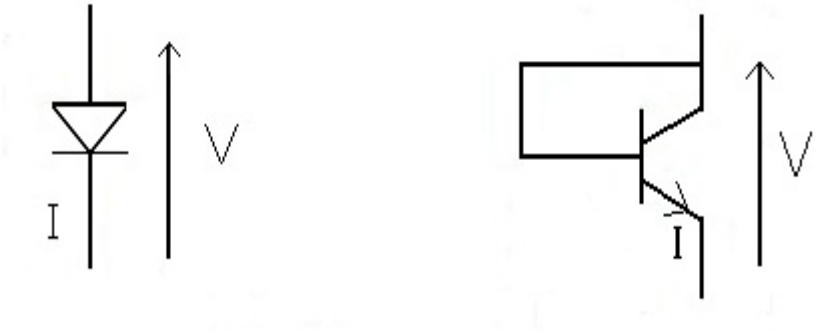

Figure I.1 Capteur de température par diode et transistor.

Relation tension-température

Lorsqu'on met la diode ou le transistor (base et collecteur reliés) sous tension, ou le courant traverse dans le sens passant on obtient l'expression de tension aux bornes de diode :

$$\mathbf{V_d = \frac{k.T}{q}.Ln(\frac{I}{I_0})} \qquad \mathbf{I.1}$$

I : courant direct parcourant la diode ou le transistor.

I_0 : courant inverse.
k : constante de Boltzman.
q : charge de l'électron.
T : température en Kelvin.
Le courant I_0 dépend de température (T), alors il faut l'éliminer en utilisant deux transistor identiques et un amplificateur différentiel[1].

I.3.2 Capteurs de pressions

I.3.2.1 Capteurs piézo-électriques

Les structures piézoélectriques utilisées comme corps d'épreuve assurent directement la transformation de la contrainte, produite par l'application d'une force <F>, en un signal électrique. Ainsi, une lame, découpée dans cristal de quartz, perpendiculairement à l'un de ses trois axes électriques, munie d'armatures métalliques, développe par compression une polarisation diélectrique se traduisant par l'apparition sur les armatures d'une charge Q :

$$Q=k.F \qquad \text{I.2}$$

Comme la pression P est : P=F/S donc

$$Q=k.S.P \qquad \text{I.3}$$

Q : charge électrique.
k : la constante piézoélectrique.
Dans le cas du quartz, $k=2{,}32.10^{-12}$ coulomb/newton
F : la force appliquée.
S : la surface.
P : la pression [3].

I.3.2.2 Capteurs à l'effet capacitifs

Le capteur de pression à effet capacitif consiste à transformer la déformation de la membrane sous l'effet d'une pression en une variation de capacité. En effet, il suffit de placer l'une des armatures d'un condensateur sur la membrane qui se déforme et l'autre sur une pièce solidaire du corps d'épreuve, mais non soumise à la déformation[3].

I.3.2.3 Capteurs à l'effet inductifs

Ce capteur comporte deux bobinages symétriques par rapport à la position de référence de l'armature mobile. Cette armature sera déplacée par l'action d'un piston, solidaire de la membrane de capteur, ainsi chacune des inductances variant en sens inverse [3].

I.3.2.4 Capteurs piézo-résistifs

Les deux paramètres importants qui caractérisent la jauge de contrainte sont, d'une part, la valeur de la résistance au repos et, d'autre part, la sensibilité de la jauge aux contraintes. Les principaux facteurs qui conditionnent les valeurs de ces paramètres sont :

- ✓ le matériau constituant la jauge ainsi que son niveau de dopage.
- ✓ les dimensions de la jauge.

- ✓ les dimensions du corps d'épreuve.

Les jauges de valeurs nominales voisines de quelques kilo-ohms sont élaborées en silicium monocristallin ou poly cristallin de type P dont le dopage est compris entre 10^{18} et quelques 10^{19} at/cm^3. Leurs longueurs sont comprises entre 50 mm et quelques centaines de micromètres et leur largeur entre 10 mm et quelques dizaines de micromètres. La variation maximale de résistance est inférieure à 5 % afin de minimiser la non-linéarité. Celle-ci est alors inférieure à quelques dixièmes de pour-cent de la réponse pleine échelle.

Un inconvénient majeur des jauges de contraintes est la forte dérive thermique de la résistivité qui peut être du même ordre de grandeur que l'amplitude de la réponse en pression. C'est pourquoi les capteurs à jauges de contraintes sont le plus souvent réalisés avec quatre résistances montées en pont de Wheatstone telles qu'elles sont représentées dans la figure I.2 afin de rejeter le mode commun.

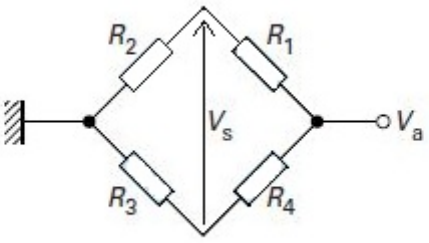

Figure I.2 Montage de Capteur piézo-résistif en pont de Wheatstone.

La réponse du capteur est alors donnée par:

$$\frac{\Delta V_s}{Va} \approx \frac{1}{4}\left(\frac{\Delta R2}{R2} + \frac{\Delta R4}{R4} - \frac{\Delta R1}{R1} - \frac{\Delta R3}{R3}\right) \qquad \textbf{I.4}$$

Avec : Va la tension d'alimentation, Vs la tension de sortie du capteur.

La sensibilité à la pression du capteur dépend de la position des différentes jauges sur le corps d'épreuve. La configuration est généralement choisie de manière à obtenir des variations des résistances R2 et R4 d'une part et R1 et R3 d'autre part de signes opposés afin d'optimiser la sensibilité. Les principales configurations utilisées sont représentées sur la figure I.3 .

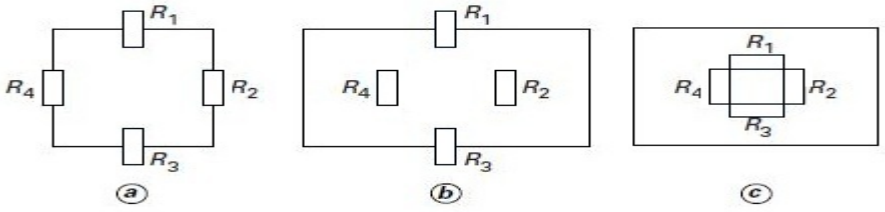

Figure I.3 Différents montages des résistances des capteurs piézo-résistifs.

La première (figure I.3a) est la plus couramment utilisée car, à surface égale de silicium, elle présente la meilleure sensibilité à la pression.

Quelle que soit la configuration utilisée, l'inconvénient majeur des capteurs de pression piézo-résistifs reste la dérive thermique de la sensibilité à la pression. Celle-ci est donnée par la dérive thermique des coefficients de piézo-résistivité et est comprise entre $-2\,500\times10^{-6}$/ °C et $-1\,000\times10^{-6}$/ °C pour des dopages de bore de 10^{18} à quelques 10^{19} at/cm^3. Ces capteurs doivent donc être compensés en température à l'aide de circuits électroniques correcteurs plus ou moins sophistiqués en fonction du degré de correction souhaité lorsque la plage d'utilisation en température et/ou la précision du capteur le nécessite. Une technique de mesure permet de minimiser la dérive thermique de la sensibilité à la pression. Elle consiste à remplacer la source de tension Va par une source de courant I_a. La dérive thermique positive de la résistance peut alors compenser plus ou moins la dérive négative du coefficient de piézo-résistivité en fonction du dopage et du matériau utilisés pour réaliser la jauge [3].

I.3.3 Capteurs de vitesse du vent

I.3.3.1 Girouettes

Elles comprennent :

- ✓ une pièce appelée *drapeau*, mobile autour d'un axe vertical et qui s'oriente toujours dans le lit du vent.
- ✓ un dispositif de recopie à distance de l'angle formé par le drapeau et la direction du Nord géographique.

La forme et les caractéristiques géométriques du capteur primaire conditionnent les performances de la girouette.
Le transmetteur peut être un codeur angulaire solidaire de l'axe du capteur, l'information est alors fournie directement sous forme numérique. On trouve aussi des systèmes de recopie angulaire par synchro-moteur. Ce procédé est surtout employé dans les girouettes installées sur les navires [3].

I.3.3.2 Anémomètres

a) Anémomètres statiques

a.1) Anémomètres à tube de Pitot

Ces appareils fonctionnent suivant le principe du tube de Pitot, avec la mesure de la surpression due au vent, dans la direction face au vent. Ils sont peu sensibles aux basses vitesses (au-dessous de 5 m · s−1) mais sont quelquefois employés dans les zones à fort givrage, leur protection contre des dépôts de givre par chauffage étant relativement aisée.

a.2) Anémomètres à fil chaud

Ils sont fondés sur la loi de King, qui établit une relation semi-empirique entre le flux de chaleur perdu par un fil chauffé et la vitesse de circulation de l'air dans lequel il est plongé. L'influence de la température sur la réponse de ces capteurs est très importante. On la compense par des circuits électroniques.

L'intérêt de ces anémomètres réside surtout dans leur très large domaine de mesure (de quelques centimètres par seconde à 100 m · s−1) et dans leur très faible inertie (réponse inférieure à 1 s).

a.3) Anémomètres à ultrasons

Ces appareils utilisent le principe de mesure de la vitesse de propagation du son dans l'air. Cette vitesse dépend de la température de l'air et de la composante longitudinale du vent dans la direction considérée.

Des couples d'émetteurs/récepteurs sont placés dans des directions orthogonales et mesurent chacun une composante du vent.

Alternativement, l'onde sonore est envoyée dans un sens et en sens opposé, en inversant l'émetteur et le récepteur. Cela permet, par soustraction, de s'affranchir de l'influence de la température sur la vitesse du son. L'électronique de calcul est associée directement au niveau du capteur qui fournit une sortie numérique. La cadence de mesure peut être très rapide, jusqu'à dix mesures par seconde. Leur seuil de détection est très faible, de l'ordre de quelques centimètres par seconde [3].

b) Anémomètres à moulinet ou à hélice

Le capteur primaire (moulinet ou hélice), dont la vitesse de rotation est fonction de la vitesse du vent, entraîne un disque crénelé. Un détecteur optique ou magnétique compte le nombre d'impulsions générées par le disque en rotation et mesure le nombre de tours effectués en un temps donné. La figure I.4 présente une illustration d'un modèle réel.

Figure I.4 Anémomètre à moulinet.

Les hélices à axe horizontal doivent être perpendiculaires à la direction du vent, aussi sont-elles toujours associées à un support orientable ; elles ont trois ou quatre pales.
Les appareils à hélice mesurent donc aussi la direction du vent. Ils sont moins précis que les appareils à moulinet.

La constante de temps varie en fonction inverse de la vitesse du vent. On définit plutôt la constante de distance qui est le produit de la constante de temps par la vitesse du vent et est telle que l'instrument indique environ 63 % de la variation à un échelon de la vitesse du vent.

La sortie électrique des anémomètres est généralement une fréquence, parfois une tension électrique.

Il fournit des impulsions générées à chaque passage du trou de disque du dispositif mécanique devant la fourche optique du support, ce qui génère des impulsions proportionnel à la vitesse du vent, celle-ci sont directement exploitées grâce à la loi :

$$V=R.\Omega \qquad \text{I.5}$$

Tel que : V : la vitesse du vent en km/h
R : rayon de l'axe d'anémomètre au centre de la coupelle en Mètre.
Ω : fréquence de rotation en Rad/s [3].

I.3.4 Capteurs d'humidité

I.3.4.1 Psychromètre

Un psychromètre est un instrument de mesure destiné à connaître des caractéristiques énergétiques de l'air humide.

Il est constitué de deux thermomètres mesurant au même moment et au même endroit la température de l'air (dite température sèche) et sa température humide.

La différence entre ces deux températures données par le psychromètre permet d'accéder à l'ensemble des données énergétiques de l'air humide et en particulier son humidité relative. L'utilisation d'un abaque permet de connaître l'humidité relative. La différence de température peut atteindre plusieurs degrés Celsius [1].

I.3.4.2 Hygromètre à cheveu

L'hygromètre le plus simple est l'hygromètre à cheveu, qui utilise la propriété du crin de cheval ou du cheveu humain de s'allonger ou se raccourcir lorsque l'hygrométrie varie. L'allongement du cheveu est de l'ordre de 2 % lorsque l'humidité (relative) varie de 0 à 100 %.mais il est peu fiable étant donné qu'il est aussi fortement sensible à la température [1].

I.3.4.3 Capteurs à impédances variables

a) Hygromètre résistif

Pour mesurer l'humidité de l'air, on emploie généralement des résistances au chlorure de lithium. Ce matériau hygroscopique possède une grande résistance lorsqu'il est sec et une faible résistance lorsqu'il est humide. Comme ce type de capteur permet de mesuré l'humidité que pour une faible plage, il faut raccorder plusieurs cellules qui on une sensibilité différente. Comme la résistance varie avec la température, il faut employer une thermistance pour ne pas fausser les résultats. On obtient alors une plage qui varie de 5% à 95% avec une précision de ± 5% [1].

b) Hygromètre capacitif

On mesure la capacité d'un condensateur dont le diélectrique est hydrophile. Pour mesurer l'humidité de l'air, on utilise généralement l'Oxyde d'aluminium comme diélectrique. Le condensateur doit avoir une armature poreuse pour facilité le passage de l'air dans le diélectrique comme le montre la figure I.5. Un pont de Sauty relie les différentes cellules. Le pont est ensuite alimenté par un courant alternatif de haute fréquence (une fréquence élevée

favorise une plus faible consommation d'énergie). La tension ainsi généré nous indique le pourcentage d'humidité. Cette technique de mesure offre des performances correctes (±3% d'erreur) pour une gamme variant entre 5% à 99% d'humidité relative [3].

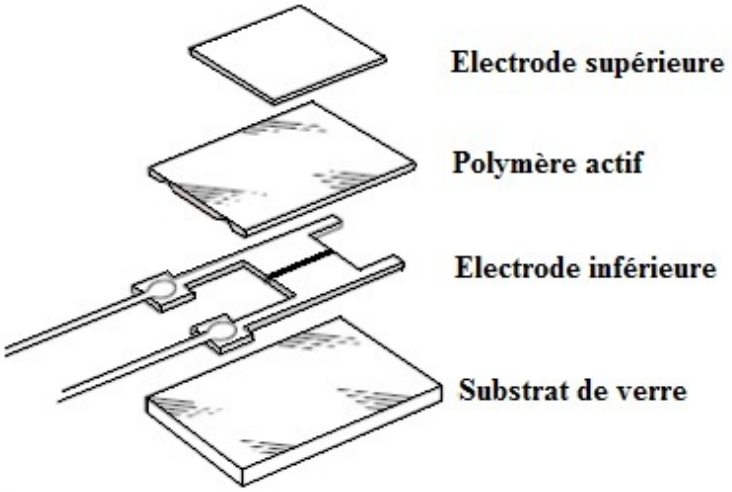

Figure I.5 Hygromètre capacitif.

I.4. Conditionnement des capteurs

I.4.1. Capteur température

Le capteur de température est un circuit intégré LM335 qui comporte 3 broches et délivre une tension de sortie en relation linéaire avec la température exprimée en Kelvin, la figure I.6 montre le modèle réel.

Pour notre station météo il faut choisir :

- ✓ La plage de mesure comprise entre valeur min et max.
- ✓ Une résolution de 0.1 °C par un quantum pour un convertisseur.
- ✓ Le capteur doit subir un traitement puis une amplification pour obtenir une variation pleine échelle entre 0 et 5 volts a l'entrée de convertisseur [2].

Pour une mesure en °C il faut une conversion de K à °C, et pour cela on est obligé de soustraire la tension de capteur et celle de l'Offset qui est d'ordre 2.2315 V (tension de sortie de capteur à -50°C ou 223 K). Et à la fin tension obtenue sera amplifié pour une variation pleine échelle [6].

Alors la relation entre tension et température est :

$$V=A.k.(T-273,15) \qquad \text{I.6}$$

A : Gain d'amplification total.
K : Sensibilité 10 mV/°C.
T : Température mesurée en °C.

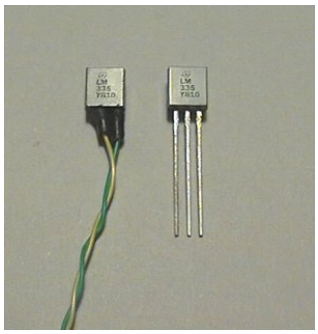

Figure I.6 Capteur de température LM335.

I.4.2. Capteur de vitesse du vent

Le capteur fournit des impulsions générées à chaque passage du trou de disque du dispositif mécanique devant la fourche optique du support (plus le disque tourne vite, plus les impulsions sont rapprochées et la fréquence est élevée). Ces impulsions sont dirigées à l'entrée de timer0 du microcontrôleur. Le modèle réel est présenté dans la figure I.7 [6].

Figure I.7 capteur Girouette et anémomètre.

I.4.3. Capteur de pression

La pression est mesurée par un capteur MPX 4115 illustré dans la figure I.8, ce dernier nous permet de choisir :

- ✓ La plage de mesure comprise entre valeur min et max.
- ✓ Une résolution de 0.15 hPa par un quantum pour un convertisseur.

Le capteur délivre une tension de sortie qui varie donc entre 0 et 5 volts [2].

$$V=f(p) \qquad \text{I.7}$$

Figure I.8 Capteur de pression MPX4115.

I.4.4 Capteur d'humidité

La technique est basée sur la mesure de la capacité de capteur par un capacimètre. Ce dernier n'est qu'un monostable de type NE555 qui délivre en sortie des impulsions d'une durée T dépendant de la valeur de capacité du capteur donc de la valeur de l'humidité relative[6].

$$T=f(C(H)) \qquad \text{I.8}$$

I.5 Présentation de la station météo

Schéma synoptique

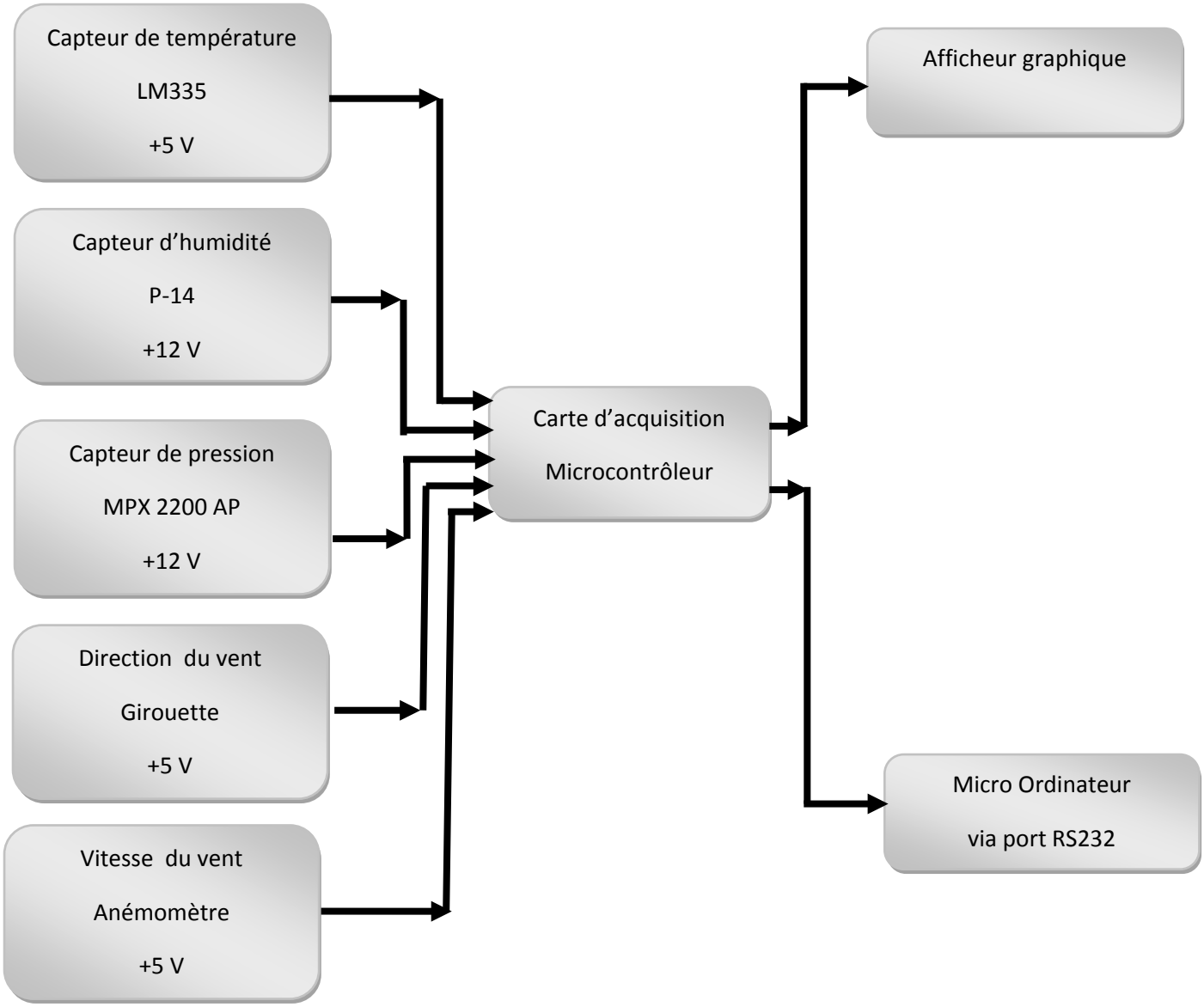

Figure I.10 Schéma synoptique de la station météo.

I.6 Conclusion

La station météo comme nous l'avons vu précédemment, présente un grand intérêt dans différents domaines, et l'avancée technologique qu'apporte notamment l'électronique et l'informatique permettent de la perfectionner, en la rendant moins encombrante, grâce aux différents composants de plus en plus miniatures, plus précise et autonome. Notre projet consiste à concevoir une station météo à base de microcontrôleur, ce dernier va gérer plusieurs composants électroniques (capteurs) où chaque composant doit transmettre une information (signal électrique) tel que la température, la pression atmosphérique, l'humidité et la vitesse du vent. Un programme écrit en langage C puis compilé permet d'interroger chaque composant pour acquérir ses données et de les transmettre vers l'afficheur LCD et le PC.

II.1 Introduction

Outre les capteurs présentés précédemment, notre station météo contient d'autres éléments essentiels pour fonctionner. Dans ce chapitre, on s'intéresse particulièrement au microcontrôleur et ses périphériques internes, car il est la base de notre station météo, et cela en énumérant les éléments les plus importants d'entres-eux, particulièrement ceux qu'on a utilisé tel que les Timers, ADC et USART...etc.

Une fois le microcontrôleur détaillé, l'autre élément essentiel est l'afficheur LCD, où on se base sur les différents modes et méthodes de configurations de ce dernier pour mieux l'utiliser.

II.2 Le Microcontrôleur

C'est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires, périphériques et interfaces d'entrées-sorties pour communiquer avec le monde extérieur. Les microcontrôleurs se caractérisent par un plus haut degré d'intégration comme le montre la figure II.1, une plus faible consommation électrique, une vitesse de fonctionnement plus faible et un coût réduit par rapport aux microprocesseurs utilisés dans les Ordinateurs personnels. Ils permettent de diminuer la taille, la consommation électrique et le coût des produits. Ils ont ainsi permis de démocratiser l'utilisation de l'informatique dans un grand nombre de produits et de procédés [4].

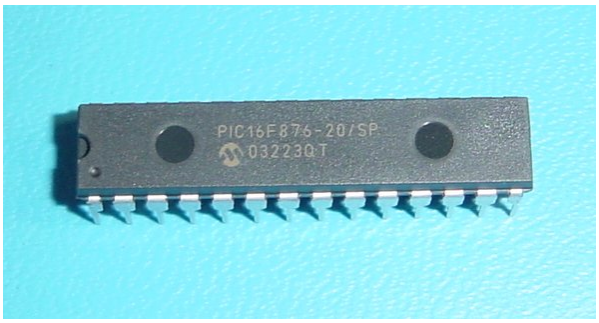

Figure II.1 Microcontrôleur PIC16F876.

Ils sont fréquemment utilisés dans les systèmes embarqués, comme les contrôleurs des moteurs automobiles, les télécommandes, l'électroménager, la téléphonie mobile, etc. Ces systèmes se démarquent selon plusieurs aspects :

- Ils sont soumis à des contraintes de taille, de consommation électrique et de coût importantes.
- La taille des programmes et la quantité de mémoire (Mémoire vive et Mémoire morte), dont ils disposent, sont modestes.
- Ils doivent communiquer avec des dispositifs d'entrées-sorties
- Ils n'ont parfois aucun dispositif d'interface homme-machine.

Un microcontrôleur est donc un composant autonome, capable d'exécuter le programme contenu dans sa mémoire morte dès qu'il est mis sous tension. Selon les modèles et les conditions de fonctionnement, les microcontrôleurs peuvent avoir besoin de quelques composants externes (quartz, quelques condensateurs, parfois une ROM) [4].

II.2.1 Composants intégrés

Un microcontrôleur intègre:

- un processeur (CPU), avec une largeur du chemin de données allant de 4 bits pour les modèles les plus basiques à 32 ou 64 bits pour les modèles les plus évolués.
- de la mémoire vive (RAM) pour stocker les données et variables.
- de la mémoire morte (ROM) pour stocker le programme. Différentes technologies peuvent être employées : EPROM, EEPROM, mémoire flash, etc.
- un oscillateur pour le cadencement. Il peut être réalisé avec un quartz, ou un circuit RC.
- les convertisseurs analogiques-numériques (CAN).
- les convertisseurs numériques-analogiques (CNA).
- les générateurs de signaux à modulation de largeur d'impulsion (MLI).
- les Timers/compteurs (compteurs d'impulsions d'horloge interne ou d'événements externes).
- les chiens de garde (watchdog).
- les comparateurs (comparent deux tensions électriques),
- les contrôleurs de bus de communication (UART, I²C, Interface de communication série, CAN, FlexRay, USB, Ethernet, etc.).

Certains microcontrôleurs ont un nombre très restreint de broches, si bien qu'une broche donnée peut correspondre à plusieurs périphériques internes. La fonction choisie doit alors être sélectionnée par logiciel.

Le choix des périphériques à intégrer dans un microcontrôleur est délicat. Car il y a un compromis entre des besoins contradictoires : utiliser des fréquences élevées, réduire la taille du circuit, apporter des fonctionnalités nombreuses, élaborer une architecture flexible, assurer des coûts modérés, etc. [8].

