SOMMAIRE :

I. INTRODUCTION……3
I.1. Principe de fonctionnement des muscles artificiels pneumatiques.....3
I.2. Objectifs.....9

II. DIMENSIONNEMENT DE LA STUCTURE MECANIQUE……11
II.1. Introduction……...11
II.2. Calcul des efforts internes……11
II.3. Choix du profil et