II.2.1.1 Le processeur (CPU)

Son rôle est de rechercher les instructions qui sont stockées en mémoire, pour les décoder et ensuite les exécuter. Il est composé de plusieurs éléments :

- L'unité arithmétique et logique (UAL) : chargée des calculs, comme l'addition, la soustraction, la multiplication et la division, et les opérations logiques (AND, OR, NOT...)
- l'unité de commande chargée de traduire puis d'exécuter les commandes et cela en allant chercher une information en mémoire centrale, puis d'analyser cette instruction (décodage), et de l'exécuter, pour enfin localiser l'instruction suivante.
- Un décodeur d'instruction
- Un séquenceur et des circuits de commande

Un ensemble de circuits électronique commandés par l'unité de contrôle permettent:

- D'échanger des informations avec la mémoire centrale et avec le monde extérieur (les périphériques)
- De mémoriser l'adresse de la prochaine instruction dans un registre particulier PC (program counter))
- De mémoriser le résultat d'opérations dans des mémoires spéciales : les Registres de travail [8].

a) Les registres CPU

Cellules mémoire interne au processeur (rapide) :

- **Compteur ordinal (PC)** pointant à l'adresse mémoire où se trouve la prochaine instruction à rechercher et exécuter. Après chaque recherche d'instruction, le compteur ordinal est incrémenté afin de pointer à la prochaine instruction (en fait après le chargement de l'octet ou du mot mémoire faisant partie d'une instruction, de manière à pointer sur un code d'instruction.
- **Registre d'instruction (I)** qui contient le code de l'instruction (code opératoire) recherchée en mémoire.
- **Les registres de données (X, Y, D0, Dx,..)** permettent de stocker les opérandes nécessaires aux instructions de calcul ainsi que les résultats lors d'opérations logiques et arithmétiques.
- **Les registres d'adresses (A, A0, Ax,..)** permettent de stocker les adresses d'opérandes qui se trouvent en mémoire [2].

b) Architecture des microprocesseurs

Un microcontrôleur peut effectuer la plupart des instructions machine habituelles, avec certaines restrictions liées à son caractère embarqué. On note cependant quelques particularités.

Les capacités mathématiques sont en général particulièrement limitées, réduites à des additions, soustractions et décalages sur des octets pour les plus simples d'entre eux. Les calculs mathématiques évolués doivent donc être ramenés à une succession d'opérations simples portant seulement sur des octets. Des routines mathématiques (petits programmes permettant de réaliser les calculs complexes) ont été développées pour la plupart des microcontrôleurs populaires [4].

Les instructions d'entrée-sortie sont bien développées, de façon à pouvoir :

- lire l'état d'un port d'entrée ;
- écrire une information dans le registre d'un port de sortie, qui maintient l'information à la disposition des circuits extérieurs.

Les microcontrôleurs disposent généralement de nombreuses instructions dédiées aux opérations sur les bits, de façon à rendre les programmes plus compacts, notamment lorsqu'ils agissent sur une entrée-sortie donnée. Ainsi, un processeur généraliste peut avoir besoin de "plusieurs" instructions pour tester la valeur d'un bit d'un registre et effectuer un saut si le bit vaut vrai. Cette fonction courante est assurée par "une seule" instruction dans certains microcontrôleurs.

Les modes d'adressage utilisables sont en général semblables à ceux des autres processeurs.

De façon générale, les instructions et modes d'adressage disponibles varient fort selon les familles de microcontrôleurs.

La majorité des microcontrôleurs actuels utilise une architecture interne dite de Von Neumann, qui est une architecture commune à celle des micro-ordinateurs elle est représentée par la figure II.2. La mémoire contient des instructions et des données placées les unes à la

suite des autres, et l'on ne dispose que d'un bus, appelé bus de données, pour véhiculer tour à tour les codes des instructions et les données qui leur sont associées, comme le montre la figure suivante [4].

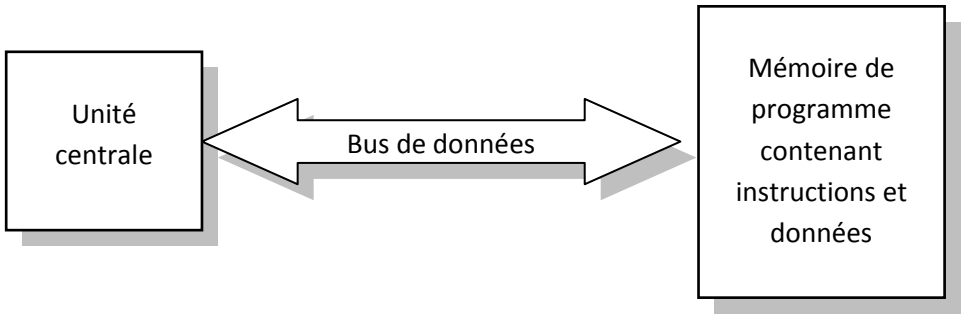

Figure II.2 : Synoptique de l'architecture de Van Neumann.

L'inconvénient de cette architecture est que l'exécution d'une seule instruction nécessite plusieurs échanges de données sur le seul et unique bus dévolu à cet usage puisqu'il faut d'abord aller chercher le code de l'instruction, puis le ou les données qu'elle doit manipuler.

Certains microcontrôleurs suivent une architecture Harvard dans laquelle les instructions et les données sont clairement différenciées et son véhiculées sur des bus différents comme l'illustre la figure II.3 , ce qui permet aux accès d'avoir lieu en même temps (on parle d'accès concurrent). Lorsqu'on utilise une architecture Harvard, les résultats obtenus en terme de vitesse d'exécution des programmes est impressionnant, car l'exécution d'une instruction ne fait plus appel qu'à un seul cycle de machine, puisqu'on peut simultanément grâce aux deux bus, rechercher le code de l'instruction et la ou les données qu'elle manipule, comme le montre la figure suivante [4]

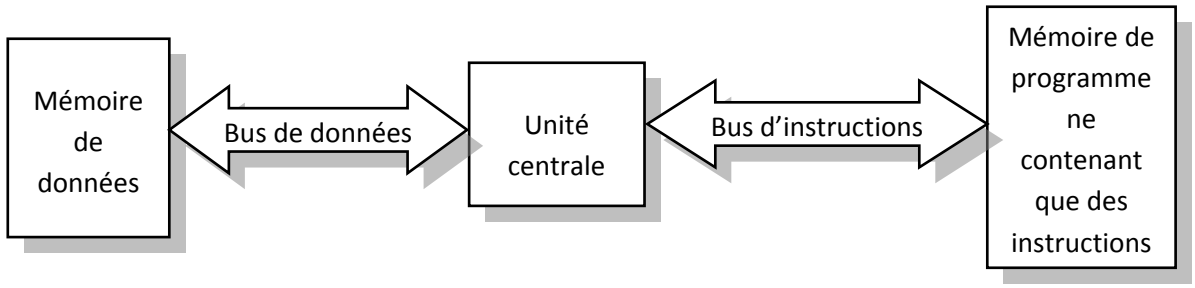

Figure II.3 : Synoptique de l'architecture Harvard.

II.2.1.2 la mémoire

La mémoire est utilisée pour le stockage d'instruction, de données et de la pile. Elle est constituée de cellule de mémorisation binaire, groupées en mots de 8, 16, 32 ou 64 bits. La mémoire est adressée par mots de 8 bits, c'est à dire par octets (bytes). Une position mémoire est spécifiée par l'adresse d'un octet [8].

a) La ROM

Elle contient le programme à exécuter, contrairement à un système informatique classique, il n'est pas question de charger le programme en mémoire vive à partir d'un support car l'application doit commencer dès la mise sous tension, l'organisation de cette mémoire est représentée par la figure II.4 [8].

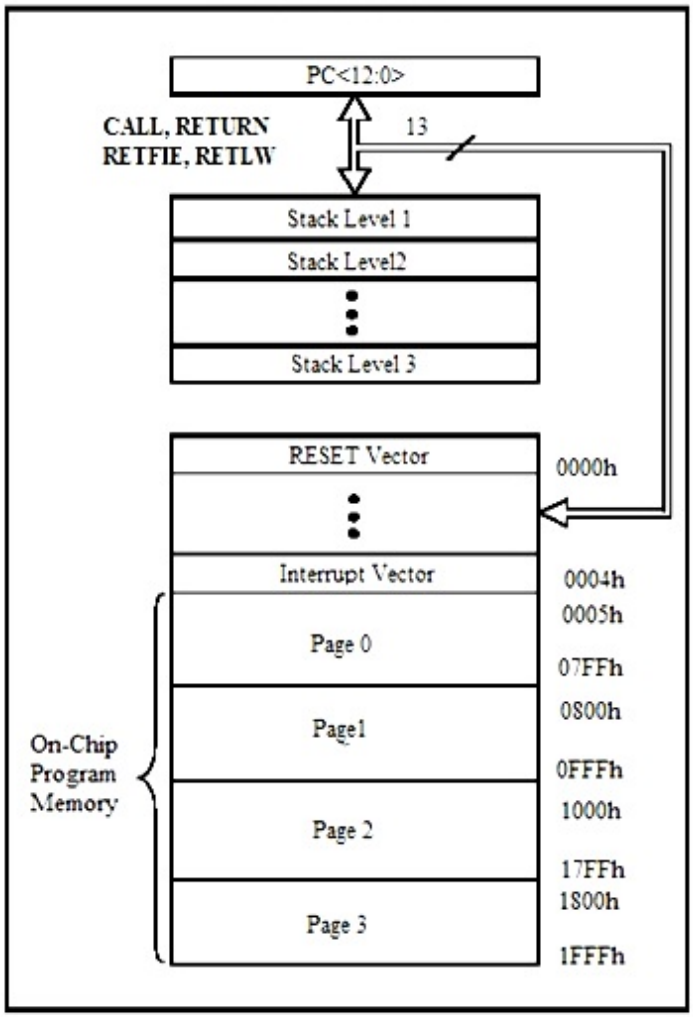

Figure II.4 : Organisation de La mémoire de programme (ROM) du microcontrôleur.

b) La RAM

Ces mémoires perdent l'information lorsqu'elles ne sont pas alimentées. Pour pouvoir travailler normalement le microcontrôleur doit stocker des données temporaires quelques part, et c'est la qu'intervient la RAM. Contrairement à un système informatique classique la RAM d'un microcontrôleur est de petite taille, son organisation est montrée en figure II.5 [8].

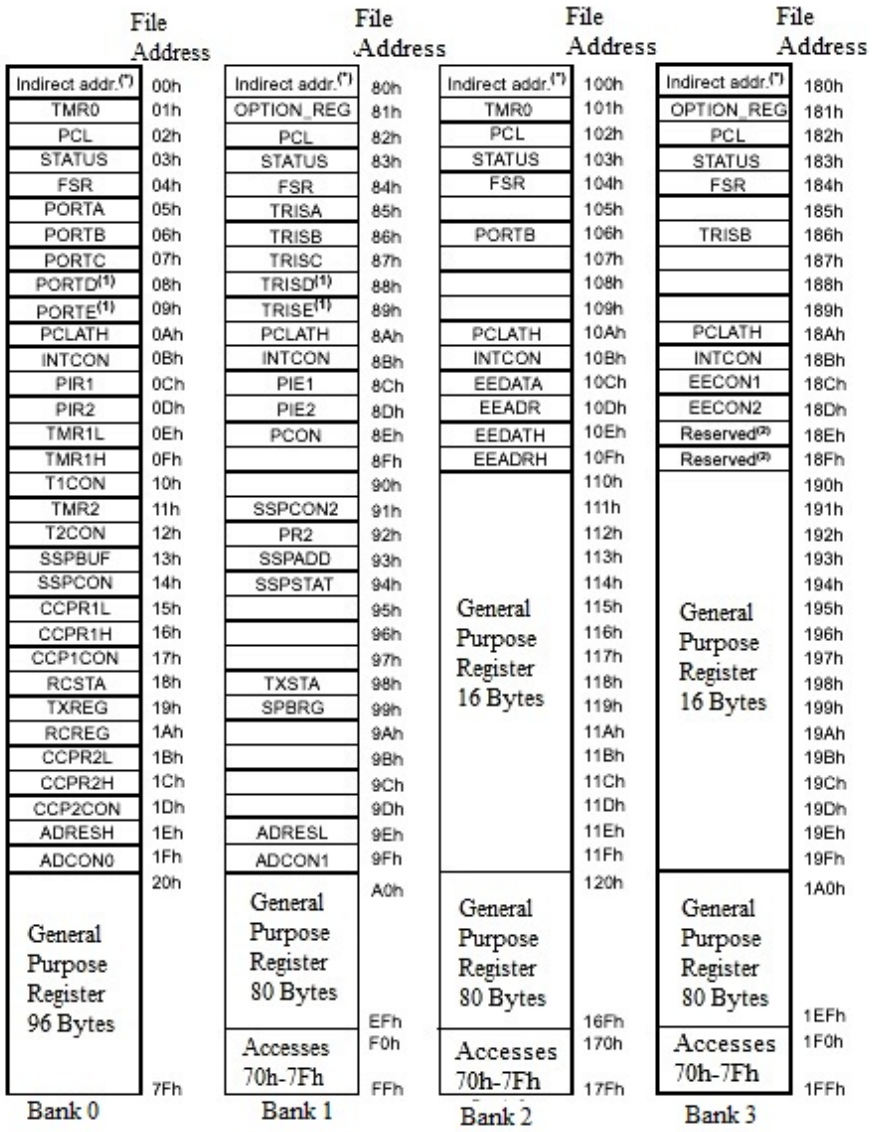

Figure II.5 : Organisation de la RAM du microcontrôleur.

II.2.1.3 Le bus système

Il comporte les lignes permettant de relier entre eux le processeur, la mémoire, et les entrées sorties. Il comprend les lignes d'adresses provenant du microprocesseur, les lignes de données bidirectionnelles et les lignes de contrôle. Le cœur d'un système informatique est formé par l'interaction entre le processeur et sa mémoire [2].

II.2.1.4 L'horloge

C'est un circuit oscillateur délivrant des impulsions à une certaine fréquence, il est nécessaire pour exécuter les opérations séquentiellement. Plus la fréquence sera grande, plus l'unité centrale travaille vite [2].

II.2.1.5 Les entrées/sorties (ports)

Elles permettent au processeur d'accéder au monde extérieur, par l'intermédiaire d'un cycle de lecture et d'écriture sur des interfaces d'entrées-sorties. Elle est généralement formée d'un registre se trouvant à une certaine position d'adressage du microprocesseur. Les circuits d'interfaces peuvent piloter des matériels très différents, (afficheur LCD, moteur pas à pas, communication avec le PC ou autres microcontrôleurs, etc.) [2].

II.2.1.6 Convertisseur Analogique Numérique (ADC)

Selon la version des µcontrôleurs choisie, un convertisseur analogique/numérique 8 ou 10 bit peut faire partie des ressources internes. C'est un modèle à approximations successives, précédé d'un échantillonneur bloqueur et d'un multiplexeur à plusieurs entrées.

La tension de référence de ce convertisseur est comprise entre Vref- et Vref+. Ces tension peuvent être interne, c'est-à-dire VDD et VSS, ou d'une source externe à travers les pattes AN3 et AN2.

Le convertisseur utilise une technique d'échantillonnage-blocage au même titre qu'une capacité avec la tension à mesurer.

La conversion se passe en deux étapes. La première est dite d'acquisition, pendant laquelle l'entrée choisie est reliée au condensateur de l'échantillonneur bloqueur. Cette phase doit durer un temps suffisant pour permettre une charge correcte de ce condensateur. Elle est suivie par la phase de conversion proprement dite, pendant laquelle la tension lue aux bornes du condensateur de l'échantillonneur bloqueur est effectivement convertie en numérique [2].

Le convertisseur (ADC) dispose de 3 registres qui sont nécessaires pour son fonctionnement : deux registres de contrôles sur 8 bits appelés ADCON0 et ADCON1 et un registre de résultat formé de deux registres 8 bits ADRESH et ADRESL.

Les registres du convertisseur analogique/numérique :

a) Le registre ADCON0

Les bits du registre ADCON0 sont représentées par le tableau II.1.

Tableau II.1 : Registre ADCON0.

ADSC1	ADSC0	CHS2	CHS1	CHS0	GO/$\overline{\text{Done}}$		ADCON

bit7 bit0

Au reset ADCON0=00000000

Bit 7 et bit 6 : ADSC1 et ADSC0= Bit de sélection d'horloge.

Ces 2 bits permettent de choisir la vitesse de conversion :

00=Fosc/2 01=Fosc/8 10=Fosc/32 11=Oscillateur RC interne.

Le temps de conversion d'un bit est Tad. Pour une conversion totale des 10 bits il faut : 12.Tad

Pour que la conversion soit correcte il faut que Tad soit au minimum de 1,6µs.

Avec l'oscillateur interne RC on a : Tad= 4µs

Bits 5,4 et 3 : CHS2, CHS1 ET CHS0 = Bits de sélection de canal

Ces 3 bits permettent de choisir l'entrée qui va être convertie.

Tableau II.2 : Configuration des ports analogiques.

Canal	CHS2	CHS1	CHS0	PORT
0	0	0	0	PA0
1	0	0	1	PA1
2	0	1	0	PA2
3	0	1	1	PA3
4	1	0	0	PA5

Bit 2 : GO/$\overline{\text{DONE}}$: Bit de statut si ADON=1.
1 = Démarre la conversion A/D. Ce bit est remis à 0 par hard.
0 = La conversion A/D est terminée.
Bit 1 : Non implanté.
Bit 0 : ADON : état A/D
1 = Convertisseur A/D en service.
0 = Convertisseur A/D à l'arrêt.
Temps de conversion Tad en fonction du Quartz et des bits du Clock select :

Tableau II.3 : Temps de conversion en fonction du Quartz.

Quartz	Clock	Tad	12.Tad	Ne convient pas si Tad <1,6µs
4 MHz	Fosc/2 = 2 MHz	0,5 µs	6 µs	Ne convient pas
	Fosc/8 = 500 KHz	2 µs	24 µs	OK
	Fosc/32 = 125 KHz	8 µs	96 µs	OK
8 MHz	Fosc/2 = 4 MHz	0.25 µs	3 µs	Ne convient pas
	Fosc/8 = 1 MHz	1 µS	12 µs	Ne convient pas
	Fosc/32 = 250 KHz	4 µs	48 µs	OK
12 MHz	Fosc/2 = 6 MHz	0,16 µs	1,92 µs	Ne convient pas
	Fosc/8 = 1,5 MHz	0,66 µs	8 µs	Ne convient pas
	Fosc/32 = 375 KHz	2,6 µs	32 µs	OK
16 MHz	Fosc/2 = 8 MHz	0,125 µs	1,5 µs	Ne convient pas
	Fosc/8 = 2 MHz	0,5 µs	6 µs	Ne convient pas
	Fosc/32 = 500 KHz	2 µs	24 µs	OK

b) Le registre ADCON0

Les bits de configurations de ce registre sont représentés dans le tableau II.4.

Tableau II.4 : Registre ADCON1.

Bit 7							Bit 0
ADFM				PCFG3	PCFG2	PCFG1	PCFG0

Au reset : ADCON1 = 00000000

Bit 7 : ADFM = A/D format du résultat.

1 = Justifié à droite. ADRESH ne contient que les 2 MSB du résultat. Les 6 MSB de ce registre sont lus comme des 0.

0 = Justifié à gauche. ADRESL ne contient que les 2 LSB du résultat. Les 6 LSB de ce registre sont lus comme des 0.

c) Le registre ADRES

La figure II.6 montre les deux cas possible du résultats de la conversion dans le registre ADRES.

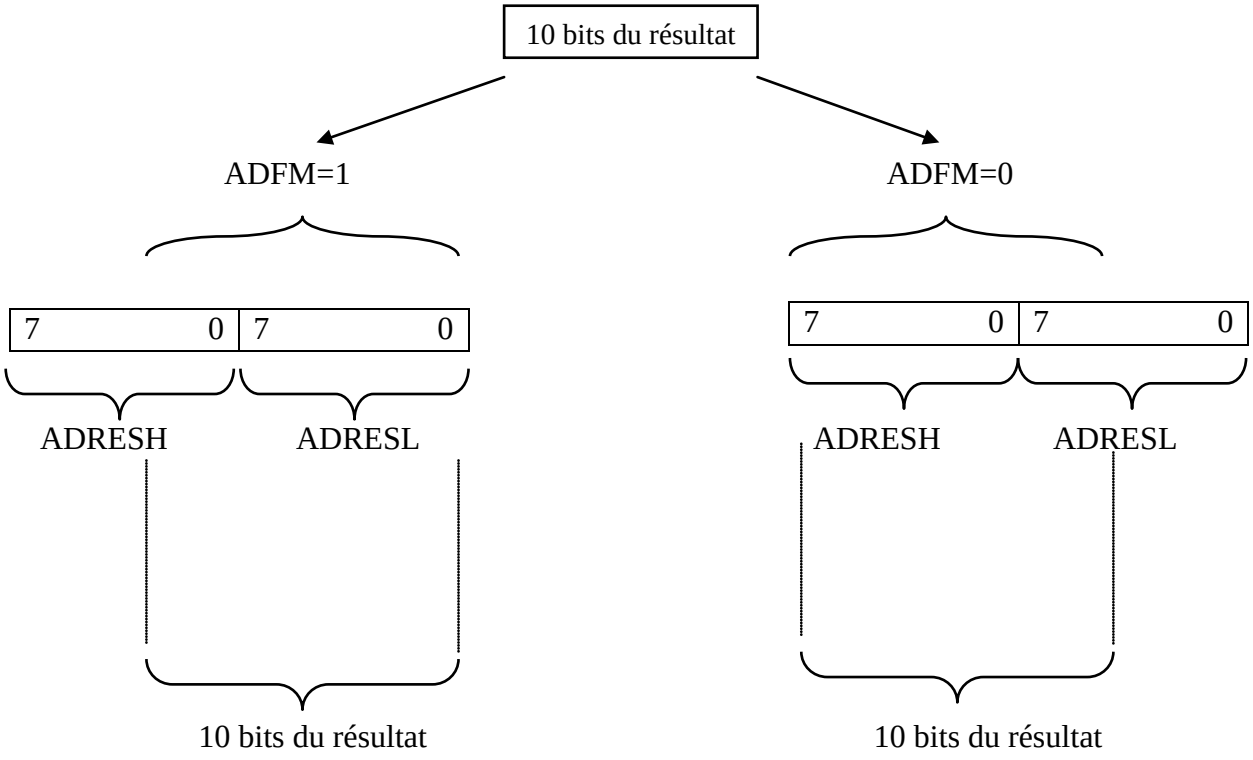

Figure II.6 : Les résultats de conversion dans le registre ADRES.

Bit 6,5 et 4 : Bits non implémentés.

Bits 3,2,1 et 0 : PCFG3, PCFG2, PCFG1 et PCFG0.

Principe de fonctionnement :

- Configurer le convertisseur en programmant les bits des registres de contrôle.
- Valider ou non les interruptions en provenance du convertisseur en programmant le bit du registre PIEx.
- Respecter le temps minimum d'échantillonnage.
- Démarrer la conversion en mettant le bit GO/DONE de ADCON0 à 1.
- Attendre la fin de la conversion, soit en testant le passage à 0 de ce bit (GO/DONE), soit par récitation de l'interruption en provenance du convertisseur si elle a été validée.

- Le résultat de la conversion peut alors être lu dans le registre ADRES, à la suite de quoi il faut aussi remettre le drapeau d'indication de fin de conversion ADIF du registre PIRx à 0 si nécessaire [2].

II.2.1.7 USART

L'USART (Universels Synchronous Asynchronous Receiver Transmitter) est l'un des deux modules de communication série dont dispose le PIC 16F876. L'USART peut être configuré comme système de communication asynchrone full duplex ou comme système synchrone half duplex.

La communication se fait sur les deux broches RC6/TX et RC7/RX qui doivent être configurées toutes les deux le premier en sortie et le deuxième en entrée par TRISC.

Les USART sont caractérisées par :

- Générateur interne de fréquence de cadencement.
- Asynchrone et synchrone opération.
- Horloge Maître ou Esclave.
- Echange de données sur 5 à 9 bits de stop.
- Gestion de parités.
- Communication en full duplex.
- Protection des débordements.
- Détection de faux départs de transmission.
- Filtrage de l'entrée.
- Plusieurs interruption programmables sur le mode émission et réception.
- Mode double vitesse de communication.

L'application principale de ce périphérique est la communication entre le microcontrôleur et un ordinateur via le port série RS232 [2].

a) La Transmission

a.1) Les registres de transmission

Les bits de ce registre sont représentés dans le tableau II.5.

Au reset : TXSTA = 00000010

Bit 7 : CSRC= Clock Source en synchrone. Sans importance en asynchrone.

Bit 6 : TX9 = Autorisation d'émission sur 9 bits.

1 = Autorisé.

0 = Non autorisé.

Bit 5 : TXEN = Autorisation d'émission.

1 = Autorisé.

0 = Non autorisé.

Bit 4 : SYNC = Sélection mode Synchrone ou Asynchrone.

1 = Mode synchrone.

0 = Mode asynchrone.

Bit 3 : Non implémenté

Bit 2 : BRGH = Sélection vitesse rapide en mode asynchrone.

1 = Vitesse haute sélectionnée.

0 = Vitesse basse sélectionnée.

Bit 1 : TRMT = bit d'état du registre à décalage Emission.

1 = Registre vide, donc émission terminée.
0 = Registre plein, donc émission en cours.
Bit 0 : TX9D = 9eme bit de Data transmise.
Ce bit peut être le bit de la parité [2].

Tableau II.5 Registre de transmission de l'USART.

Bit 7							Bit0
CSRC	TX9	TXEN	SYNC		BRGH	TRMT	TX9D

a.2) Schéma fonctionnel du port de transmission

La figure II.7 résume le schéma fonctionnel du port de transmission.

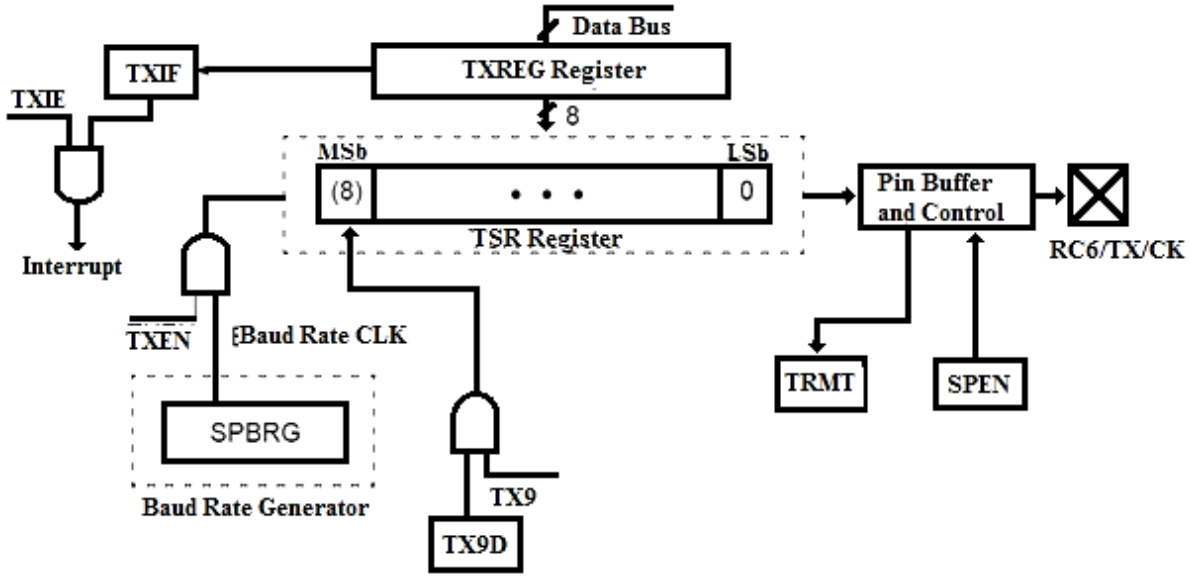

Figure II.7 : Schéma fonctionnel du port de transmission.

b) La réception

b.1) Les registres de réception

Les bits de ce registre sont représentés dans le tableau II.6.

Au reset : RCSTA = 0000000X

Bit 7 : **SPEN**= Serial Port Enable. PC7 et PC6 configurés pour le port série.
1 = Port série en service.
0 = Port série désactivé.
Bit 6 : RX9 = Autorisation de réception sur 9 bits.
1 = Autorisé.
0 = Non autorisé.
Bit 5 : SREN = Single Receive Enable. Réservé pour mode Synchrone.
Non utilisé en mode Asynchrone.
Bit 4: CREN = Continuous Receive Enable.
1 = Autorise la réception en continu.
0 = Désactive la réception en continu.
Bit 3: ADDEN = Adress Detect Enable. En mode Asynchrone 9 bits :
1 = Autorise la détection d'adresse, et charge la Data dans le registre de réception RCREG quand le 9eme bit du registre de désérialisation vaut "1".

0 = Désélectionne la détection d'adresse. Tous les octets sont reçus et le 9eme bit peut servir de bit de parité.
Bit 2 : FERR = Framing Error.
1 = Une erreur de Framing est survenue.
0 = Pas d'erreur de Framing.
Bit 1 : OERR = Overrun Error.
Un octet est reçu alors que le registre de réception n'a pas été vidé par lecture.
1 = Erreur Overrun.
0 = Pas d'erreur Overrun.
Bit 0 : RX9D = 9eme bit de Data reçue.
Ce bit peut être le bit de la parité [2].

Tableau II.6 Registre de réception de l'USART.

SPEM	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

Bit 7 Bit 0

b.2) Schéma fonctionnel du port de réception de l'USART

La figure II.8 résume le schéma fonctionnel du port de transmission.

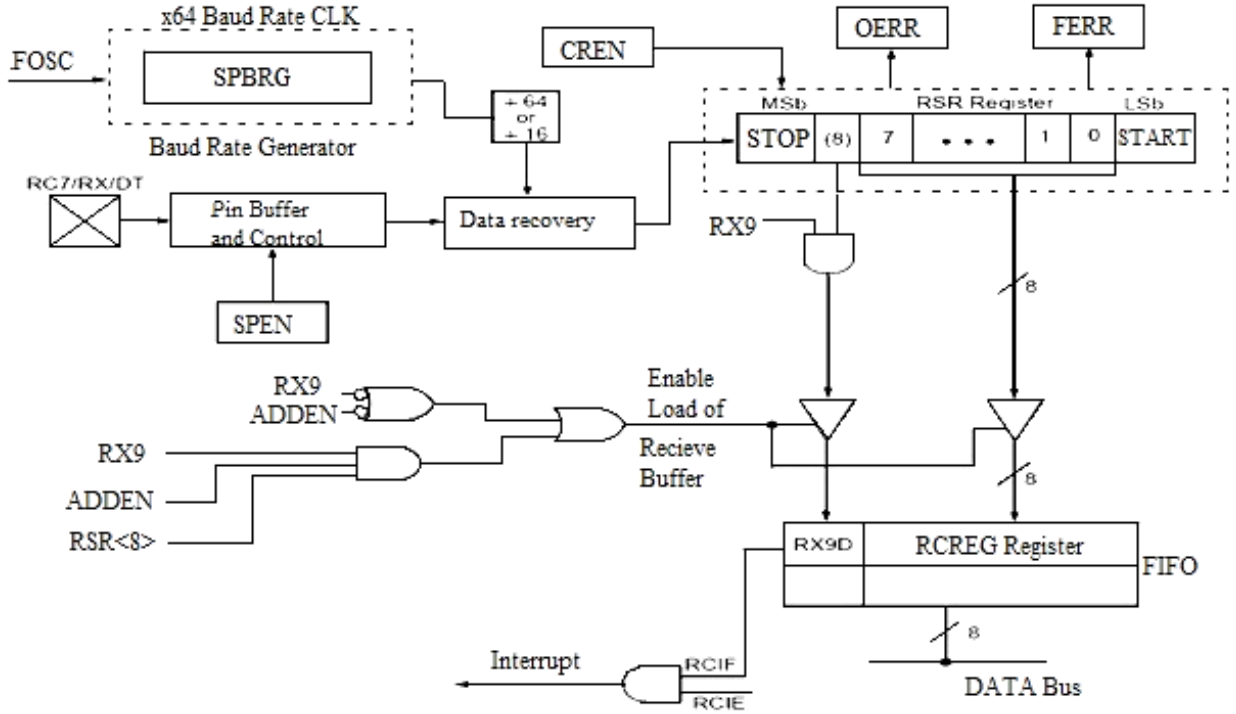

Figure II.8 : Schéma fonctionnel du port de réception de l'USART.

II.2.1.8 Les Timers

Un Timer est un registre interne au microcontrôleur, celui-ci s'incrémente au grès d'une horloge. Ce registre peut servir pour la gestion de temps (réaliser des temporisations), ou bien encore la gestion d'autre fonctions (comptage, captures, comparateur, MLI...).
Le PIC 16F876 possède deux Timers de 8 bits et un de 16 bits configurables par logiciel [2].

a) Timer0

C'est un compteur 8 bits ayant les caractéristiques suivantes :

- Il est incrémenté en permanence soit par l'horloge interne Fosc/4 (mode Timer) soit par une horloge externe appliquée à la broche RA4 du port A (mode compteur). Le choix de l'horloge se fait à l'aide du bit T0CS du registre OPTION_REG

T0CS = 0 horloge interne

T0CS = 1 horloge externe appliquée a RA4

- Dans le cas de l'horloge externe, Le bit T0SE du registre OPTION_REG permet de choisir le front sur lequel le TIMER s'incrémente.

T0SE = 0 incrémentation sur fronts montants

T0SE = 1 incrémentation sur fronts descendants

- Quelque soit l'horloge choisie, on peut la passer dans un diviseur de fréquence programmable (prescaler) dont le rapport DIV est fixé par les bits PS0, PS1 et PS2 du registre OPTION_REG (tableau II.6). L'affectation ou non du prediviseur se fait à l'aide du bit PSA du registre OPTION_REG

PSA = 0 on utilise le prediviseur.

PSA = 1 pas de prediviseur (affecté au chien de garde).

Tableau II.6 : Configuration du prédiviseur.

PS2	PS1	PS0	Div
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

Le contenu du Timer TMR0 est accessible par le registre qui porte le même nom. Il peut être lu ou écrit à n'importe quel moment. Apres une écriture, le Timer ne compte pas pendant deux cycles machine [2].

a.1) Le registre de contrôle du T0CON

Les bits de ce registre sont représentés dans le tableau II.7.

Tableau II.7 : Registre TMR0.

OPTION_REG

RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Bit 7 Bit 0

a.2) Schéma fonctionnel du Timer0

La figure II.9 résume le schéma fonctionnel du Timer0.

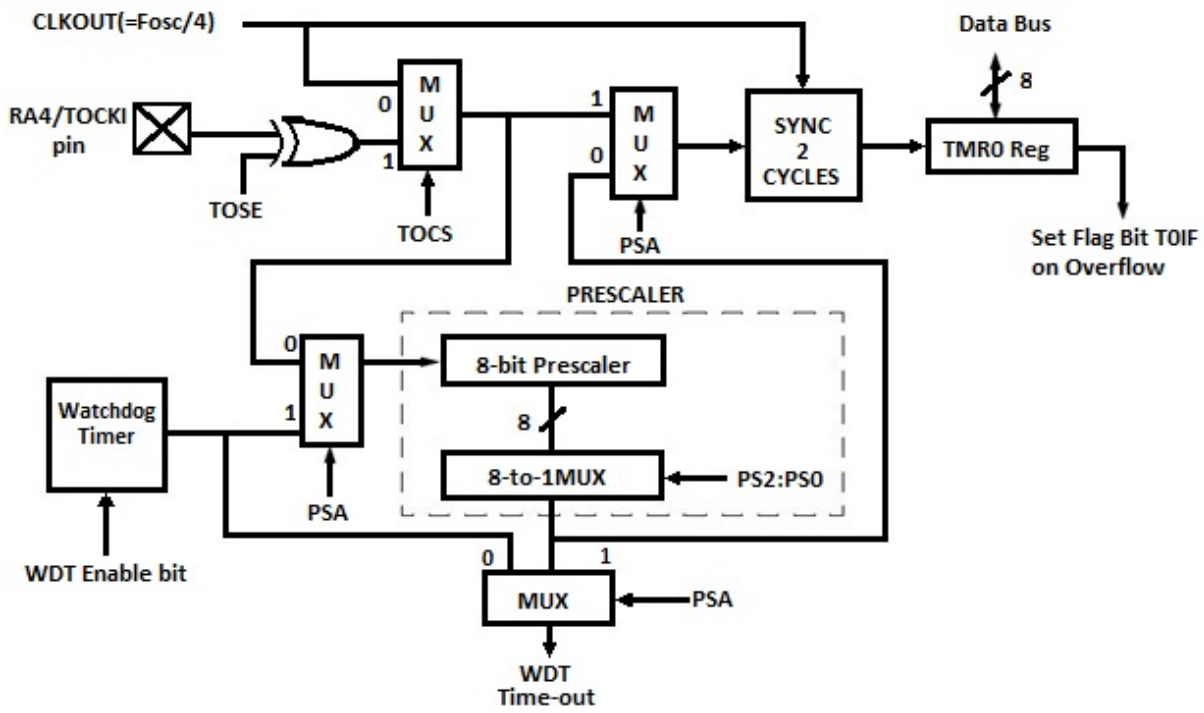

Figure II.9 : Schéma fonctionnel du Timer0.

b) Le Timer1

TMR1 est un Timer/Compteur 16 bits accessible en lecture/écriture par l'intermédiaire des registres 8 bits TMR1H et TMR1L qui constituent sa partie haute et sa partie basse.
On le configure à l'aide du registre T1CON [2].

b.1) Le registre de contrôle du T1CON

Les bits de ce registre sont représentés dans le tableau II.8.

Tableau II.8: Registre TMR1.

Bit 7 Bit 0

———	———	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON

T1CKPS1, T1CKPS0 : Control du prescaler
00 : division par 1
01 : division par 2
10 : division par 4
11 : division par 8
T1OSCEN : Validation de l'Oscillateur associé à TMR1
0 : Oscillateur arrêté
1 : Oscillateur activé
$\overline{\text{T1SYNC}}$: Synchronisation de l'horloge externe (ignoré en mode Timer)
0 : Synchronisation
1 : pas de synchronisation
TMR1CS : Choix de l'horloge du Timer

0 : horloge système (Fosc/4) : mode Timer
1 : Horloge externe : mode compteur
TMR1ON : Démarrer ou arrêter le Timer
0 : Timer1 arrêté.
1 : Timer1 en fonctionnement [2].

b.2) Schéma fonctionnel du Timer1

La figure II.10 résume le schéma fonctionnel du Timer1.

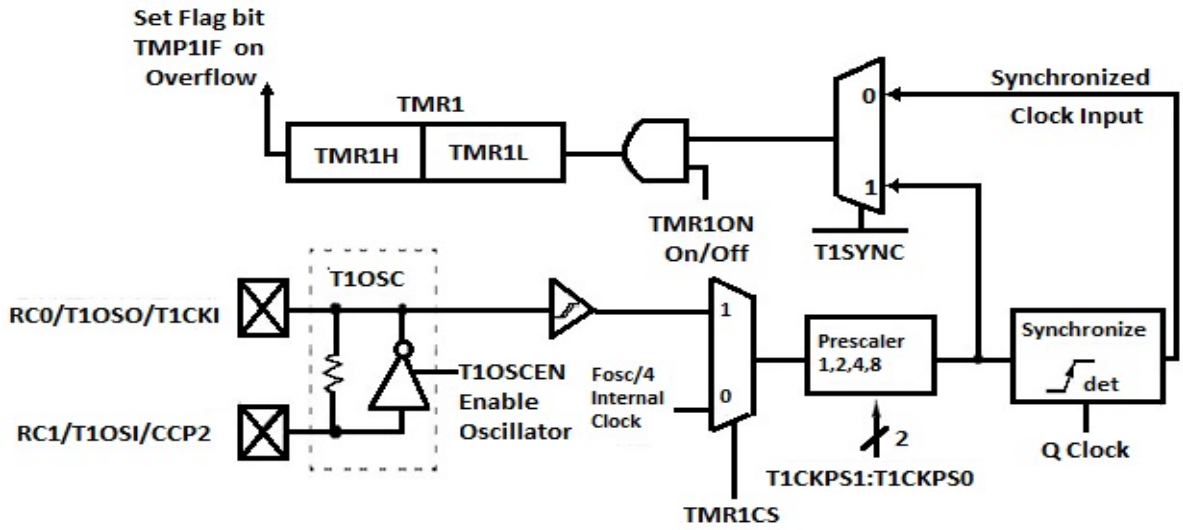

Figure II.10 : Schéma fonctionnel de timer1.

c) Timer2

Timer2 est un compteur 8 bits accessible en lecture et écriture constitué de :

- Un registre de control T2CON
- Un prediviseur (1,4,16)
- Un registre de période PR2 accessible en lecture et écriture
- Un comparateur
- Un post diviseur (1 à 16)

TMR2 est incrémenté par l'horloge interne Fosc/4. Il commence à 0 et quand il atteint la valeur du registre PR2, le comparateur génère un signal qui Remet TMR2 à 0.

- Incrémente le post diviseur qui fonctionne comme un diviseur de fréquence comme le comptage commence à 0, si PR2=N, alors le comparateur annonce une égalité tous les N+1 coups d'horloge.
- Au débordement du post diviseur, le drapeau PIR1.TMR2IF est positionné, l'interruption correspondante et déclenchée si elle est validée.
- TMR2 est remis à zéro à chaque RESET.
- Le prédiviseur et le post diviseur sont initialisés à chaque écriture dans TMR2 ou dans T2CON et au RESET du processeur.
- Le fonctionnement de TMR2 est configuré à l'aide du registre de control T2CON [2].

c.1) Le registre T2CON

Les bits de ce registre sont représentés dans le tableau II.9.

Tableau II.9 : Registre T2CON

Bit7							Bit0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

TOUTPS3:TOUTPS0 : ratio du post diviseur
0000 : division par 1
0001 : division par 2
. . . .
1111 : division par 16
TMR2ON : démarrer/arrêter Timer2
0 : Timer2 arrêté
1 : Timer2 démarré
T2CKPS1, T2CKPS0 : ratio du prédiviseur
00 : prédiviseur par 1
01 : prédiviseur par 4
1x : prédiviseur par 16 [2].

c.2) Schéma fonctionnel du timer2

La figure II.11 résume le schéma fonctionnel du Timer2.

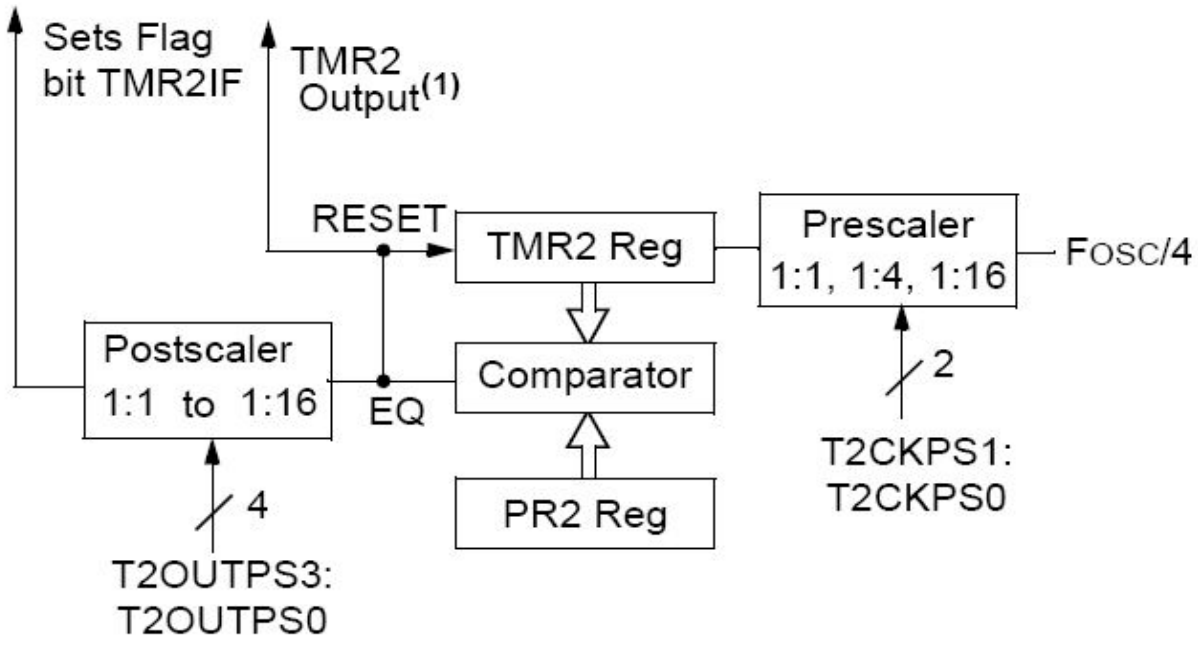

Figure II.11 : Schéma fonctionnel du timer2.

II.2.2 Familles de microcontrôleurs et environnements de programmation

Parmi les fabricants des microcontrôleurs on distingue : Microchip (PIC et dsPIC), PICBASIC, ATMEL, Hitachi, Siemens, Intel et Freescale.

À l'origine, les microcontrôleurs se programmaient en assembleur. Fortement bas niveau, il posa et pose toujours d'énormes problèmes pour la maintenance et l'évolution des logiciels embarqués. Désormais, on utilise de plus en plus des langages de haut niveau, notamment le langage C, capable de faciliter la programmation de microcontrôleurs toujours plus puissants. Avec l'augmentation de la puissance et de la quantité de mémoire de stockage (FLASH) disponible dans les microcontrôleurs, les programmes de ces derniers peuvent désormais être écrits en C++.

II.2.3 Les avantages des microcontrôleurs

- Diminution de l'encombrement du matériel et du circuit imprimé.
- Simplification du tracé du circuit imprimé (plus besoin de tracer de bus)
- Augmentation de la fiabilité du système en réduisant le nombre de composants.
- Intégration en technologie MOS, CMOS, ou HCMOS qui permet la diminution de la consommation.
- moins cher que les composants qu'il remplace.
- Diminution des coûts de main d'œuvre (conception et montage).
- Environnement de programmation et de simulation évolués.

II.2.4 Les défauts des microcontrôleurs

- le microcontrôleur est souvent surdimensionné devant les besoins de l'application
- Investissement dans les outils de développement
- Incompatibilité possible des outils de développement pour des microcontrôleurs de même marque.
- Les microcontrôleurs les plus intégrés et les moins coûteux sont ceux disposant de ROM programmables par masque.

II.3 Afficheur à cristaux liquides (LCD)

II.3.1 Introduction

Les afficheurs à cristaux liquides, appelés afficheurs LCD (Liquid Crystal Display), sont des modules compacts intelligents et nécessitent peu de composants externes pour un bon fonctionnement. Ils consomment relativement peu (de 1 à 5 mA).

Plusieurs afficheurs sont disponibles sur le marché et diffèrent les uns des autres, par leurs dimensions, (de 1 à 4 lignes de 6 à 80 caractères), et aussi par leurs caractéristiques techniques et leur tension de service. Certains sont dotés d'un rétro-éclairage. Cette fonction fait appel à des LEDs montées derrière l'écran du module.

II.3.2 Schéma fonctionnel de l'afficheur LCD

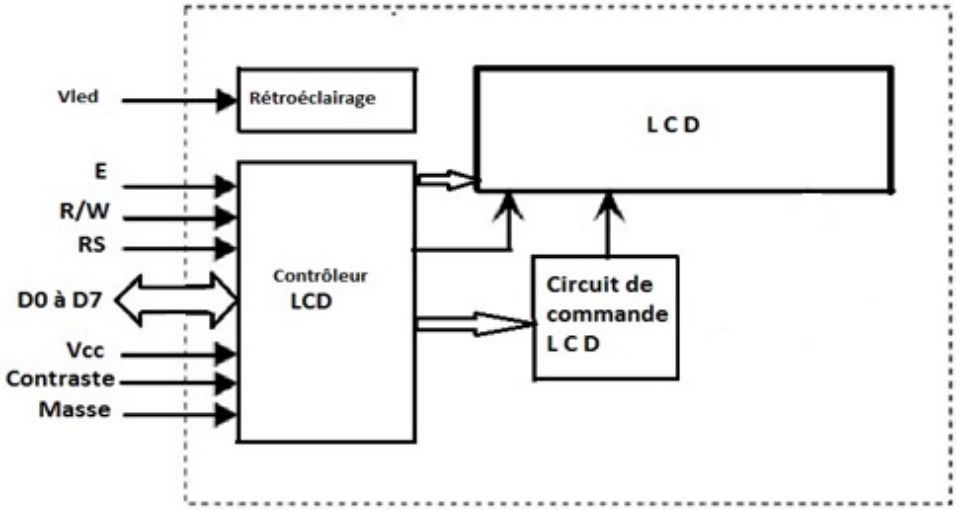

Figure II.12 : Schéma fonctionnel d'un afficheur LCD.

Comme il le montre le schéma fonctionnel dans la figure II.12, l'affichage comporte d'autres composants que l'afficheur à cristaux liquides (LCD) seul. Un circuit intégré de commande spécialisé, le LCD Controller, est chargé de la gestion du module. Le contrôleur remplit une double fonction: d'une part il commande l'affichage et de l'autre se charge de la communication avec l'extérieur [2].

II.3.3Brochage d'un afficheur LCD

Les broches de l'afficheur sont représentées dans le tableau II.10.

Tableau II.10 : Les broches de l'afficheur LCD.

Broches	Nom	Désignation
1,2 et 3	VSS,VDD et VEE	Broches d'alimentation.
4,5 et 6	RS,RW et E	Broches de contrôle.
7,8,9,10,11,12,13 et 14	D0,D1,D2,D3,D4,D5,D6 et D7	Broches de données.

II.3.4 La mémoire d'un afficheur LCD

L'afficheur possède deux types de mémoire, la DD RAM et la CG RAM. La DD RAM est la mémoire d'affichage et la CG RAM est la mémoire du générateur de caractères.

II.3.4.1 La mémoire d'affichage (DD RAM) d'un afficheur LCD :

La DD RAM est la mémoire qui stocke les caractères actuellement affiché à l'écran.

II.3.4.2 La mémoire du générateur de caractères (CG RAM) d'un afficheur LCD:

Le générateur de caractère est quelque chose de très utile. Il permet la création d'un maximum de 8 caractères ou symboles 5x7. Une fois les nouveaux caractères chargés en

mémoire, il est possible d'y accéder comme s'il s'agissait de caractères classiques stockés en ROM [2].

II.3.5 Commande d'un afficheur LCD

Deux modes de fonctionnement de l'afficheur sont disponibles, le mode 4 bits et le mode 8 bits, modes que l'on choisira à l'initialisation de l'afficheur.

II.3.5.1 Mode 8 bits :

Dans ce mode 8 bits, les données sont envoyées à l'afficheur sur les broches D0 à D7. On place la broche RS à 0 ou à 1 selon que l'on désire transmettre une commande ou une donnée. Il faut aussi placer la broche R/W à 0 pour indiquer à l'afficheur que l'on désire effectuer une écriture. Il reste à envoyer une impulsion d'au moins 450 ns sur la broche E, pour indiquer que des données valides sont présentes sur les broches D0 à D7. L'afficheur lira la donnée sur le front descendant de cette entrée.

Si on désire au contraire effectuer une lecture, la procédure est identique, mais on place cette fois la ligne R/W à 1 pour demander une lecture. Les données seront valides sur les broches D0 à D7 lors de l'état haut de la broche E [2].

II.3.5.2 Mode 4 bits :

Il peut, dans certains cas, être nécessaire de diminuer le nombre de fils utilisés pour commander l'afficheur, comme, par exemple lorsqu'on dispose de très peu de broches d'entrées sorties disponibles sur un microcontrôleur. Dans ce cas, on peut utiliser le mode quatre bits de l'afficheur LCD. Dans ce mode, seuls les 4 bits de poids fort (D4 à D7) de l'afficheur sont utilisées pour transmettre les données et les lire. Les 4 bits de poids faible (D0 à D3) sont alors connectés à la masse. On a donc besoin, hors alimentation de sept fils pour commander l'afficheur. Les données sont alors écrites ou lues en envoyant séquentiellement les quatre bits de poids fort suivi des quatre bits de poids faible. Une impulsion positive d'au moins 450 ns doit être envoyée sur la ligne E pour valider chaque demi-octet ou nibble.

Dans les deux modes, on peut, après chaque action sur l'afficheur, vérifier que celui-ci est en mesure de traiter l'information suivante. Pour cela, il faut demander une lecture en mode commande, et tester le flag Busy BF. Lorsque BF=0, l'afficheur est prêt à recevoir une nouvelle commande ou donnée.

Il se peut qu'on dispose encore de moins de broches disponibles dans l'application envisagée. Dans ce cas, on peut alors relier la ligne R/W à la masse de façon à forcer l'afficheur en écriture. On a alors besoin, hors alimentation de seulement six fils en mode 4 bits, et dix fils en mode 8 bits, pour commander l'afficheur, mais on ne peut alors plus relire l'afficheur. Ceci n'est pas gênant dans la mesure où on sait ce qu'on a écrit sur l'afficheur, mais on ne peut alors plus relire le flag Busy. Il faut alors utiliser des temporisations après chaque écriture sur l'afficheur. On perd alors un peu en temps d'affichage, mais on gagne une broche d'entrée sortie [2].

II.3.6 Initialisation d'un afficheur LCD

II.3.6.1 En mode 8 bits

À la mise sous tension de l'afficheur, la ligne supérieur devrait être totalement sombre, celle du bas complètement claire. Si tel n'était pas le cas, il faudra régler le contraste de l'afficheur en jouant sur la tension de la broche Vo. Avant de pouvoir utiliser l'afficheur, il faut tout d'abord l'initialiser. Pour cela, la première commande à envoyer, est la commande

permettant de définir le mode de dialogue avec l'afficheur (DL), et le nombre de lignes sélectionnées (N), et l'envoyer plusieurs fois de façon à ce que celle ci soit comprise, que le mode de départ soit quatre ou huit bits. On peut ensuite paramétrer l'afficheur, puis l'effacer.

Voici donc les différentes commandes (RS=0) à envoyer à l'afficheur LCD. Entre chaque valeur, il faut envoyer une impulsion d'au moins 450 ns sur la ligne E.

33h, 33h, 33h : on force le LCD en mode 8 bits (3xh)
38h : mode 8 bits, 2 lignes, caractères 5x7
0Ch : affichage en fonction, pas de curseur
06h : le curseur se déplace vers la gauche
01h : on efface l'afficheur [2].

II.3.6.2 En mode 4 bits

Pour l'initialisation d'un afficheur en mode quatre bits, on commence par forcer celui-ci dans le mode huit bits, puis quand on est sûr que celui-ci est valide, on bascule en mode quatre bits. Comme on ne sait pas au début de l'initialisation si l'afficheur est positionné en quatre ou huit bits, il est nécessaire d'envoyer la commande de passage en mode huit bits plusieurs fois de façon à ce que celle-ci soit comprise, que le mode de départ soit quatre ou huit bits. Les données sont écrites ou lues en envoyant séquentiellement les quatre bits de poids fort suivi des quatre bits de poids faible, séparés par une impulsion positive d'au moins 450 ns sur la ligne E. En résumé, voici sur quatre bits, les commandes (RS=0) à envoyer à l'afficheur LCD. Entre chaque valeur, il faut envoyer une impulsion positive sur la ligne E.

0h, 1h, 0h, 0h, 1h : on commence par effacer l'afficheur (01h)
3h, 3h, 3h : on force le LCD en mode 8 bits (3xh)
2h : on passe en mode 4 bits (20h)
2h, 8h : mode 4 bits, 2 lignes, caractères 5x7 (28h)
0h, Ch : affichage en fonction, pas de curseur (0Ch)
0h, 6h : le curseur se déplace vers la gauche (06h)
0h, 1h : on efface l'afficheur [2].

La figure II.13 montre un modèle réel d'un afficheur LCD.

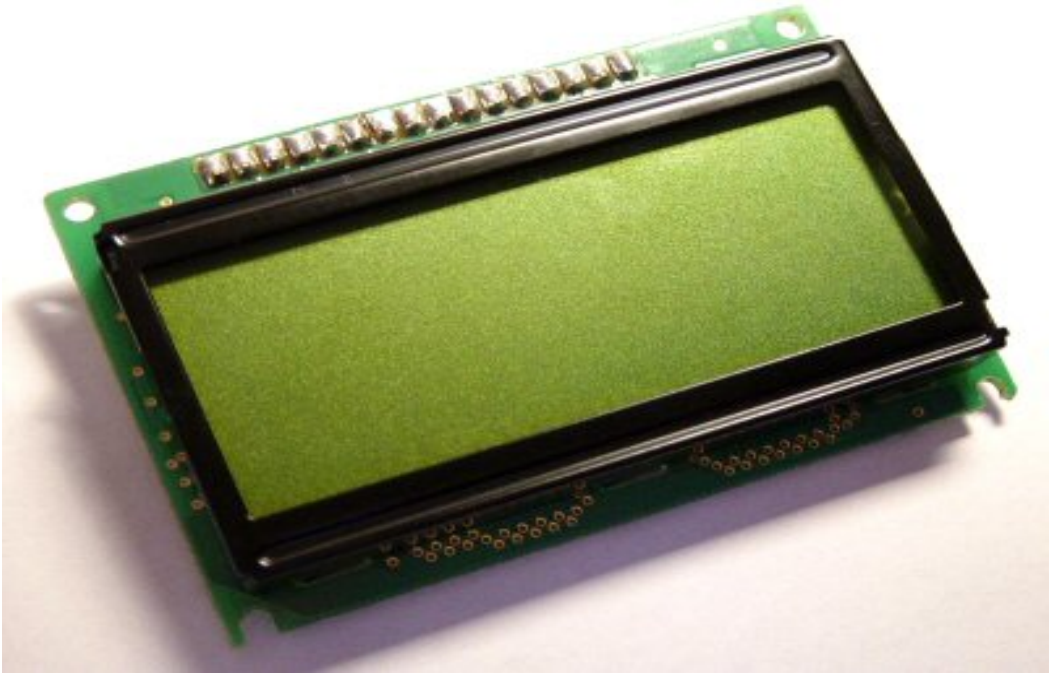

Figure II.13 Afficheur LCD 4x20.

II.4 Conclusion

Dans ce chapitre nous avons décrit l'architecture interne du microcontrôleur et les fonctions de ses périphériques ainsi que l'afficheur LCD. Ce qui nous a permis d'apprendre en particulier à manipuler les différents registres du microcontrôleur, et les méthodes de configuration et d'affichage sur l'afficheur LCD. De sorte à pouvoir les intégrer sans difficultés et les exploiter facilement lors de la conception de la station météo. Cela nous permettra de gérer au mieux ces composants pour les implanter lors de la réalisation du schéma électrique sur le logiciel ISIS PROTEUS.

III.1 Introduction

une fois tous les composants décrits dans les chapitres précédents sont connus, il est temps de les réunir sur un seul schéma électrique. Le logiciel ISIS PROTEUS nous permet de faire cela. Dans ce chapitre, on aborde les étapes pour réaliser un projet sous ce logiciel de simulation, et cela en sélectionnant les éléments décrits auparavant dans la bibliothèque de ISIS, ce qui nous facilite vraiment la tâche, car le logiciel contient une bibliothèque riche en composants électroniques avec des modèles graphiques. On abordera aussi les étapes pour programmer le microcontrôleur avec l'environnement MPLAB et le compilateur CCS Compiler.

III.2 PROTEUS

PROTEUS est une suite logicielle permettant la conception assistée par ordinateur électronique éditée par la société Labcenter Electronics. Il est composé de deux logiciels principaux : ISIS, permettant entre autres la création de schémas et la simulation électrique, et ARES, dédié à la création de circuits imprimés.

Grâce à des modules additionnels, ISIS est également capable de simuler le comportement d'un microcontrôleur (PIC, Atmel, 8051, ARM, HC11...) et son interaction avec les composants qui l'entourent. C’est ce dernier atout qui nous a convaincu de le choisir pour concevoir notre projet.

III.2.1 ARES

Le logiciel ARES est un outil d'édition et de routage qui complémente parfaitement ISIS. Un schéma électrique réalisé sur ISIS peut alors être importé facilement sur ARES pour réaliser le circuit imprimé de la carte électronique. Bien que l'édition d'un circuit imprimé soit plus efficiente lorsqu'elle est réalisée manuellement, ce logiciel permet de placer automatiquement les composants et de réaliser le routage automatiquement (figure II.1).

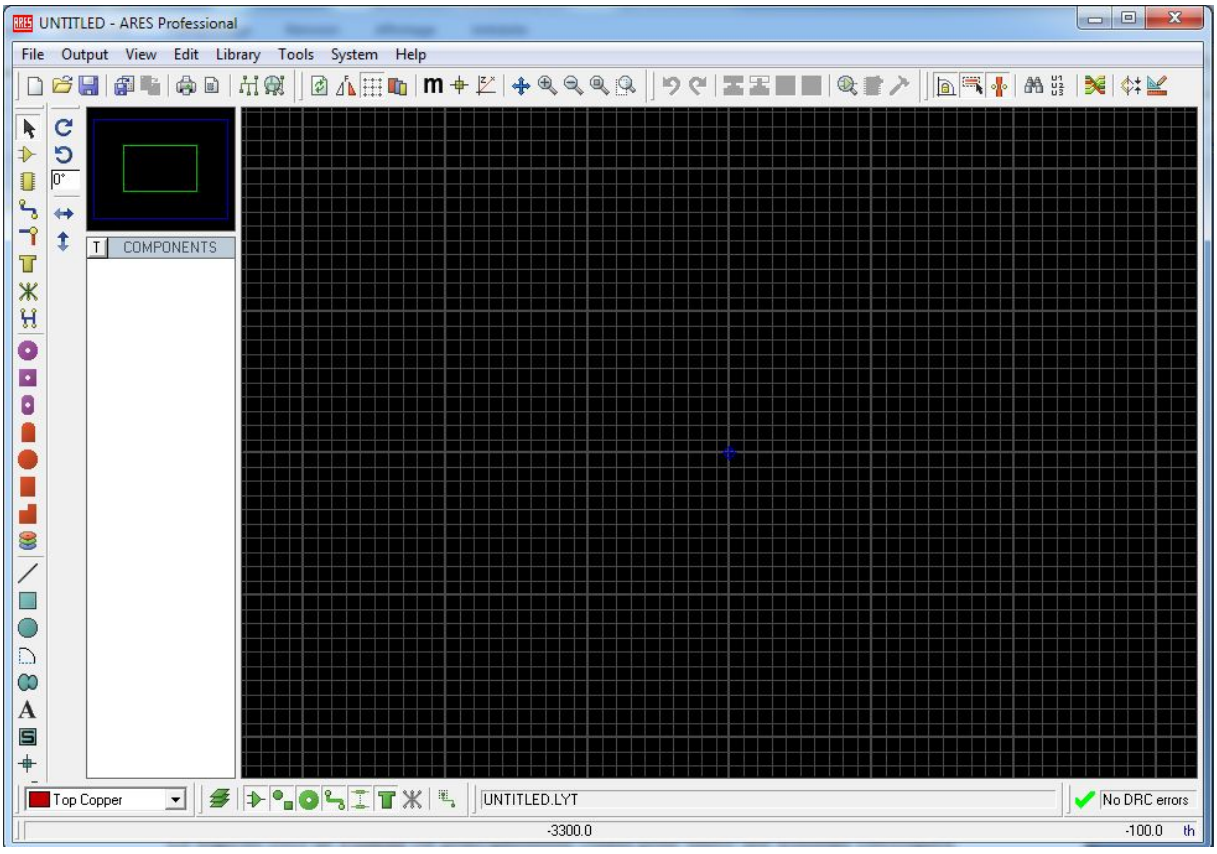

Figure III.1: Fenêtre principale de travail sur ARES.

III.2.2 ISIS

Le logiciel ISIS de PROTEUS est principalement connu pour éditer des schémas électriques. Par ailleurs, le logiciel permet également de simuler ces schémas ce qui permet de déceler certaines erreurs dès l'étape de conception. Indirectement, les circuits électriques conçus grâce à ce logiciel peuvent être utilisé dans des documentations car le logiciel permet de contrôler la majorité de l'aspect graphique des circuits (figure II.2).

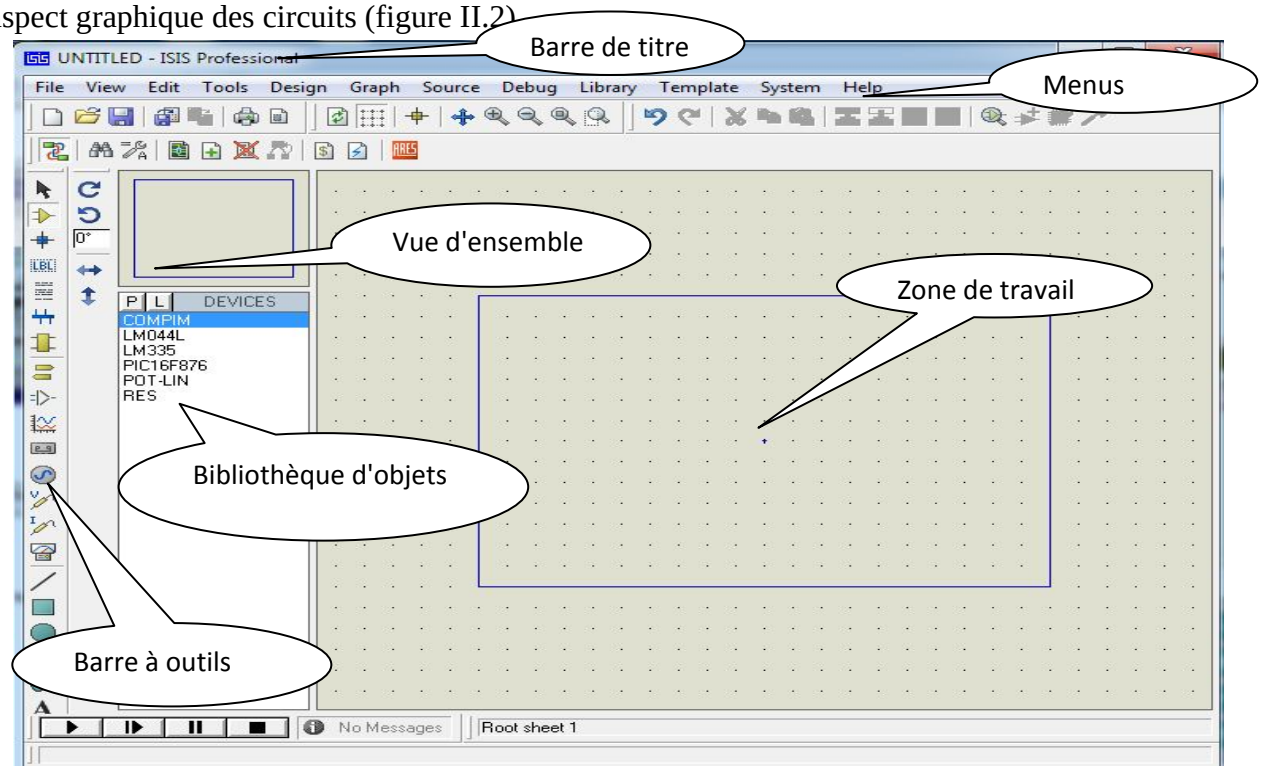

Figure III.2: La fenêtre principale de travail sur ISIS.

III.2.2.1 Sélection des composants à utiliser

Pour faire la sélection des éléments qu'on veut utiliser:

Un clique sur l'icone (Component Mode) puis sur bouton parcourir la bibliothèque P (figure III.3).

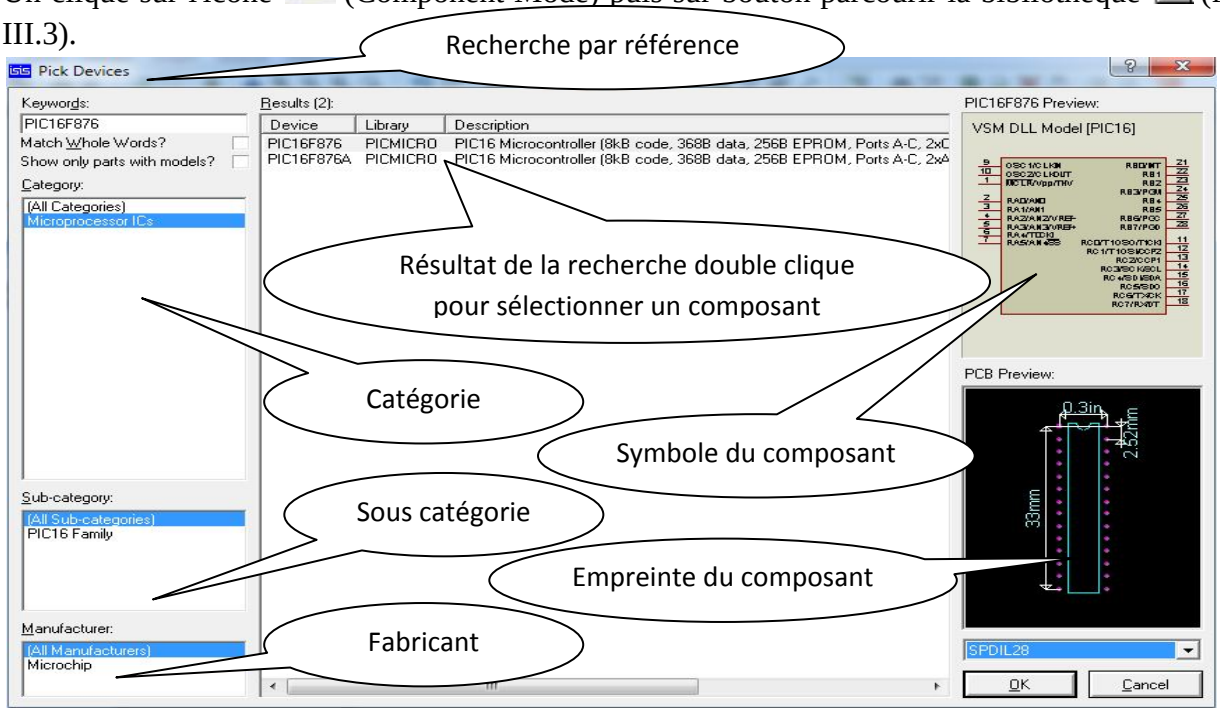

Figure III.3: Bibliothèque ISIS.

III.3 MPLAB

MPLAB est un environnement de développement intégré (IDE), qui permet le développement logiciel des microcontrôleurs PIC et les contrôleurs de signal numériques dsPIC de la société Microchip, il est présenté par la figure III.4. Il nous permet:

- De créer le code source à l'aide de l'éditeur intégré.
- D'assembler, compiler et lier les fichiers sources qui peuvent provenir de langages différents. Un assembleur et un gestionnaire de librairie sont fournis avec MPLAB. un compilateur C est vendu à part par Microchip.
- De déboguer le code exécutable en observant le déroulement du programme.
- D'effectuer des mesures temporelles avec le simulateur ou l'émulateur.
- D'avoir des variables grâce à des fenêtres d'observation.
- De programmer les composants grâce à PICSART ou PROMATE II.

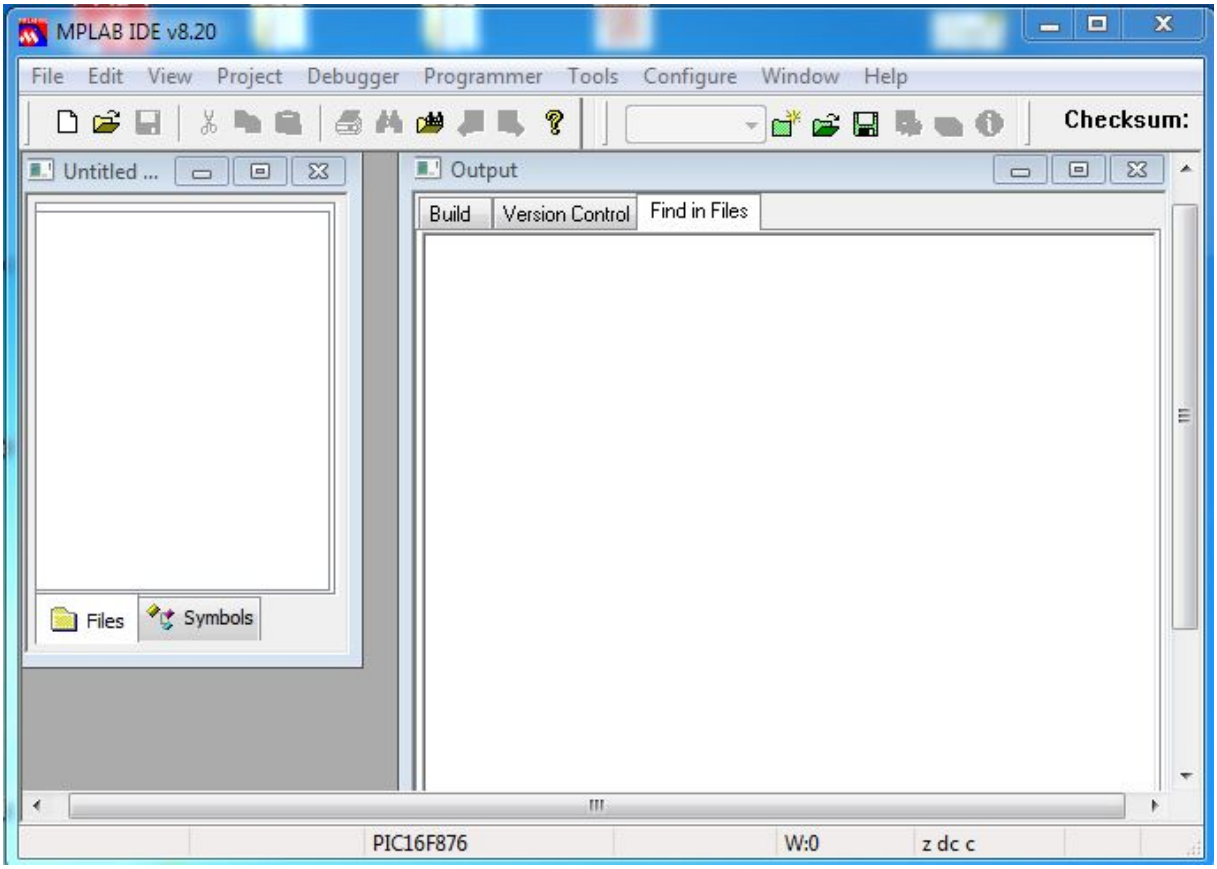

Figure III.4: La fenêtre d'accueil de MPLAB.

III.3.1 Les étapes pour créer un projet dans MPLAB:

Dans la barre de menu, Project; on sélectionne Project wizard qui est un guide pour créer un projet (voir figure III.5).

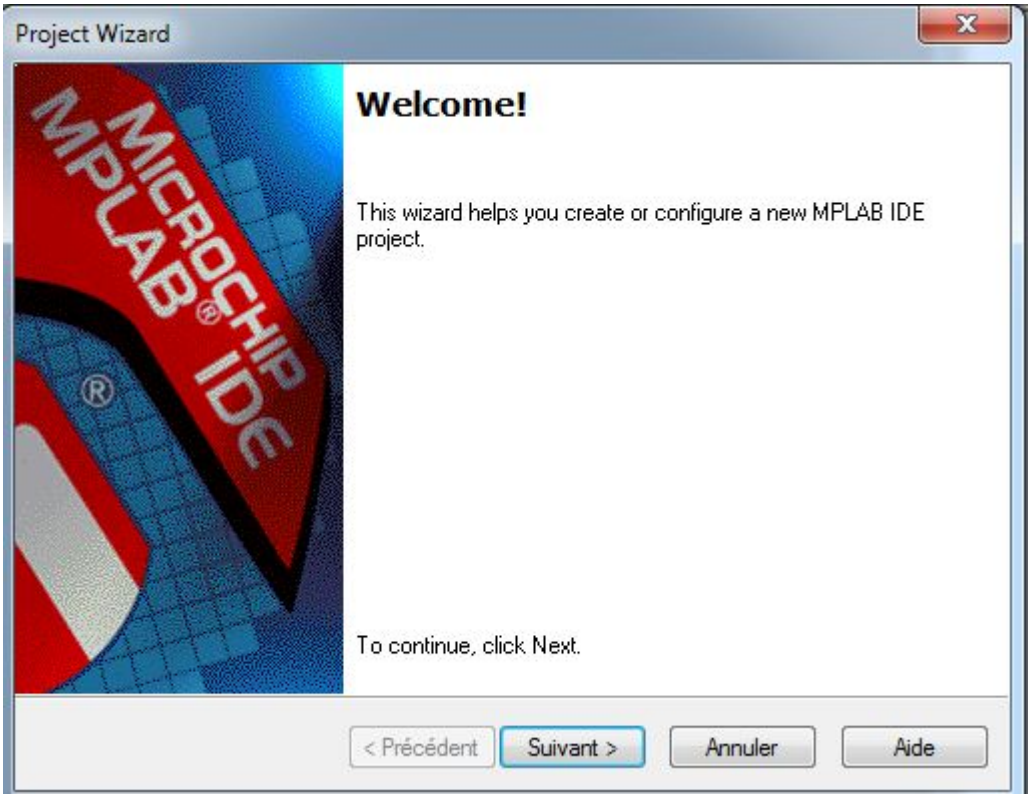

Figure III.5: Guide pour nouveau projet.

La première étape consiste à choisir le composant voulu dans la liste déroulante, dans notre cas on a choisi le PIC16F876 comme le montre la figure III.6.

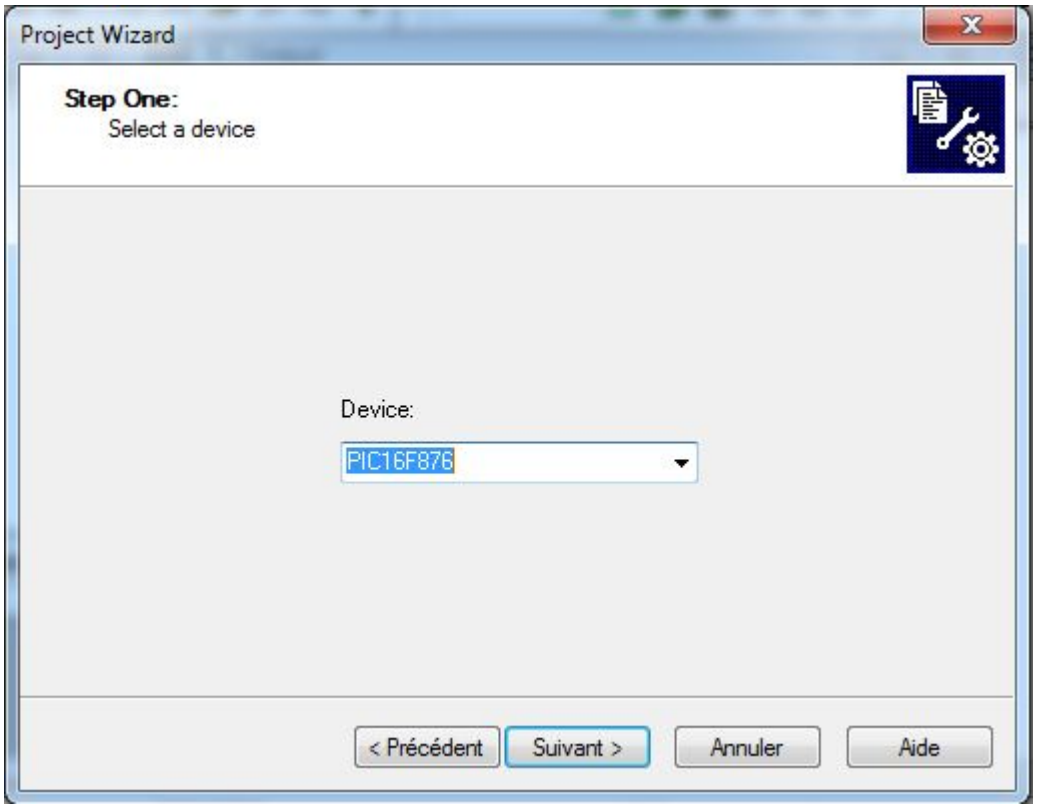

Figure III.6: Choix de composant PIC16F876.

La deuxième étape consiste à sélectionner la suite d'outils logiciels qui va nous permettre de compiler le code source. On a choisis le CCS C Compiler for PIC12/14 /16/18 (voir figure III.7).

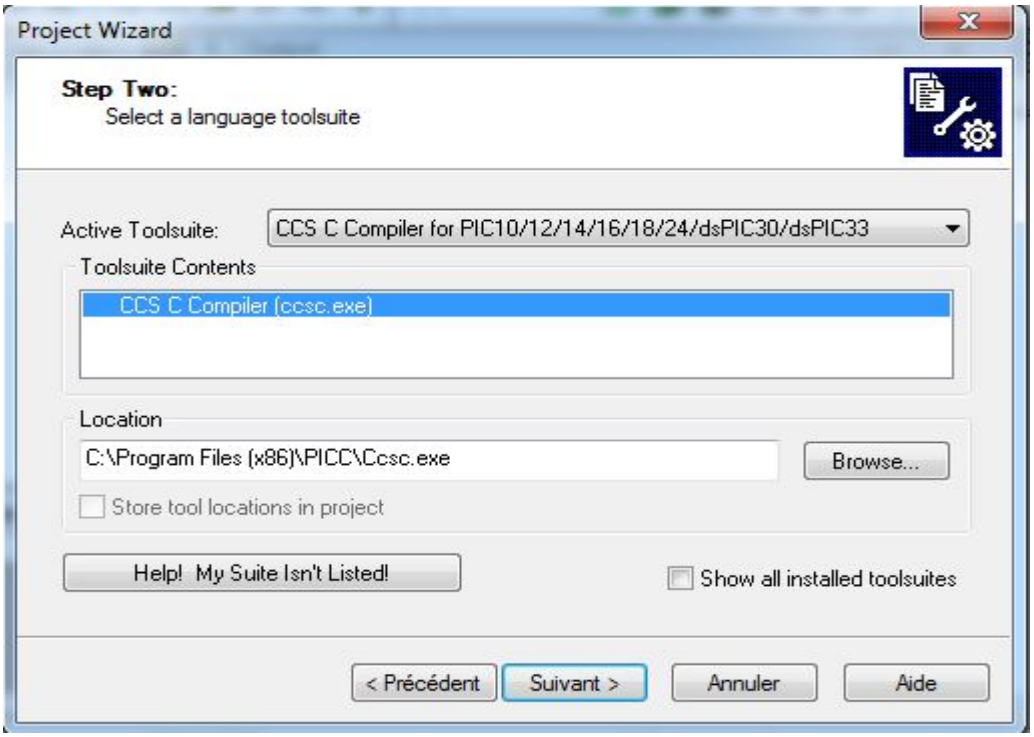

Figure III.7: Choix du compilateur CCS Compiler.

L'étape suivante est de nommer le projet, la fenêtre suivante nous permet de donner un nom à notre projet et d'indiquer dans quel répertoire on souhaite le ranger en cliquant sur Browse comme le montre la figure III.8.

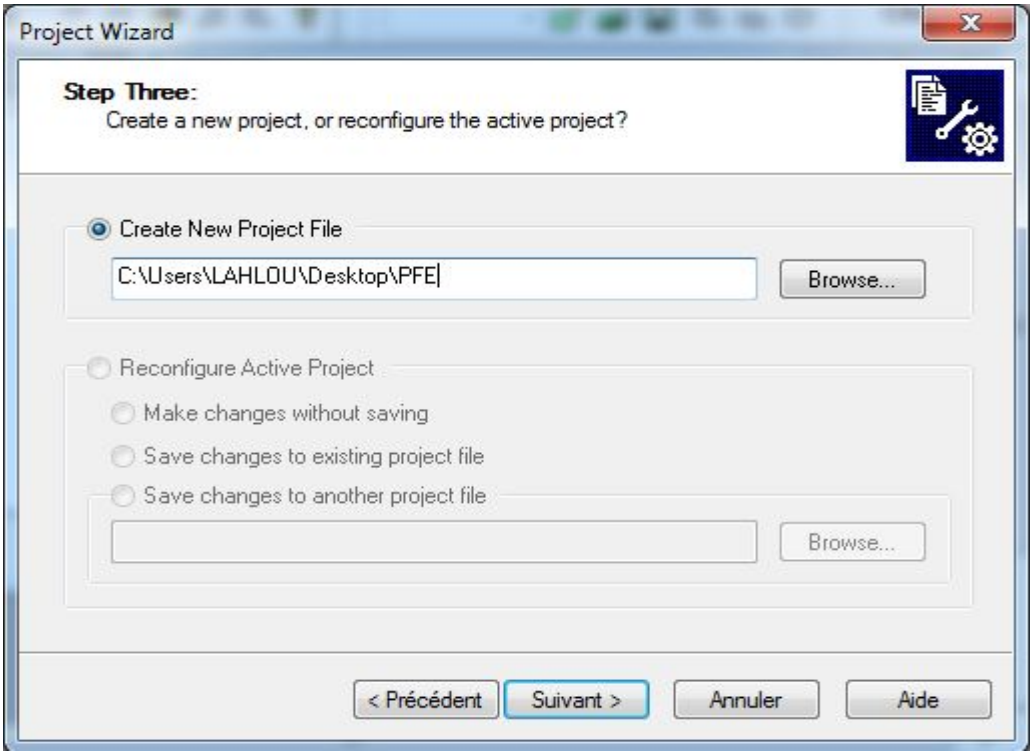

Figure III.8: Nommer le projet et le sauvegarder.

La dernière étape est facultative car elle nous permet d'inclure d'autres fichiers à notre projet si cela est nécessaire. Chose qu'on pourrait faire plus tard aussi comme l'illustre la figure III.9.

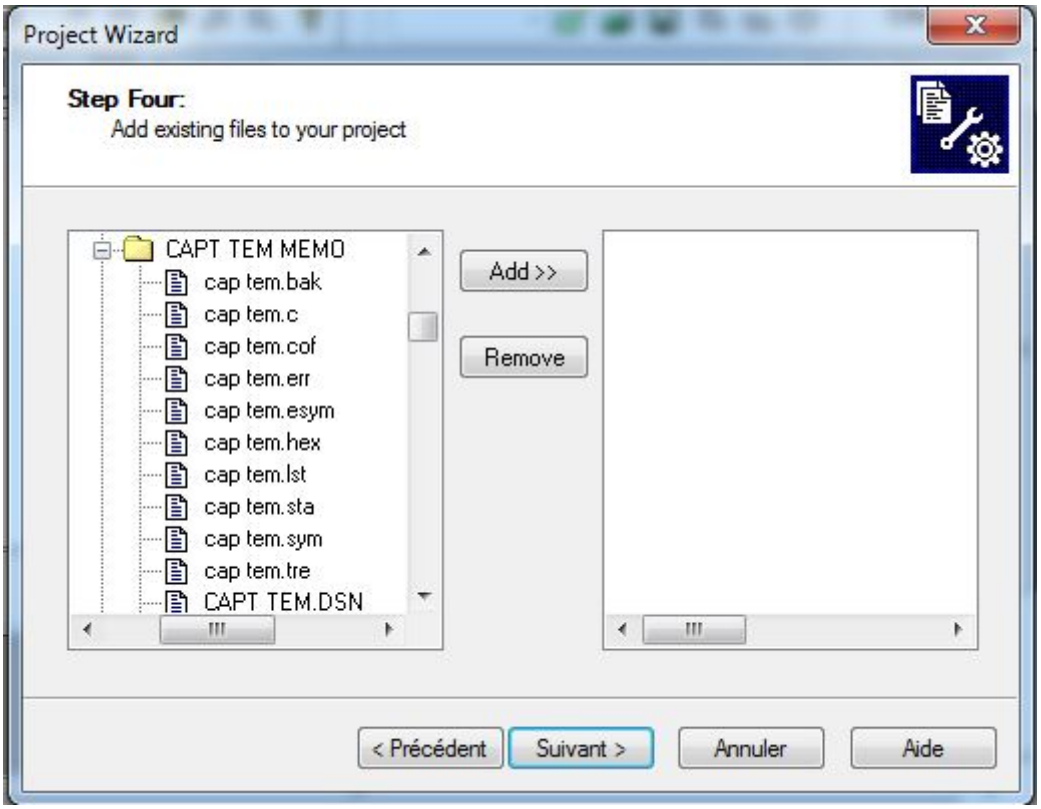

Figure III.9: Inclure d'autres fichiers au projet.

Voici un aperçu (figure III.10) de l'espace de travail de MPLAB:

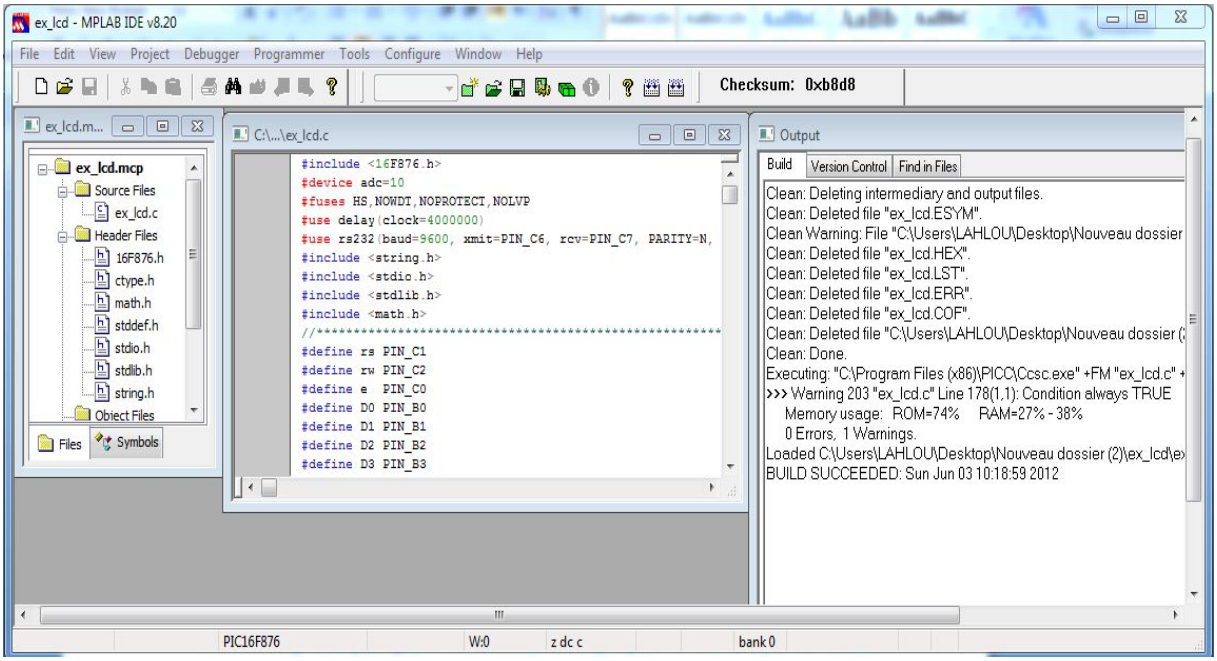

Figure III.10: L'espace de travail de l'IDE MPLAB.

Une fois le code source écrit, on doit construire le projet en appuyant sur le bouton Build All, on doit alors corriger les erreurs éventuelles que le compilateur détecte. Cela permet de générer

un fichier en extension (.hex) à partir du code source. C'est ce dernier fichier qu'on va injecter au microcontrôleur.

III.4 CCS COMPILER

C'est un compilateur de langage C de la firme Custom Computer Services, qui permet de programmer les PICs de Microchip avec le langage C. Il contient les opérateurs standards du langage C, et des bibliothèques intégrées qui sont spécifiques aux registres des microcontrôleurs PICs. Le compilateur fonctionne sous Windows 95, 98, ME, NT4, 2000, XP, Vista, 7 ou Linux. Il génère des fichiers hexadécimaux et de débogage qui sont sélectionnables et compatibles avec les émulateurs populaires et les programmeurs y compris l'IDE MPLAB [7].

III.5 Les éléments constituants la station météo

III.5.1 Microcontrôleur

C'est le composant principal de la station météo, on a opté pour le PIC 16F876 illustré en figure III.11, car il nous convient sur tous les plans. Il est chargé d'acquérir les données, les afficher puis les transmettre via le port RS232. Les instructions de son programme sont contenues dans sa mémoire morte(ROM). Pour injecter le programme dans sa ROM, il suffit de parcourir le code source sous l'extension (.hex) dans les propriétés du microcontrôleur, catégorie program file. Ce dernier fichier est généré par l'environnement MPLAB et le compilateur CCS Compiler. Le PIC saura alors gérer la suite lors de la simulation.

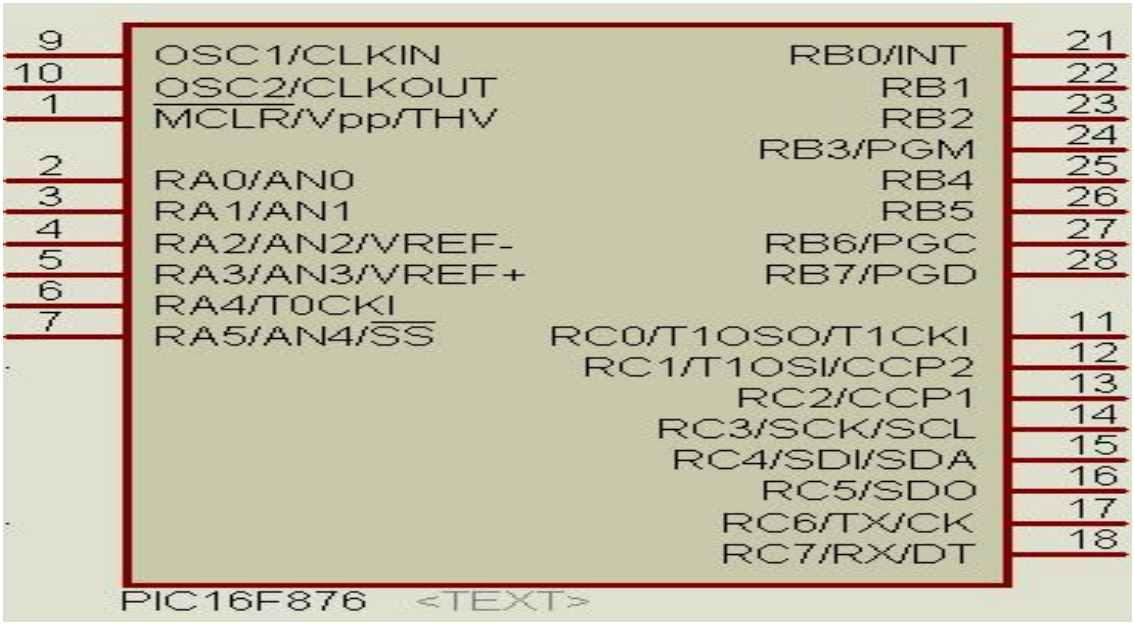

Figure III.11: Image du PIC16F876 sous ISIS.

La figure III.12 montre les propriétés du microcontrôleur dans ISIS.

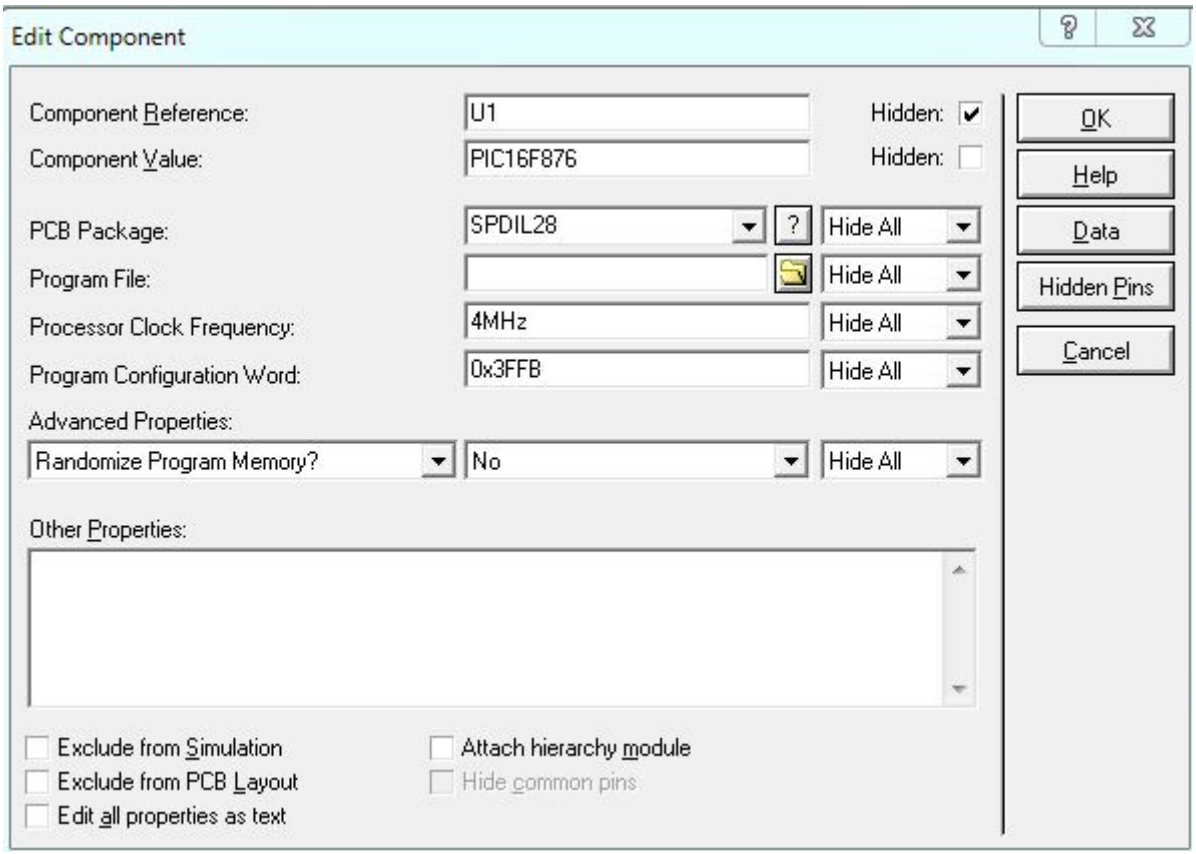

Figure III.12: La fenêtre décrivant les propriétés du PIC16F876.

III.5.2 Capteur de température dans PROTEUS

Le capteur de température est le composant décrit précédemment dans le chapitre I qui n'est autre que le LM335 qui est représenté par la figure III.13. On a opté pour ce modèle car il présente pour nous les avantages suivant:

- ✓ une large plage de mesure.
- ✓ disponibilité.
- ✓ une grande précision.
- ✓ il doit être conditionné pour avoir une variation pleine échelle, chose qui nous a permis d'exploiter d'autre composants électroniques sous proteus.

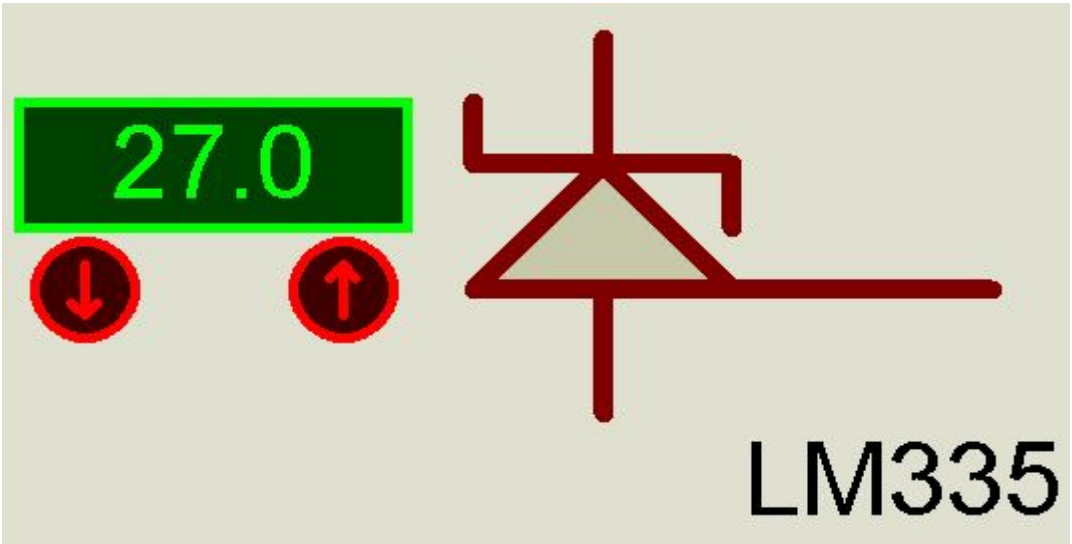

Figure III.13: Capteur de température LM335 sous ISIS.

La figure III.14 montre le conditionnement de capteur LM335 dans ISIS.

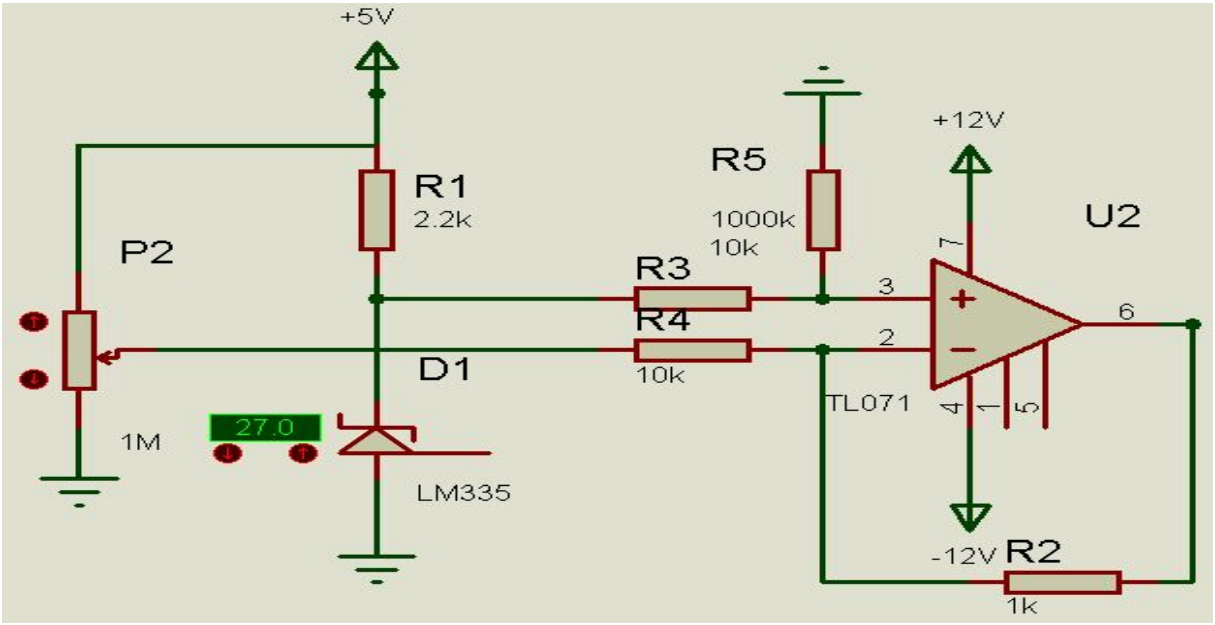

Figure III.14: Capteur de température conditionné.

III.5.3 Capteurs de pression dans PROTEUS

Le logiciel de simulation ISIS ne dispose pas de grande variétés de capteurs de pression atmosphérique, seulement deux sont disponibles. On a choisis le capteur MPX4115 (figure III.15) même s'il est conçu pour mesurer la pression absolue alors que nous souhaitons mesure la pression relative. Il est compensé en température et ne nécessite pas de conditionnement car il a un système de conditionnement intégré ce qui nous permet de l'exploiter directement.

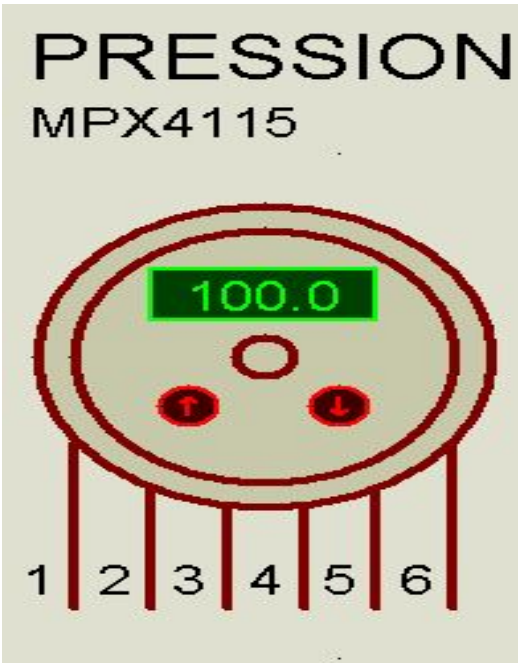

Figure III.15: Capteur de pression MPX4115 sous ISIS.

III.5.4 Capteur d'humidité , vitesse et direction du vent dans PROTEUS

Les capteurs d'humidité relative, direction et vitesse du vent n'existent pas dans la bibliothèque de ISIS, c'est pour cela qu'on était contraint d'utiliser des résistances variables (voir figure III.16), elles nous permettent d'exploiter la tension délivrée à leurs bornes comme sortie d'un capteur réel.

une adaptation doit être faite au niveau du programme du microcontrôleur pour avoir la plage de mesure souhaité.

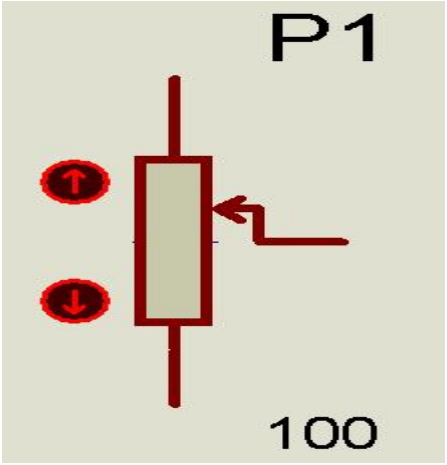

Figure III.16: Résistance variable sous ISIS.

III.5.5 Afficheur LCD

C'est l'instrument qui affiche les résultats acquis par les capteurs, il est aussi géré par le microcontrôleur. On a choisit le modèle à 4 ligne de 20 caractère chacune, alimenté par une tension de 5V. La figure III.17 illustre la référence LM044L du constructeur HITACHI.

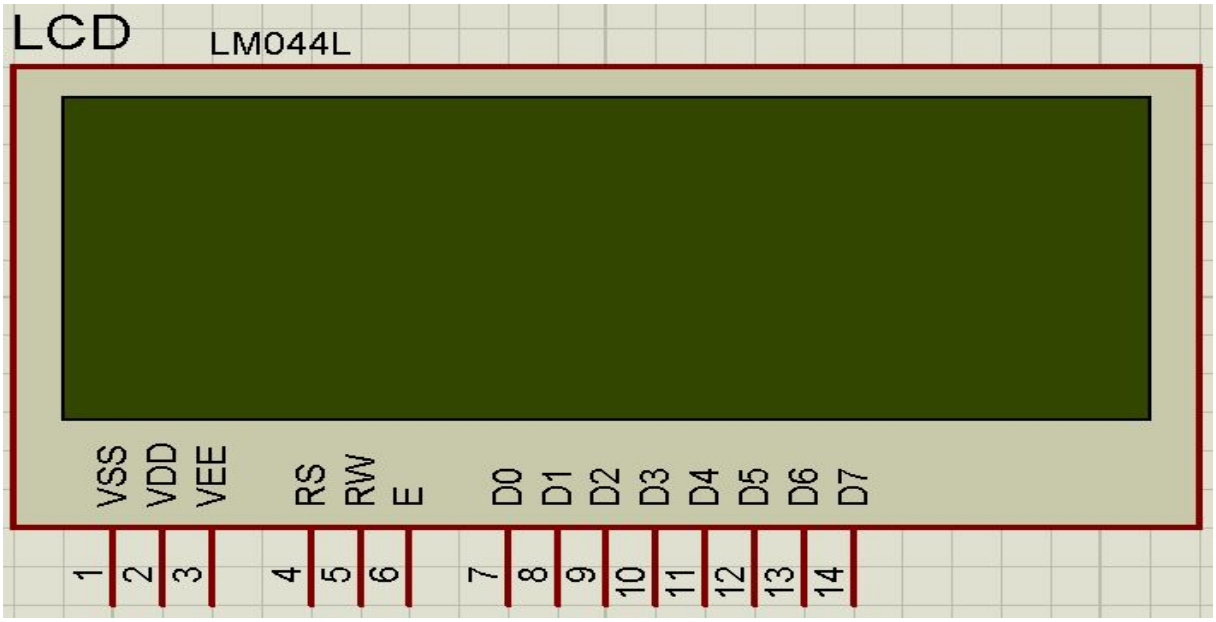

Figure III.17: L'afficheur LCD LM044L 20x4.

La figure III.18 montre les propriétés du l'afficheur dans ISIS.

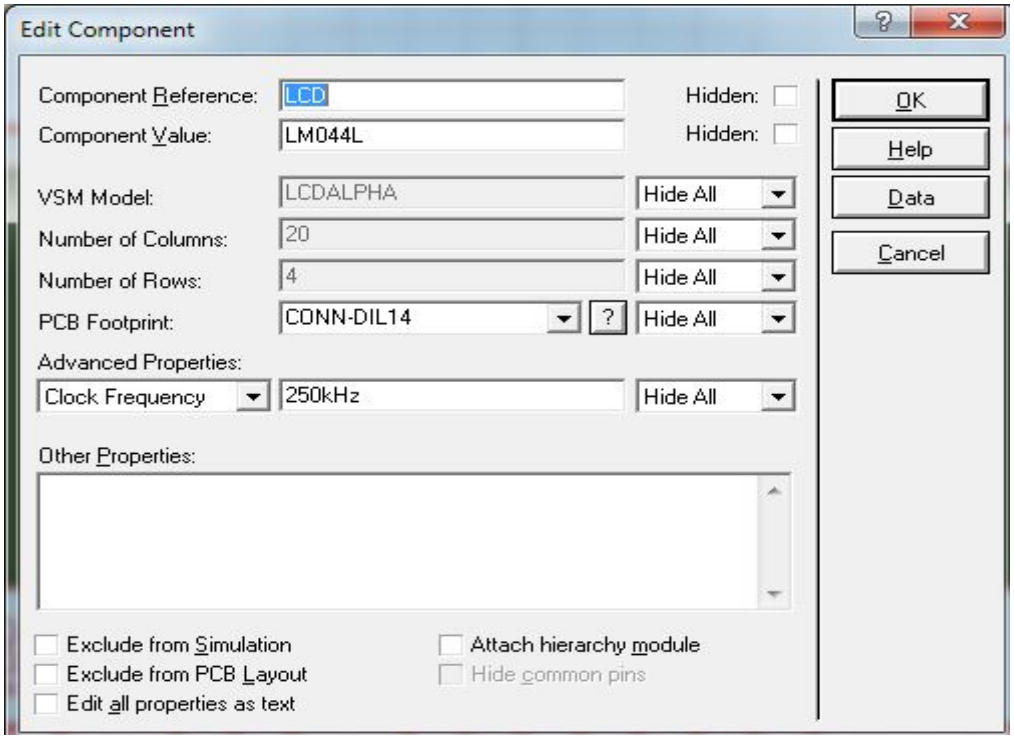

Figure III.18: Les propriétés de l'afficheur.

III.5.6 Le port de communication série RS232

Le composant COMPIM (figure III.19) nous permet de simuler une communication série et de paramétrer les réglages nécessaires pour communiquer avec le PC. Plusieurs choix sont possible, tel que le numéro du port, la vitesse de transmission, nombre de bits de données, avec ou sans bit de parité. Actuellement, la plupart des PC ne disposent pas du port RS232, c'est notre cas. Et pour y remédier, il faut soit acheter un adaptateur USB/RS232, ou bien installer un émulateur de ports série virtuels. on a opter pour la deuxième option car le logiciel Virtual Serial Port Emulateur nous permet non seulement de créer autant de port qu'on souhaite, mais aussi de créer une connexion null modem sans recours à d'autre logiciel pour créer cette passerelle.

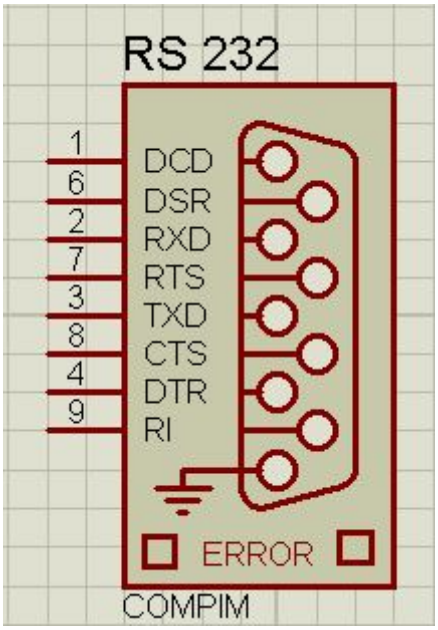

Figure III.19: Port RS232 sous ISIS.

La figure III.20 montre les propriétés du port série dans ISIS.

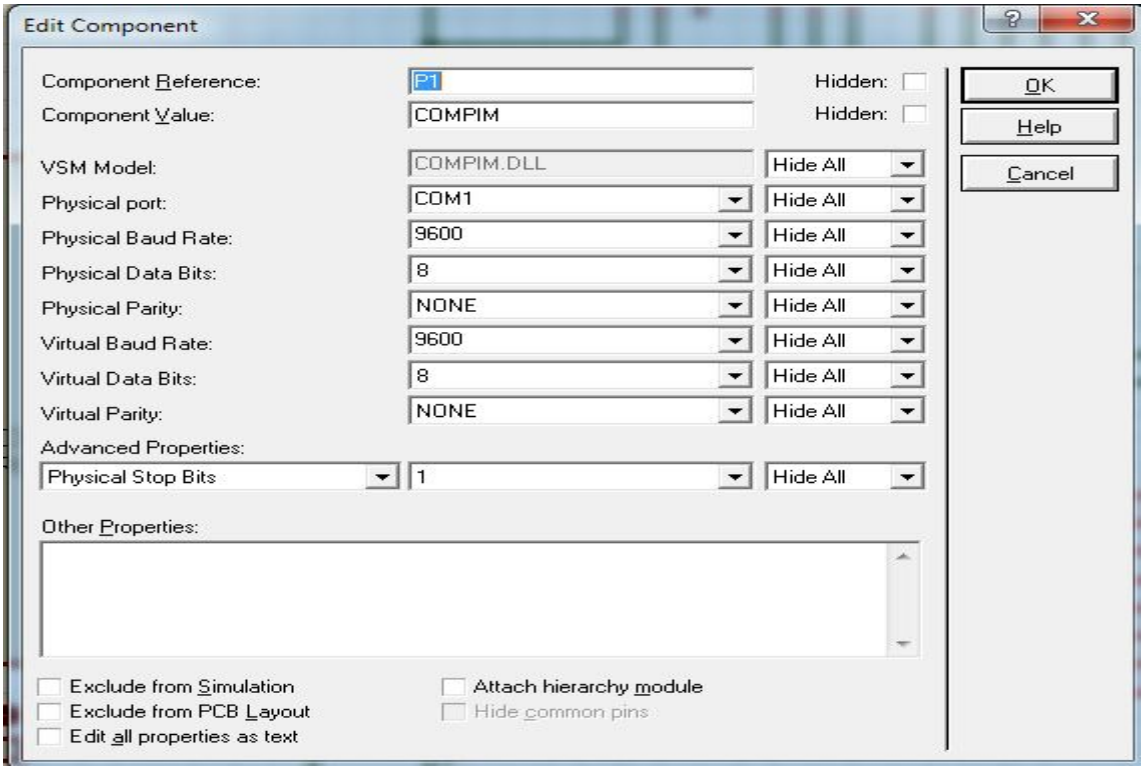

Figure III.20: Paramétrage du port RS232.

III.6 Schéma d'implantation de tous les composants sous ISIS

La phase finale de notre projet est d'implanter le microcontrôleur, les différents capteurs avec leurs éléments de conditionnements, l'afficheur LCD et le port RS232 dans la zone de travail. ISIS met à notre disposition l'outil de liaison virtuel pour connecter les différents composants comme montre la figure III.21.

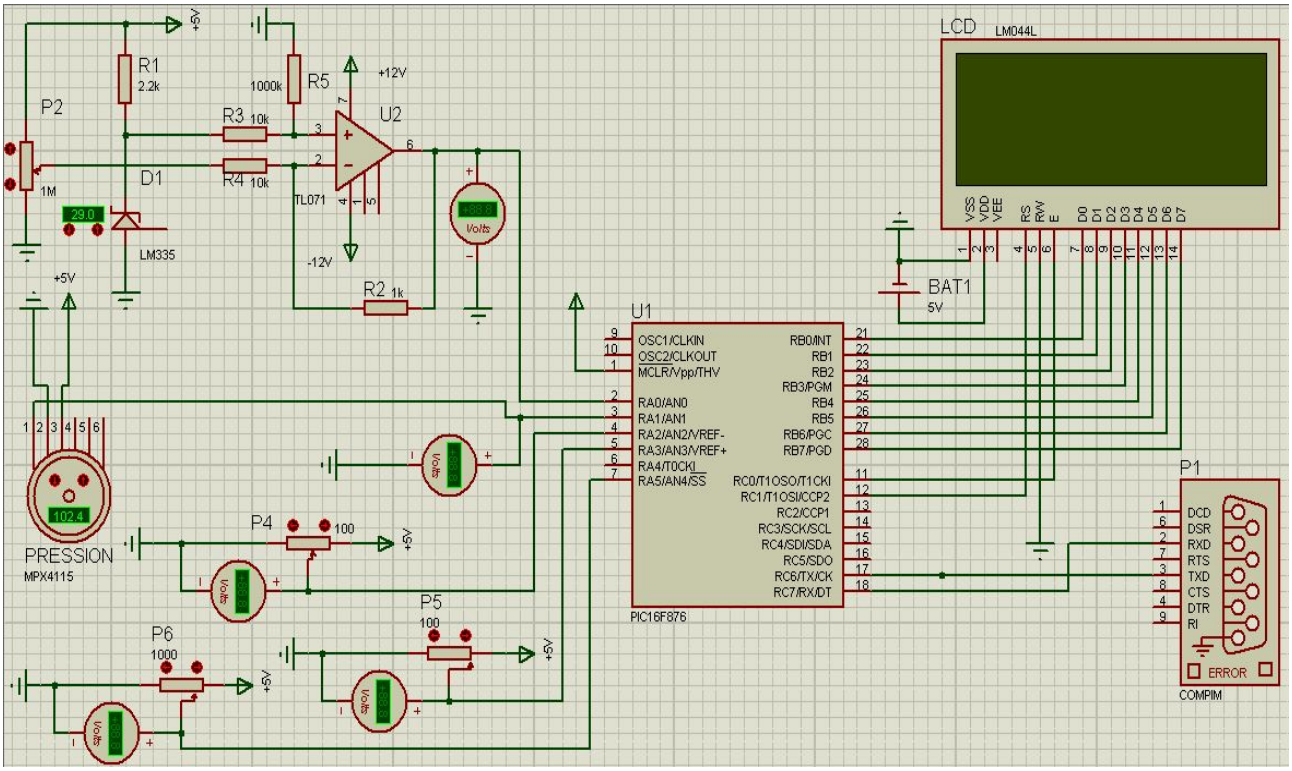

Figure III.21: Schéma d'implantation complet de la station météo.

Une fois les liaisons effectuées, on charge le fichier (hex) dans le microcontrôleur pour piloter l'ensemble des éléments, puis on lance la simulation. On constate que la simulation se passe sans problème, le microcontrôleur se comporte comme on le souhaite car il affiche correctement les grandeurs mesurées aux bornes des capteurs sur l'afficheur LCD même en variant les valeurs des capteurs. Les pins en rouges indiquent qu'elles sont en état logique haut (1), les pins en bleus sont en état logique bas (0) et les pins non configurées sont en couleur grise.

Le port RS232 aussi fonctionne correctement car on peut voir (figure III.22) son activité grâce aux couleurs rouge et vert des ses pins qui correspondent respectivement aux états logiques haut et bas.

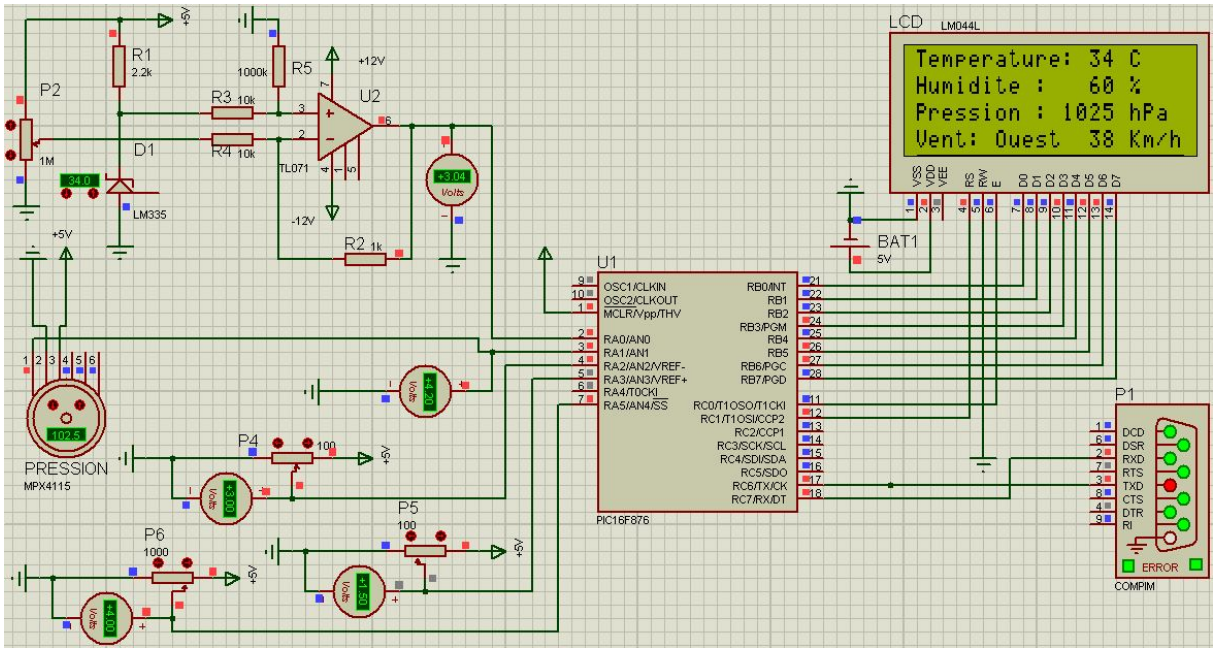

Figure III.22: Capture d'écran du montage lors de la simulation.

III.7 Description de l'organigramme

A la mise sous tension, le microcontrôleur configure ses pins en entrées et sorties, selon leurs configurations effectuées dans le code source. Cela lui permet d'activer l'USART et l'ADC, pour acquérir les données envoyées par les capteurs via ses entrées analogiques (pins 2,3,4,5). Le convertisseur ADC se charge de les convertir en numérique pour les traiter et cela en temps réel. Une fois les données sont connues, le microcontrôleur se charge de les envoyer vers l'afficheur LCD via les pins de données (pins 21-28), tout en respectant les commandes de l'afficheur qui ont été programmées dans le code source et connectées via les pins 11 et 12.

Le microcontrôleur est aussi programmé pour répondre à une éventuelle requête reçue par son port RS232 grâce au périphérique USART activé juste après l'alimentation , relié aux pins 17 et 18. Il permet une interactivité avec le PC. Sa réponse est conditionnée par le caractère reçu, ainsi :

- ✓ s'il reçoit le caractère '0', il envoie la valeur de la température.
- ✓ s'il reçoit le caractère '1' il envoie la valeur de l'humidité.
- ✓ S'il reçoit le caractère '2' il envoie la valeur de la pression atmosphérique.
- ✓ S'il reçoit le caractère '3' il envoie la valeur de la vitesse du vent.
- ✓ S'il reçoit le caractère '4' il envoie la valeur de la direction du vent.
- ✓ S'il reçoit un autre caractère, il arrête la transmission via le port RS232.

L'organigramme de la figure III.23 résume ce procédé.

III.8 Organigramme du programme principal

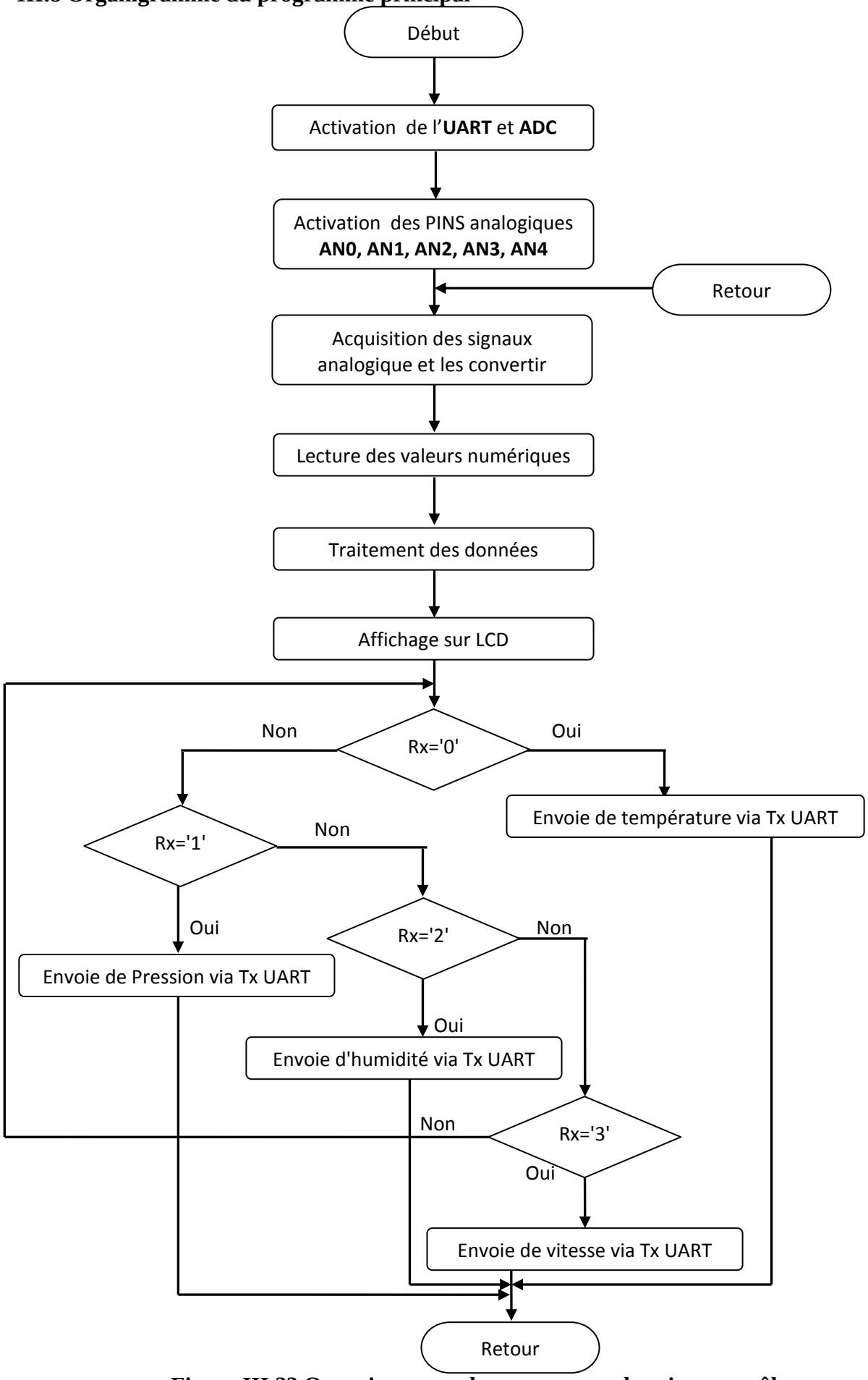

Figure III.23 Organigramme du programme du microcontrôleur.

III.9 Conclusion

Dans ce chapitre on a pu faire le schéma d'implantation dans le logiciel ISIS de tous les composants électroniques de la station météo, où on a pu voir le comportement de la station météo lors de la simulation, y compris l'afficheur LCD, et tout cela piloté par le microcontrôleur comme s'il s'agissait d'un modèle réel. La simulation donnait des résultats satisfaisants, où on constate que même en changeant les valeurs au niveau des capteur, le microcontrôleur les met à jour en envoyant leurs valeurs vers l'afficheur. l'inconvénient majeur est au niveau de l'afficheur qui ne pouvait pas afficher une chaine de caractères, chose qui nous a contraint à lui envoyer caractère par caractère, ce qui nous a obligé à faire appel a plusieurs boucles. Reste à voir l'interactivité avec le PC grâce à la liaison série RS232, chose que nous allons aborder dans le chapitre suivant.

IV.1 Introduction

L'autre aspect de notre projet est la liaison série avec le PC via le port RS232, cela va nous permettre de visualiser directement les grandeurs mesurées sur le PC, et de les sauvegarder dans un fichier grâce à une application conçue spécialement à cet effet. Ce fichier pourrait nous servir ultérieurement pour une éventuelle reproduction des variations des grandeurs sous forme de graphe ou pour d'autres traitements.

IV.2 Présentation de DELPHI

Delphi est un outil de développement informatique qui permet de créer des applications. C'est un langage de programmation basé sur Pascal, fondé sur les notions d'événements et d'objets. Il permet de créer simplement des interfaces graphiques. En Delphi, on construit des projets. Un projet est constitué d'unités qui ressemblent à des programmes Pascal et de fiches qui définissent l'interface graphique correspondante. Dans un projet, une unité est associée à chaque fiche. Chaque unité et chaque fiche seront stockées dans des fichiers différents (extension PAS pour les unités, DFM pour les fiches ; le projet ayant l'extension DPR). Delphi engendre automatiquement d'autres fichiers [5].

IV.3 Vue d'ensemble de la Programmation Orientée Objet :

IV.3.1 L'objet

Il est impossible de parler de Programmation Orientée Objet sans parler d'objet, bien entendu. Un objet est avant tout une structure de données. Autrement, il s'agit d'une entité chargée de gérer des données, de les classer, et de les stocker sous une certaine forme. En cela, rien ne distingue un objet d'une quelconque autre structure de données. La principale différence vient du fait que l'objet regroupe les données et les moyens de traitement de ces données. Un objet rassemble de fait deux éléments de la programmation procédurale.

- ✓ Les champs : Ils sont à l'objet ce que les variables sont à un programme : ce sont eux qui ont en charge les données à gérer. Tout comme n'importe quelle autre variable, un champ peut posséder un type quelconque défini au préalable : nombre, caractère... ou même un type objet.
- ✓ Les méthodes : Elles sont les éléments d'un objet qui servent d'interface entre les données et le programme. Sous ce nom se cachent simplement des procédures ou fonctions destinées à traiter les données.

Les champs et les méthodes d'un objet sont ses membres. Si nous résumons, un objet est donc un type servant à stocker des données dans des champs et à les gérer au travers des méthodes. Si on se rapproche du Pascal, un objet n'est donc qu'une extension évoluée des enregistrements (type record) disposant de procédures et fonctions pour gérer les champs qu'il contient [9].

IV.3.2 Objet et classe

Avec la notion d'objet, il convient d'amener la notion de classe. Cette notion de classe n'est apparue dans le langage Pascal qu'avec l'avènement du langage Delphi et de sa nouvelle approche de la Programmation Orientée Objet. Elle est totalement absente du Pascal standard.

Ce que l'on a pu nommer jusqu'à présent objet est, pour Delphi, une classe d'objet. Il s'agit donc du type à proprement parler. L'objet en lui-même est une instance de classe, plus simplement un

exemplaire d'une classe, sa représentation en mémoire. Par conséquent, on déclare comme type une classe, et on déclare des variables de ce type appelées des objets. Si cette distinction est à bien prendre en considération lors de la programmation en Delphi, elle peut toutefois être totalement ignorée avec la plupart des autres compilateurs Pascal. En effet, ceux-ci ne s'appuient que sur les notions d'objet et d'instance d'objet [9].

IV.3.3 Les trois fondamentaux de la POO

La Programmation Orientée Objet est dirigée par trois fondamentaux qu'il convient de toujours garder à l'esprit : encapsulation, héritage et polymorphisme.

IV.3.3.1 Encapsulation

Derrière ce terme se cache le concept même de l'objet : réunir sous la même entité les données et les moyens de les gérer, à savoir les champs et les méthodes. L'encapsulation introduit une nouvelle manière de gérer des données. Il ne s'agit plus de déclarer des données générales puis un ensemble de procédures et fonctions destinées à les gérer de manière séparée, mais bien de réunir le tout sous le couvert d'une seule et même entité. Si l'encapsulation est déjà une réalité dans les langages procéduraux (comme le Pascal non objet par exemple) au travers des unités et autres librairies, il prend une toute nouvelle dimension avec l'objet. En effet, sous ce nouveau concept se cache également un autre élément à prendre en compte : pouvoir masquer aux yeux d'un programmeur extérieur tous les rouages d'un objet et donc l'ensemble des procédures et fonctions destinées à la gestion interne de l'objet, auxquelles le programmeur final n'aura pas à avoir accès. L'encapsulation permet donc de masquer un certain nombre de champs et méthodes tout en laissant visibles d'autres champs et méthodes. Pour conclure, l'encapsulation permet de garder une cohérence dans la gestion de l'objet, tout en assurant l'intégrité des données qui ne pourront être accédées qu'au travers des méthodes visibles, la figure IV.1 donne le schéma de principe de l'encapsulation dans la POO. [9].

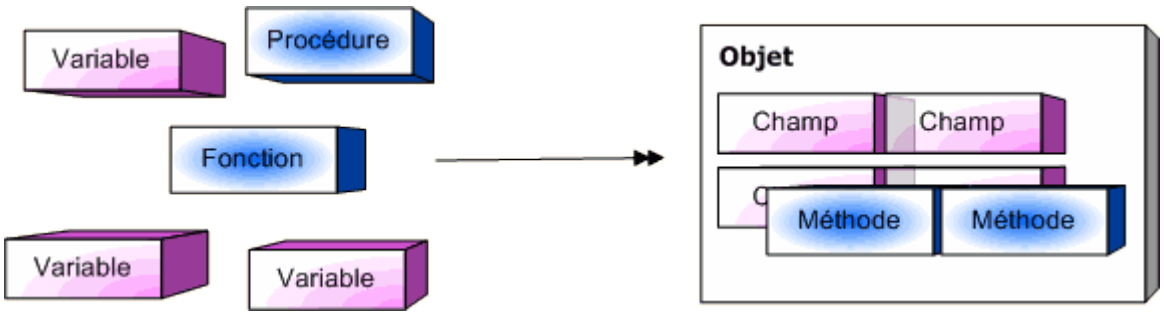

Figure IV.1 Encapsulation dans la POO.

IV.3.3.2 Héritage

Si l'encapsulation pouvait se faire manuellement (grâce à la définition d'une unité par exemple), il en va tout autrement de l'héritage. Grâce au concept d'héritage, un objet peut donner naissance à un ou des descendants. Ces descendants vont tous bénéficier des caractéristiques propres de leur ancêtre, à savoir ses champs et méthodes. Cependant, les descendants conservent la possibilité de posséder leurs propres champs et méthodes. Tout comme un enfant hérite des caractéristiques de ses parents et développe les siennes, un objet peut hériter des caractéristiques de son ancêtre, mais aussi en développer de nouvelles, ou bien encore se spécialiser comme le montre la figure IV.2.

Ce concept d'héritage ouvre donc la porte à un nouveau genre de programmation. On notera qu'une fois qu'un champ ou une méthode sont définis, il ou elle le reste pour tous les descendants, quel que soit leur degré d'éloignement [9].

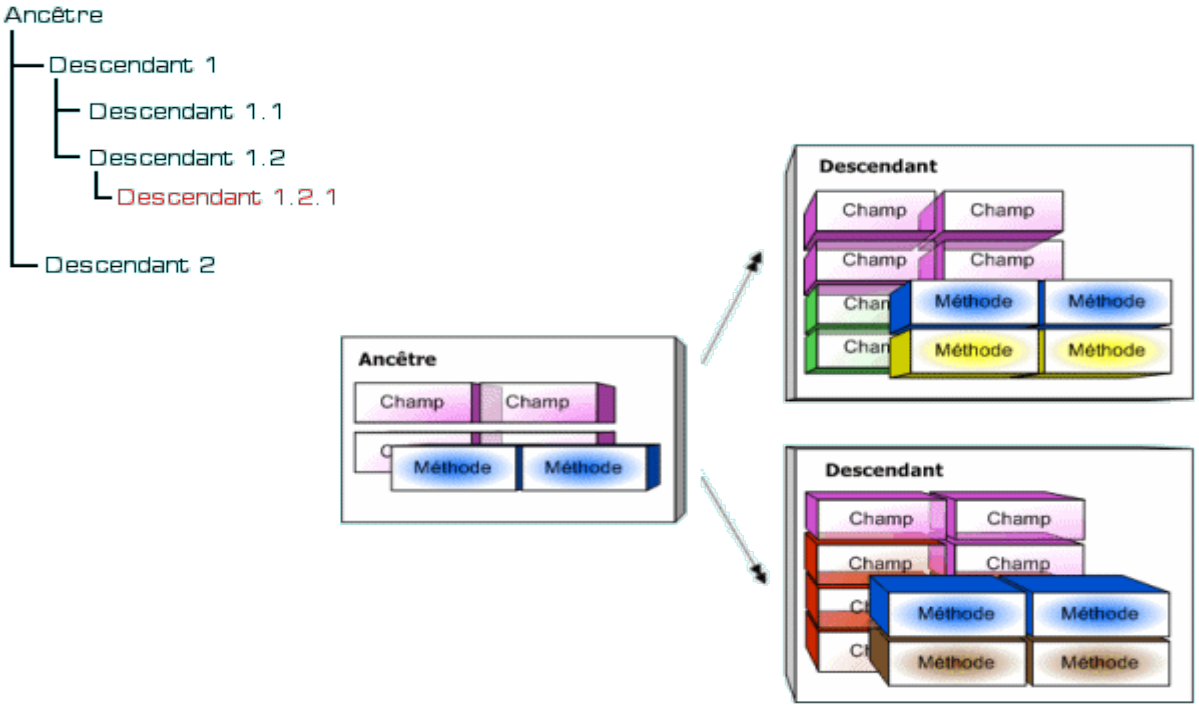

Figure IV.2 Héritage dans la POO.

IV.3.3.3 Polymorphisme

Le polymorphisme traite de la capacité de l'objet à posséder plusieurs formes. Cette capacité dérive directement du principe d'héritage vu précédemment. En effet, comme on le sait déjà, un objet va hériter des champs et méthodes de ses ancêtres. Mais un objet garde toujours la capacité de pouvoir redéfinir une méthode afin de la réécrire, ou de la compléter.

On voit donc apparaître ici ce concept de polymorphisme : choisir en fonction des besoins quelle méthode ancêtre appeler, et ce au cours même de l'exécution. Le comportement de l'objet devient donc modifiable à volonté. Le polymorphisme, en d'autres termes, est donc la capacité du système à choisir dynamiquement la méthode qui correspond au type réel de l'objet en cours dont la figure IV.3 montre le principe [9].

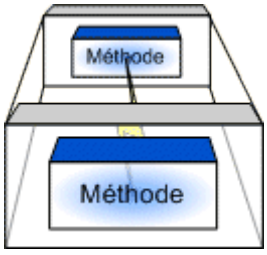

Figure IV.3 Polymorphisme dans la POO.

IV.4 Environnement de travail de DELPHI

L'écran de Delphi 7 se présente à l'ouverture comme sur la figure IV.4. Il affiche une fenêtre supérieure contenant une barre de menus et plusieurs barres d'outils, ainsi que trois fenêtres : Vue arborescente des objets, inspecteur d'objets et Form1.

Les menus reprennent la philosophie à laquelle nous sommes habitué dans Windows à la différence prêt qu'un menu projet comporte toutes les fonctions qui nous permettront de modifier, tester puis compiler nos objets afin de les rendre fonctionnels et opérationnels indépendamment de Delphi.

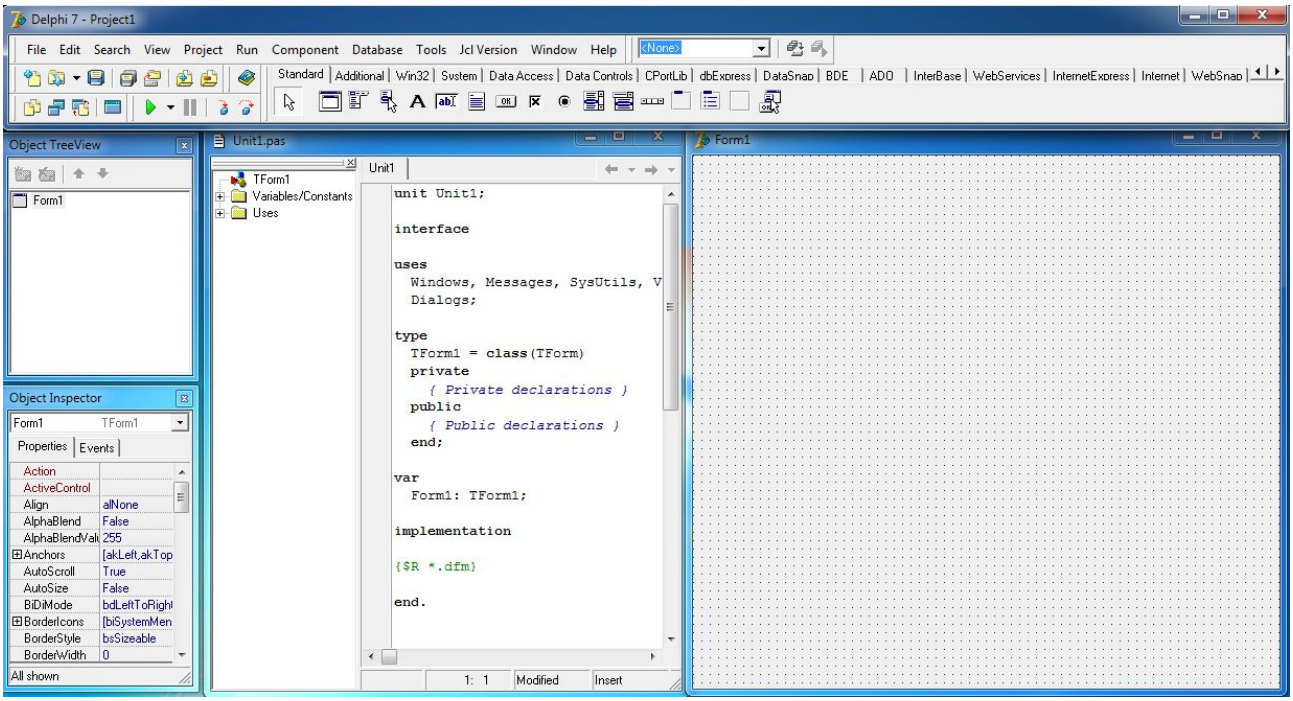

Figure IV.4 Environnement de travail de DELPHI.

Une barre d'outils comportant des onglets dont le premier s'intitule Standard comme le montre la figure IV.5 nous permettra de choisir et de placer directement dans nos programme des éléments déjà programmés tels que : Buttons, Edit, CheckBox, RadioButon, ComPort, Label, GroupBox, MaiMenu, Timer, etc.

Figure IV.5 Barre d'outils du DELPHI.

IV.5 Les éléments constituants l'application

IV.5.1 Fiche (Form)

Le composant Form1 est sous forme d'une fenêtre qui contient tout les autres composants de l'application (voir figure IV.6), Les fiches ne se créent pas depuis la palette des composants, mais elle est accessible depuis le menu. Elle dispose de plusieurs propriétés et d'événements modifiables directement par l'inspecteur d'objets.

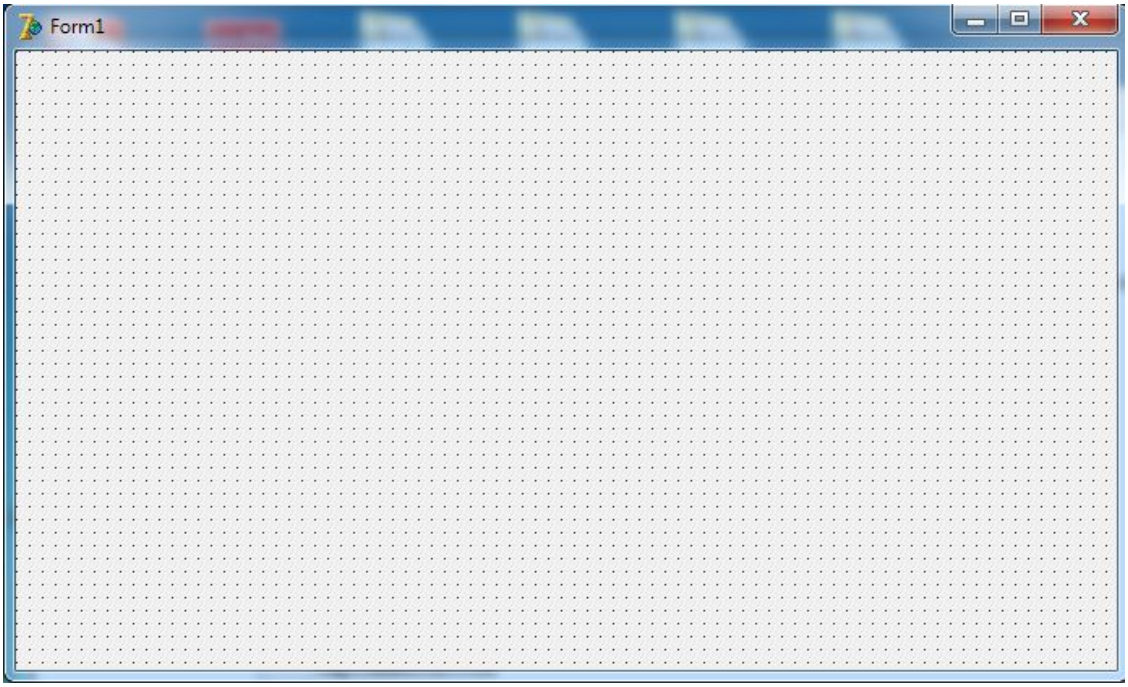

Figure IV.6 Composant Fiche.

Les propriétés utilisées : Caption pour le nom, icon pour attribuer au composant Form.

Les événements : OnShow : on lui a attribué l'événement MainMenu pour qu'il affiche le Menu.

IV.5.2 Menu principal (MainMenu)

Ce composant permet de munir la fiche d'une barre de menus déroulants. Ce composant est non visuel lorsqu'on vient de le poser sur une fiche : au lieu d'un composant visible, l'icône du composant vient se placer sur la fiche. A partir de son pseudo-bouton, on peut accéder via un double-clique à une interface spécifique de création de menus. Cette interface permet de créer directement de façon visuelle les menus en utilisant l'inspecteur d'objets et quelques raccourcis clavier. Cette interface permet en fait d'éditer la propriété "Items" du composant. La figure IV.7 illustre cette propriété. Dans notre cas on a utilisé ce composant pour afficher la fenêtre "à propos", et pour sauvegarder l'historique via le menu "Fichier".

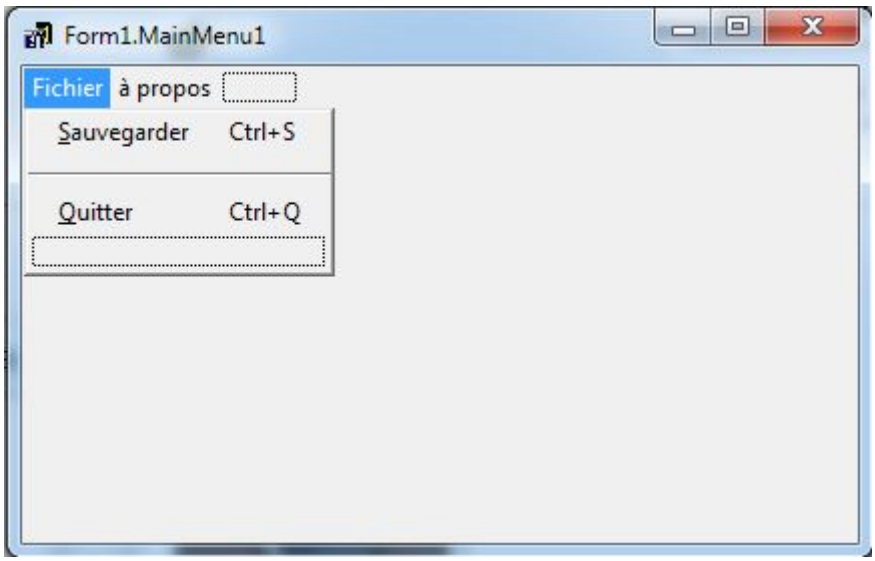

Figure IV.7 Interface MainMenu.

IV.5.3 Bouton (Button)

C’est le composant qui propose à l'utilisateur une action, cette action est expliquée par un texte sur le bouton comme l'indique la figure IV.8, lorsqu'on click dessus, l'événement OnClick se produit et offre une possibilité de réaction. C'est dans la procédure de réponse à cet événement qu'on effectue l'action proposée à l'utilisateur. Par exemple, dans notre cas, on a utilisé ces composants pour envoyer les requêtes au microcontrôleur de la station météo, configurer et activer le port RS232.

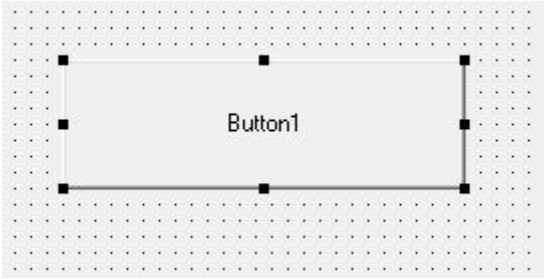

Figure IV.8 Composant Button.

On a utilisé pour ce composant la propriété Caption et l’événement OnClick.

IV.5.4 Connexion via le port COM (Comport)

Le comport est un composant appartenant à la librairie tierce appelée CportLib. Il permet d'intégrer le port série RS232 à l'application, et la gestion de la communication via ce dernier. On peut le paramétrer à notre guise comme le choix du port, la vitesse de transmission, le nombre de bits de données...etc, il est présenté par la figure IV.9 ainsi que sa libraire par la figure IV.10.

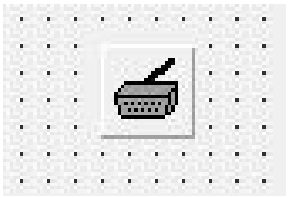

Figure IV.9: Composant Comport.

Figure IV.10: Librairie CportLib.

Pour ce composant, on a utilisé les propriétés BaudRate, Port, Parity, DataBits, StopBits et les événements : OnAfterClose, OnAfterOpen, et OnRxChar.

IV.5.5 Etiquette de texte (Label)

Il permet d'inclure facilement du texte sur une fiche. Ce texte n'est pas éditable par l'utilisateur comme l'illustre la figure IV.11, c'est un composant que l'on utilise souvent comme étiquette pour d'autres contrôles. Bien que l'on puisse modifier la police utilisée pour l'écriture du texte. Dans notre cas, on a utilisé ce composant pour afficher les unités des grandeurs, Date, Heure, l'état du port et l'état de la réception.

Figure IV.11: Composant Label.

On a utilisé les propriétés Caption et Font.

IV.5.6 Zone de saisie (Edit)

Il permet de proposer des zones d'édition tel qu'il est indiqué dans la figure IV.12. Ce composant permet à l'utilisateur d'entrer une information quelconque tapée au clavier. Le texte entré dans le composant est accessible via une propriété "Text". Il est possible de fixer une limite à la longueur du texte entré, de masquer les caractères (utile pour les mots de passe), de désactiver la zone ou d'interdire toute modification du texte. Dans notre cas, on a utilisé ce composant pour affecter et afficher les valeurs des grandeurs reçues.

Figure IV.12: Composant Edit.

Les propriétés utilisées sont : Text et Font.

IV.5.7 Boite de groupement (GroupBox)

Il a pour but d'effectuer des regroupements visibles de composants, il se présente sous la forme d'un cadre 3D et d'un texte en haut à gauche. Un composant GroupBox est un conteneur, donc susceptible de contenir d'autres composants. Au niveau du code, un composant GroupBox n'a pas souvent de rôle particulier à jouer, étant donné que c'est surtout son caractère conteneur qui importe. Aucune propriété, méthode ou événement n'est à signaler ici, cependant, il ne faut pas sous-estimer ce composant qui tient un grand rôle dans l'organisation tant visuelle que logique des interfaces les plus fournies. Dans notre cas, on a utilisé des GroupBox pour regrouper les composants de chaque grandeur (Température, Pression, Humidité et Vitesse du vent).

La figure IV.13 illustre ce composant .

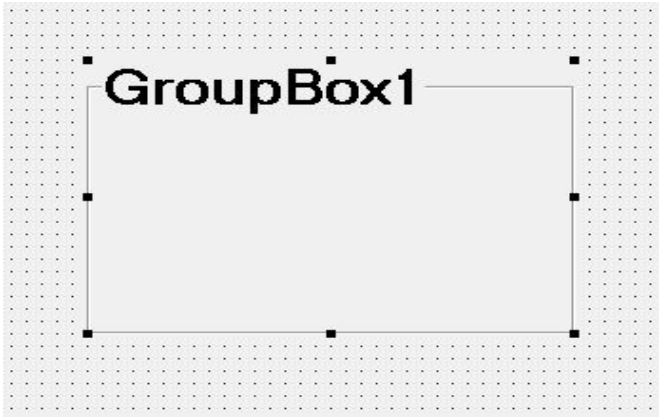

Figure IV.13: Composant GroupBox.

On a juste modifié la propriété Caption pour donner un nom aux groupes.

IV.5.8 Case à cocher (CheckBox)

Un composant CheckBox permet de donner à l'utilisateur un choix de type "Oui/Non". S'il coche la case, la réponse est "Oui", sinon, c'est "Non", comme le montre la figure IV.14. Au niveau du code, une propriété de type booléen stocke cette réponse : Checked. Il est possible de donner un texte explicatif de la case à cocher dans la propriété Caption. Ce texte apparait à coté de la case. Une modification de l'état de la case déclenche un événement OnClick, que cette modification provienne effectivement d'un clic ou d'une pression sur la touche Espace. Dans notre cas, on a utilisé ce composant pour activer la sauvegarde automatique de l'historique dans un fichier spécifique.

Figure IV.14: Composant ChekBox.

On a utilisé la propriété Caption et l'événement OnClick.

IV.5.9 Jauge (Gauge)

Il permet d'étendre une variable appelée "Progress" dans un intervalle limité par les valeurs max et min. Tel qu'il est illustré dans la figure IV.15. Dans notre cas, on l'a utilisé pour refléter l'image des valeurs comme une gauge réelle.

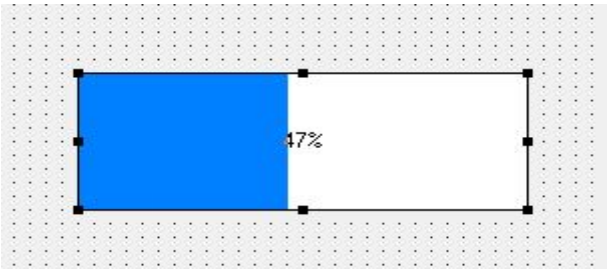

Figure IV.15: Composant Gauge.

On a changé les propriétés suivantes : MaxValue, MinValue, Name, Progress et Suffix.

IV.5.10 LED de visualisation (ComLed)

Elle appartient à la librairie CportLib, elle permet de prendre deux états logiques, allumée ou éteinte comme elle est indiquée dans la figure IV.16. Dans notre cas, on l'a utilisé comme témoin sur l'état du port et de la réception.

Figure IV.16: Composant ComLed.

On a juste utilisé les propriétés ComPort et Name.

IV.5.11 Compteur (Timer)

Il peut être utilisé comme un compteur ou un gestionnaire de temps. Dans notre cas, on l'a utilisé pour gérer la date et l'heure, la mise à jour automatique des grandeurs et la sauvegarde de l'historique dans un fichier spécifique. La figure IV.17 illustre ce composant.

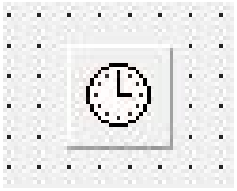

Figure IV.17: Composant Timer.

On a modifié les propriétés Enabled et Interval, et l'événement OnTimer

IV.6 Principe de fonctionnement de l'application

Lorsqu'on lance l'application, on choisit le numéro du port série que l'on souhaite utiliser en appuyant sur le bouton 'réglages', puis on click sur le bouton 'connecter' pour établir la liaison. Pour acquérir la valeur d'une grandeur, par exemple la température, on appuis sur le bouton 'recevoir la température'.

Les boutons sont programmés à envoyer les requêtes sous forme de caractère vers le microcontrôleur via le port série RS232. Le caractère '0' est attribué à la température, le '1' à humidité, le '2' à la pression et le '3' à la vitesse du vent. Le microcontrôleur est programmé à répondre à chaque cas possible, sa tâche est d'envoyer la grandeur correspondant au caractère reçu via le port RS232, cette grandeur sera récupérer via l'application et sera attribuée au composants Edit et Gauge. Pour arrêter la transmission un bouton est programmé pour envoyer le caractère 'A' au microcontrôleur, ce dernier n'étant pas programmé pour répondre cesse l'envoie par le port série. L'envoie et la réception suit les instructions de l'organigramme dans la figure IV.18.

IV.7 Organigramme de l'application

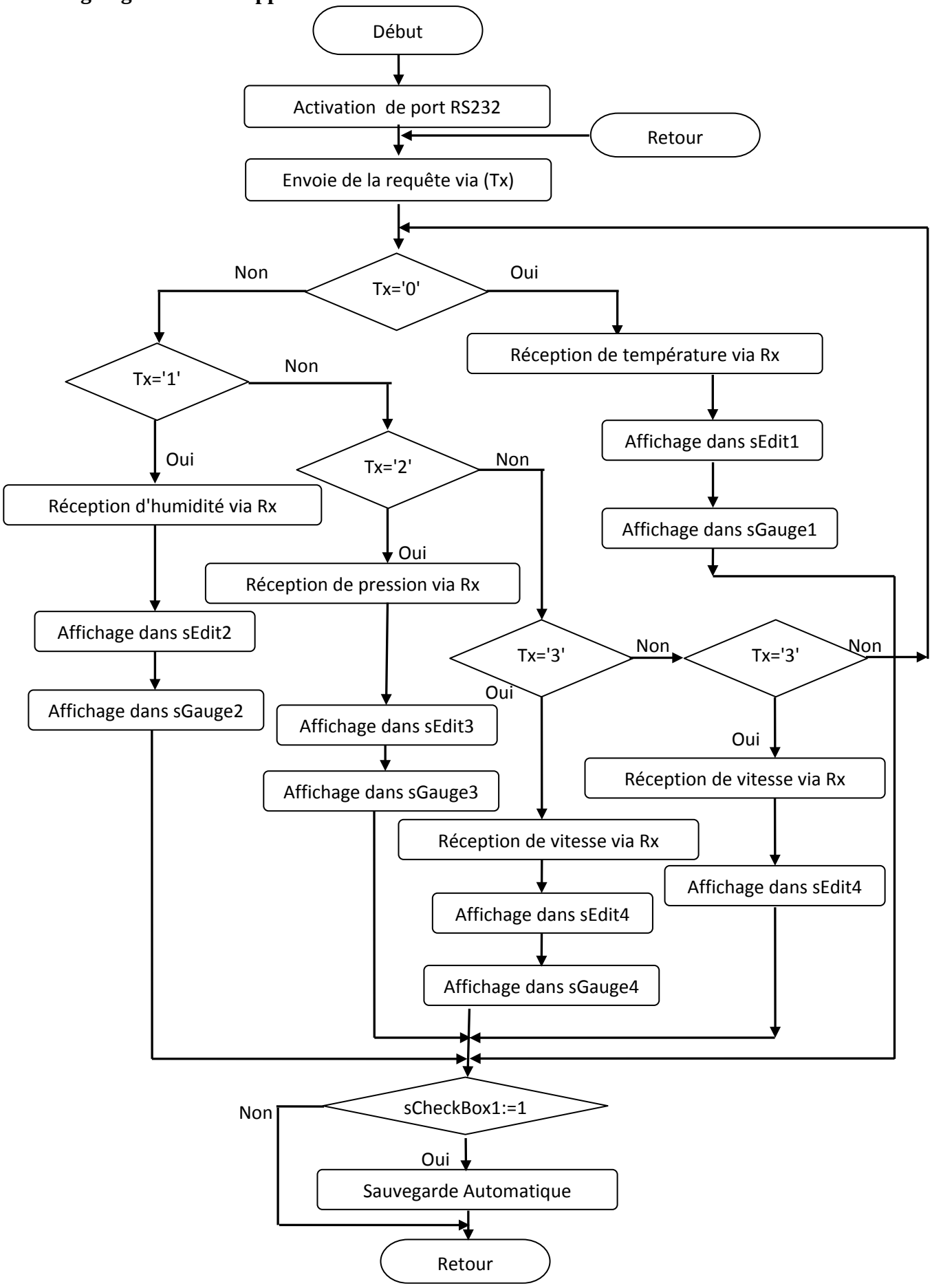

Figure IV.18: Organigramme de l'application

IV.8 Interface finale de l'application

La figure IV.19 illustre la version finale de l'application.

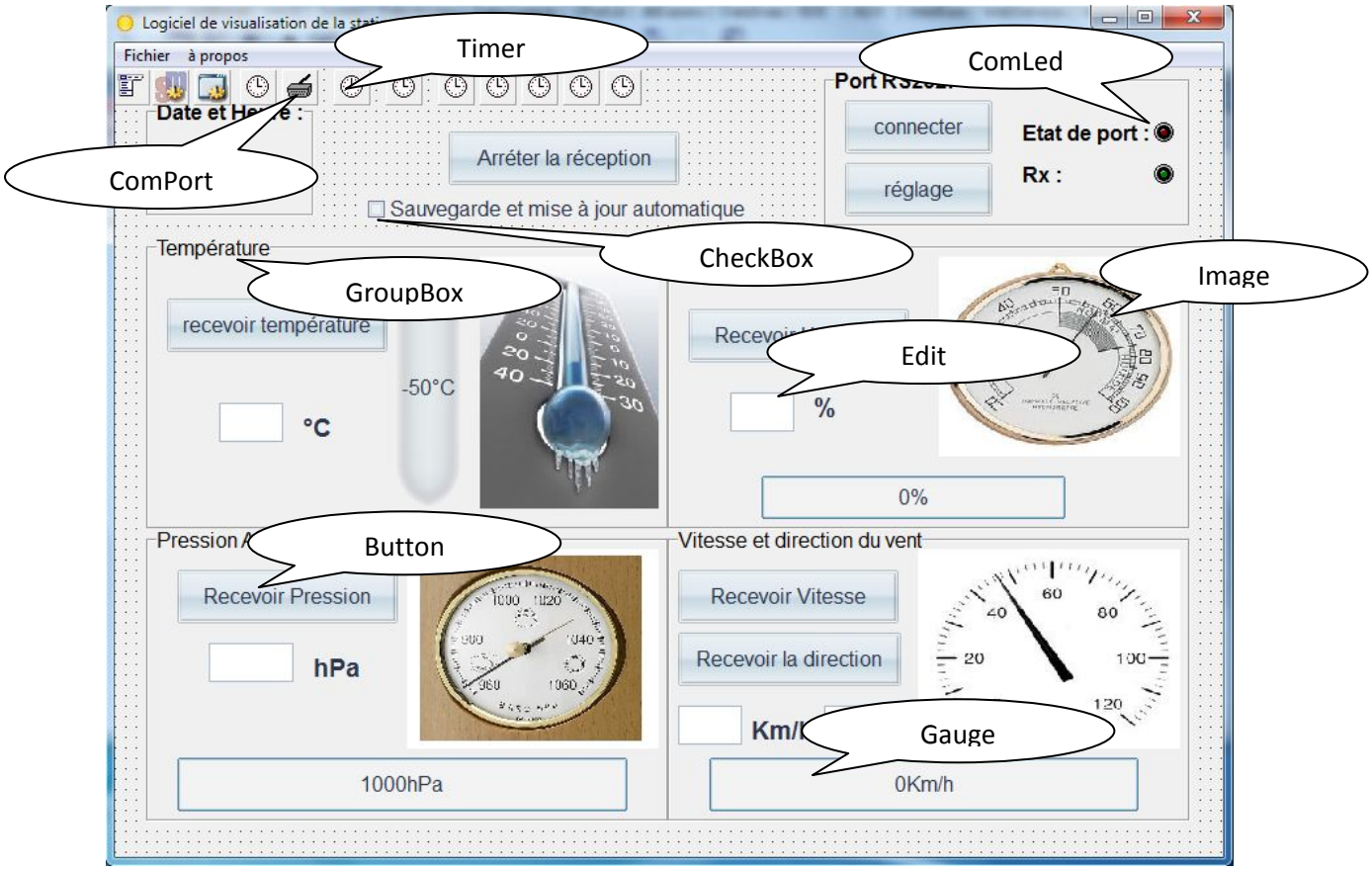

Figure IV.19: Application non compilée.

IV.9 Description de l'application

Une fois l'application lancée, son écran d'accueil apparait sous une forme intuitive comme l'illustre la figure IV.20, disposé en forme de quatre écrans distincts grâce au composant GroupBox qui représentent les espaces réservés pour chaque grandeur pour que l'utilisateur puisse se retrouver facilement.

Chaque espace contient une image qui correspond à l'instrument de mesure de la grandeur, un bouton programmé pour envoyer la requête vers le microcontrôleur via le port série RS232, un Label qui contient l'unité de la grandeur, un Edit qui affiche la valeur de la grandeur, et une gauge qui évolue entre une valeur max et une valeur min en fonction de la valeur de la grandeur.

Un GroupBox contenant deux boutons: un, configuré pour accéder aux paramètres du port RS232 et l'autre pour connecter le port voulu. Deux Led sont aussi programmées pour refléter l'état des boutons.

Un autre GroupBox nous permet d'acquérir la date et l'heure du système d'exploitation et de l'afficher sur l'application.

Un CheckBox programmé pour la sauvegarde et la mise à jour automatique des grandeurs.

Un menu 'à propos', et Fichier pour sauvegarder manuellement les grandeurs.

Figure IV.20: Application au démarrage.

La figure IV.21 montre l'application en fonctionnement.

Figure IV.21 Application en fonctionnement.

IV.10 Conclusion

Dans ce chapitre nous avons présenté l'application de visualisation sur PC des grandeurs mesurées. Cette application est réalisée avec le langage DELPHI. Nous avons décrit les étapes phares de sa conception et les éléments utilisés pour ce but, puis nous avons expliqué son principe de fonctionnement pour faciliter la tâche à l'utilisateur.

Il est à noter que pour la conception d'une application dédiée à être connectée vers le monde extérieur (carte à microcontrôleur), il est indispensable de faire appel à des bibliothèques de composants conçus spécialement pour cette fin. A cet effet, notre application permet cette connexion afin d'afficher et de stocker les informations (mesures) envoyées par la station météo à base de microcontrôleur.

Un autre point essentiel est le pourvoir d'activer la réception séparée des grandeurs via cette application.

Conclusion générale

L'objet de notre projet était de concevoir une station météo à base de microcontrôleur avec transmission de données via le port RS232.

Nous avons étudié les capteurs, et les différentes méthodes de mesure des grandeurs météorologique. Lors de la simulation, nous avons utilisé le capteur à semi-conducteur LM335 pour la température, le capteur piézo-résistif MPX4115 pour la pression atmosphérique, et des résistances variables pour l'humidité et la vitesse du vent, car ISIS ne dispose pas dans sa bibliothèque de ces capteurs.

La station météo contient un microcontrôleur PIC16F876 qui gère les différents éléments. Ce qui nous permet l'acquisition des grandeurs mesurées, puis leur affichage sur un afficheur LCD, et enfin l'interactivité avec le PC pour transmettre ces grandeurs et sauvegarder l'historique.

Nous avons effectué des simulations du circuit avec le logiciel ISIS, ce qui nous a aidé à étalonner et tester le conditionnement ainsi que la mise en œuvre du programme de gestion du microcontrôleur.

L'afficheur graphique LCD utilisé satisfait nos besoins, malgré son incapacité à afficher une chaine de caractère. Car il permet l'affichage de toutes les grandeurs et leurs unités sans problème sur quatre lignes de vingt caractères chacune.

La station météo conçue peut être réalisé sans aucun problème. Elle peut être utilisée pour l'acquisition des grandeurs météorologiques à des fins personnelles ou professionnelles (agriculteur, navigation maritime, aviation)

Des améliorations peuvent être apportées à ce travail, comme:

- ✓ L'ajout d'autre capteurs pour d'autre grandeurs.
- ✓ Exploiter d'autre type de protocole de communication (ETHERNET, WIFI...etc).
- ✓ Ajout des modules pour le traitement des données stockées via des graphiques ou par des fonctions mathématiques telle que la recherche des valeurs minimales, moyennes, et maximales des grandeurs mesurées dans des périodes spécifiques.

Bibliographie

[1] Georges Asch, Les capteurs en instrumentation industrielle, Dunod, Paris 2010.

[2] Datasheet LM335, MPX4115, PIC16F876,LM044L et P-14.

www.alldatasheet.com

[3] Technique de l'ingénieur, Mesure en météorologie, référence, 42540210-r3050.

Par : Pierre GRÉGOIRE et Michel LEROY.

[4] Christian Tavernier, Les microcontrôleurs PIC Description et mise en œuvre, Dunod, Paris 2002.

[5] Pierre-Jean Bellavoine, Delphi 7, Dunod, Paris 2003.

[6] Guy Isabel, Construire ses capteur météo, ETSF, PARIS 1994.

[7] CCS Compiler Reference Manual

http://www.ccsinfo.com/download.shtml

[8] Bigonoff, La programation et description des PICs 16F87x

www.abcelectronique.com/bigonoff/

[9] Bertrand Meyer, Conception et programmation orientées objet, Eyrolles,1997

Annexe 1 Code source du microcontrôleur.

```
#include <16F876.h>
#device adc=10
#fuses HS,NOWDT,NOPROTECT,NOLVP
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, PARITY=N, STREAM=1)
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
//***************************************************************************
#define rs PIN_C1
#define rw PIN_C2
#define e  PIN_C0
#define D0 PIN_B0
#define D1 PIN_B1
#define D2 PIN_B2
#define D3 PIN_B3
#define D4 PIN_B4
#define D5 PIN_B5
#define D6 PIN_B6
#define D7 PIN_B7
#define rcv PIN_C7
//***************************************************************************
void lcd_init();
void lcd_display (char);
void lcd_temperature();
void lcd_pression();
void lcd_humidite();
void lcd_vent();
void lcd_commande (char);
void negatif();
void positif();
void nul();
//**********************initialisation de l'afficheur LCD***************
void lcd_init()
{
   output_low(rs);
   output_low(rw);
   output_low(e);
   lcd_commande(0x33);//LCD en mode 8 bits
   lcd_commande(0x38);
   lcd_commande(0x0E);//curseur visible
```

```
    lcd_commande(0x06);//ecriture de gauche vers la droite
    lcd_commande(0x01);
    delay_ms(10);
}
//*****************************************************************************
void lcd_commande (char x)
{
   output_low(rs);
   output_low(rw);
   output_b(x);
   output_high(e);
   delay_us(1);
   output_low(e);
   delay_ms(10);
}
//*****************************************************************************
 float t,p,v1,h,dv;float valnum0,valnum1,valnum2,valnum3,valnum4;float
volt0,volt1,volt2,volt3,volt4;float temp,pres,hum,vent,divent;
int Ta,Tb,Tc,Ha,Hb,Hc,Pa,Pb,Pc,Pd,Va,Vb,Vc,dv1; char b;
 char V[20]={'V','e','n','t',':'};
void lcd_temperature()
 {
char T[19]={'T','e','m','p','e','r','a','t','u','r','e',':'};
int i=0;
 for (i=0;i<20;i++)
 {
T[12]=Ta+48;
T[13]=Tb+48;
T[14]=Tc+48;
T[15]=' ';
T[16]='C';
lcd_display(T[i]);
}
}
void lcd_humidite()
 { char H[19]={'H','u','m','i','d','i','t','e',' ',':'};
int l=0;
 for (l=0;l<20;l++)
 {
H[12]=Ha+48;
H[13]=Hb+48;
H[14]=Hc+48;
H[15]=' ';
H[16]='%';
```

```
lcd_display(H[l]);
}
}
void lcd_pression()
 { char P[20]={'P','r','e','s','s','i','o','n',' ',':',' '};
int k=0;
 for (k=0;k<20;k++)
 {
P[11]=Pa+48;
P[12]=Pb+48;
P[13]=Pc+48;
P[14]=Pd+48;
P[16]='h';
P[17]='P';
P[18]='a';
lcd_display(P[k]);
}
}
void lcd_vent()
 {
int j=0;
 for (j=0;j<20;j++)
 {
V[12]=Va+48;
V[13]=Vb+48;
V[14]=Vc+48;
V[16]='K';
V[17]='m';
V[18]='/';
V[19]='h';
lcd_display(V[j]);
}
}
void lcd_display (char x)
{
   output_high(rs);
   output_low(rw);
   output_b(x);
   output_high(e);
   output_low(e);
   delay_us(10);
}
void positif ()
{
```

```
if (t<10) {Ta=-16; Tb=-16; Tc=t;}
else if (10<=t<=100) {Ta=-16; Tb=t/10;Tb=floor(Tb);Tc=t-Tb*10;}
if (t>=100)  {Ta=t/100; Ta=floor(Ta);Tb=t-100; Tb=Tb/10;Tb=floor(Tb);Tc=t-100-(Tb*10);}
}
void positif1 ()
{
 if (p>100)  {Pa=0; Pb=p/100; Pb=floor(Pb);Pc=p-Pb*100; Pc=Pc/10;Pc=floor(Pc);Pd=p-
Pb*100-(Pc*10);}
 if (p>=1000)
{Pa=p/1000; Pa=floor(Pa); Pb=(p-(Pa*1000)); Pb=Pb/100; Pb=floor(Pb); Pc=(p-(Pa*1000)-
(Pb*100));
Pc=Pc/10; Pc=floor(Pc); Pd=(p-(Pa*1000)-(Pb*100)-(Pc*10));}
}
void positif2 ()
{
if (h<10) {Ha=-16; Hb=-16; Hc=h;}
else if (10<=h<=100) {Ha=-16; Hb=h/10;Hb=floor(Hb);Hc=h-Hb*10;}
if (H>=100)  {Ha=h/100; Ha=floor(Ha);Hb=h-100; Hb=Hb/10;Hb=floor(Hb);Hc=h-100-
(Hb*10);}
}
void positif3 ()
{
if (v1<10) {Va=-16; Vb=-16; Vc=v1;}
else if (10<=v1<=100) {Va=-16; Vb=v1/10;Vb=floor(Vb);Vc=v1-Vb*10;}
if (v1>=100)  {Va=v1/100; Va=floor(Va);Vb=v1-Va*100; Vb=Vb/10;Vb=floor(Vb);Vc=v1-
Va*100-(Vb*10);}
}
void negatif()
{
t=fabs(t);
if (t<10)
{Ta=-16; Tb=-3; Tc=t;}
else if (10<=t<100) {Ta=-3; Tb=t/10;Tb=floor(Tb);Tc=t-Tb*10;}
}
void nul()
{
Ta=-16; Tb=-16; Tc=0;
Pa=-16; Pb=-16; Pc=-16; Pd=0;
Ha=-16; Hb=-16; Hc=0;
Va=-16; Vb=-16; Vc=0;
}
void main ()
    {  SET_TRIS_B(0);
       SET_TRIS_C(0b10000000);
```

```
//************CONVERTISSEUR ANALOGIQUE NUMERIQUE*******************
  setup_adc_ports(ALL_ANALOG);
  setup_adc(ADC_CLOCK_INTERNAL);

//************acquisition de la température via le canal 0 de l'ADC*******************
while(1)
{
     delay_ms(1000);
//*****************************acquisition de la température**********************
          set_adc_channel(0);
          delay_us(1);
          valnum0=read_adc();
          volt0=(valnum0/1024)*5;
          temp=(volt0/0.009916)-273.15;
     temp=ceil(temp);
     t=ceil(temp);

//*****************************acquisition de la pression**************************
          set_adc_channel(1);
          delay_us(1);
          valnum1=read_adc();
          volt1=(valnum1/1024)*5;
          pres=(volt1*244.07);
     p=ceil(pres);

//*****************************acquisition de l'humidité**************************
          set_adc_channel(2);
          delay_us(1);
          valnum2=read_adc();
          volt2=(valnum2/1024)*5;
          hum=(volt2/0.05);
     h=ceil(hum);

//*****************************acquisition de la vitesse du vent*******************
          set_adc_channel(3);
          delay_ms(10);
          valnum3=read_adc();
          volt3=(valnum3/1024)*5;
          vent=(volt3*25);
     v1=ceil(vent);
//*****************************acquisition de la direction du vent*******************
          set_adc_channel(4);
          delay_ms(10);
          valnum4=read_adc();
```

```
            volt4=(valnum4/1024)*5;
            divent=(volt4);
      dv=divent+0.01;
dv1=floor(dv);
switch(dv1){
case 1: {V[6]='N';V[7]='o';V[8]='r';V[9]='d';V[10]=' '; }
break;
case 2: {V[6]='E';V[7]='s';V[8]='t';V[9]=' ';V[10]=' ';}
break;
case 3: {V[6]='S';V[7]='u';V[8]='d';V[9]=' ';V[10]=' ';}
break;
case 4: {V[6]='O';V[7]='u';V[8]='e';V[9]='s';V[10]='t';}
break;
}
//*******************************afichage************************************
      if (t>0){positif();}
            if (t<0){negatif();}
            if (t==0){nul();}
      delay_us(1);
      if (p>=0){positif1();}
            if (p<0){nul();}
      delay_us(1);
      if (h>0){positif2();}
            if (h==0){nul();}
      delay_us(1);
      if (v>0){positif3();}
            if (v==0){nul();}
      lcd_init();
      lcd_temperature();
      delay_us(1);
      lcd_commande(0x94);
      lcd_pression();
      delay_us(1);
      lcd_commande(0xC0);
      lcd_humidite();
      delay_us(1);
      lcd_commande(0xD4);
      delay_us(1);
      lcd_vent();
//****************L'envoie des données via le port RS232**************************
if (kbhit(1))
   {b = getc();}
delay_ms(1);
       if (b==0x30){printf("%f"temp);}
```

```
        if (b==0x31){printf("%f"h);}
                if (b==0x32){printf("%f"p);}
                if (b==0x33){printf("%f"v1);}
        if (b==0x34){printf("%c %c %c %c %c"V[6],V[7],V[8],V[9],V[10]);}
}
}
```

Annexe 2 Code source de l'application DELPHI.

```
unit Unit1;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, Menus, ExtCtrls, jpeg, StdCtrls, ComCtrls, sSkinProvider,
 sSkinManager, sEdit, DB, CPortCtl, CPort, acProgressBar, sButton, sLabel,
 sGauge, sCheckBox;
 type
 TForm1 = class(TForm)
 MainMenu1: TMainMenu;
 fi1: TMenuItem;
 sauvegarder1: TMenuItem;
 quitter1: TMenuItem;
 N1: TMenuItem;
 sSkinManager1: TsSkinManager;
 sSkinProvider1: TsSkinProvider;
 N2: TMenuItem;
 ComPort: TComPort;
 GroupBox2: TGroupBox;
 ComLed2: TComLed;
 Label2: TLabel;
 RxChar: TComLed;
 Label3: TLabel;
 GroupBox3: TGroupBox;
 Timer1: TTimer;
 Label5: TLabel;
 Label6: TLabel;
 GroupBox4: TGroupBox;
 GroupBox6: TGroupBox;
 GroupBox7: TGroupBox;
 GroupBox5: TGroupBox;
 sButton1: TsButton;
 sButton2: TsButton;
 sButton3: TsButton;
 sButton4: TsButton;
 Timer2: TTimer;
 Image2: TImage;
 sEdit1: TsEdit;
 Image3: TImage;
 sEdit2: TsEdit;
 sLabel1: TsLabel;
 Image4: TImage;
 sEdit3: TsEdit;
```

```
sLabel2: TsLabel;
Image5: TImage;
sEdit4: TsEdit;
sLabel3: TsLabel;
sLabel4: TsLabel;
sGauge1: TsGauge;
sGauge2: TsGauge;
sGauge3: TsGauge;
sGauge4: TsGauge;
sCheckBox1: TsCheckBox;
Button_open: TsButton;
Button2: TsButton;
Timer3: TTimer;
Timer4: TTimer;
Timer5: TTimer;
Timer6: TTimer;
Timer7: TTimer;
sButton5: TsButton;
sButton6: TsButton;
sEdit5: TsEdit;
Timer8: TTimer;
procedure quitter1Click(Sender: TObject);
procedure N1Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure ComPortOpen(Sender: TObject);
procedure ComPortClose(Sender: TObject);
procedure Button_openClick(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Timer1Timer(Sender: TObject);
procedure sauvegarder1Click(Sender: TObject);
procedure ComPortRxChar(Sender: TObject; Count: Integer);
procedure sButton1Click(Sender: TObject);
procedure sButton2Click(Sender: TObject);
procedure sButton3Click(Sender: TObject);
procedure sButton4Click(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure sCheckBox1Click(Sender: TObject);
procedure Timer3Timer(Sender: TObject);
procedure Timer4Timer(Sender: TObject);
procedure Timer5Timer(Sender: TObject);
procedure Timer6Timer(Sender: TObject);
procedure Timer2Timer(Sender: TObject);
procedure Timer7Timer(Sender: TObject);
procedure sButton5Click(Sender: TObject);
```

```
 procedure sButton6Click(Sender: TObject);
 procedure Timer8Timer(Sender: TObject);
 private
 { Private declarations }
 public
 { Public declarations }
 end;
 var
 Form1: TForm1;
 t: char;
implementation
uses MONCARNETAPROPOS;
{$R *.dfm}
procedure TForm1.quitter1Click(Sender: TObject);
begin
form1.Close;
ComPort.Close;
end;
procedure TForm1.N1Click(Sender: TObject);
begin
Aboutbox.showmodal;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
 ComPort.ShowSetupDialog;
end;
procedure TForm1.Button_openClick(Sender: TObject);
begin
 if ComPort.Connected then
 begin
Timer3.Enabled:=False;
Timer4.Enabled:=False;
Timer5.Enabled:=False;
Timer6.Enabled:=False;
Timer7.Enabled:=False;
ComPort.Close;
end
else
ComPort.Open;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
ComPort.ShowSetupDialog;
end;
```

```
procedure TForm1.ComPortOpen(Sender: TObject);
begin
Button_Open.Caption := 'déconnecter';
end;
procedure TForm1.ComPortClose(Sender: TObject);
begin
if Button_Open <> nil then
Button_Open.Caption := 'connecter';
end;
procedure TForm1.Timer1Timer(Sender: TObject);
begin
Label5.Caption := DateToStr(Date);
Label6.Caption := TimeToStr(Time);
end;
procedure TForm1.sauvegarder1Click(Sender: TObject);
var
Sauv: TextFile;
begin
 AssignFile(Sauv, 'c:\historique.txt');
 if not FileExists('c:\historique.txt') then
 begin
 Rewrite(Sauv);
 end
 else
 Append(Sauv);
 Writeln(Sauv,'-----------------------------------' );
 Writeln(Sauv,'Date :                  '+Label5.Caption );
 Writeln(Sauv,'Heure :                 '+Label6.Caption );
 Writeln(Sauv,'Température :           '+sEdit1.text+' °C');
 Writeln(Sauv,'Humidité :              '+sEdit2.text+' %');
 Writeln(Sauv,'Pression Atmosphérique : '+sEdit3.text+' hPa' );
 Writeln(Sauv,'Vitesse du vent :       '+sEdit4.text+' Km/s');
 Writeln(Sauv,'Direction du vent :     '+sEdit5.text);
 CloseFile(Sauv);
 end;
procedure TForm1.ComPortRxChar(Sender: TObject; Count: Integer);
var
 Str,c: String;
 a: Integer;
 b:real;
begin
 ComPort.ReadStr(Str,9);
  case t of
  '0':
```

```
begin
b:=StrTOfloat(Str);
a:=trunc(b);
sEdit1.Text:=IntTOstr(a);
sGauge1.Progress:=a;
end;
 '1':
 begin
 b:=StrTOfloat(Str);
 a:=trunc(b);
 sEdit2.Text:=intTOstr(a);
 sGauge2.Progress:=a;
 end;
 '2':
 begin
 b:=StrTOfloat(Str);
 a:=trunc(b);
 sEdit3.Text:=intTOstr(a);
 sGauge3.Progress:=a;
 end;
 '3':
 begin
 b:=StrTOfloat(Str);
 a:=trunc(b);
 sEdit4.Text:=intTOstr(a);
 sGauge4.Progress:=a;
 end;
 '4':
 begin
 sEdit5.Text:=(str);
 end;
 end;
 end;
 procedure TForm1.sButton1Click(Sender: TObject);
 begin
 t:='0';
 Comport.WriteStr(t);
 end;
 procedure TForm1.sButton2Click(Sender: TObject);
 begin
 t:='1';
 Comport.WriteStr(t);
 end;
 procedure TForm1.sButton3Click(Sender: TObject);
```

```
begin
t:='2';
Comport.WriteStr(t);
end;
procedure TForm1.sButton4Click(Sender: TObject);
begin
t:='3';
Comport.WriteStr(t);
end;
procedure TForm1.FormShow(Sender: TObject);
begin
Decimalseparator:='.';
end;
procedure TForm1.sCheckBox1Click(Sender: TObject);
begin
sCheckBox1.Checked:=true;
if sCheckBox1.Checked then
begin
Timer3.Enabled:=true;
Timer2.Enabled:=True;
end;
end;
procedure TForm1.Timer3Timer(Sender: TObject);
begin
sButton1.Click;
Timer3.Enabled:=False;
Timer4.Enabled:=True;
end;
procedure TForm1.Timer4Timer(Sender: TObject);
begin
sButton2.Click;
Timer4.Enabled:=False;
Timer5.Enabled:=True;
end;
procedure TForm1.Timer5Timer(Sender: TObject);
begin
sButton3.Click;
Timer5.Enabled:=False;
Timer6.Enabled:=True;
end;
procedure TForm1.Timer6Timer(Sender: TObject);
begin
sButton4.Click;
Timer6.Enabled:=False;
```

```
 Timer7.Enabled:=True;
 end;
 procedure TForm1.Timer2Timer(Sender: TObject);
 begin
 sauvegarder1.Click;
 end;
procedure TForm1.Timer7Timer(Sender: TObject);
begin
sButton6.Click;
Timer7.Enabled:=False;
Timer8.Enabled:=True;
end;
procedure TForm1.sButton5Click(Sender: TObject);
begin
ComPort.WriteStr('A');
sCheckBox1.Checked:=False;
comport.FlowControl.XonXoffIn:=False;
end;
procedure TForm1.sButton6Click(Sender: TObject);
begin
t:='4';
Comport.WriteStr(t);
end;
procedure TForm1.Timer8Timer(Sender: TObject);
begin
Timer8.Enabled:=False;
Timer3.Enabled:=True;
end;
end.
```

Une mini station météo piloté par un microcontrôleur (PIC16F876). Son rôle est de mesurer la température, l'humidité, la pression atmosphérique, la direction et la vitesse de vent, ces grandeurs vont être affichées sur un écran LCD, et au même temps seront transmises au PC via un port RS232.

Dans le premier chapitre, on a abordé les généralités sur les capteurs météorologiques, pour comprendre les types et les variantes de ces derniers pour choisir les plus adaptés.

Le deuxième chapitre traite les informations relatives au microcontrôleur (PIC16F876), et ses périphériques. On a pu apprendre à agir sur ses registres pour le configurer à notre volonté. Et aussi les méthodes qui vont nous permettre de configurer et d'afficher sur l'afficheur LCD.

Le troisième chapitre est consacré à la conception de la station météo sous le logiciel ISIS PROTEUS. Cela va nous permettre de voir le comportement des différents composants électronique utilisés, y compris le microcontrôleur PIC16F876 lors de la simulation. Où on a pouvoir faire des tests adéquats.

Dans le dernier chapitre qui est dédié à l'application de visualisation via le PC, on a d'abord présenté le principe de la programmation orientée objets. Puis on a décrit le langage de programmation qui nous a permis de la réaliser, qui n'est autre que DELPHI. Ensuite, on a expliqué les étapes importantes pour sa réalisation. Et enfin on a présenté son interface, et effectué des tests et des simulation, pour faciliter son exploitation à l'utilisateur final, et pouvoir apporter des améliorations souhaitées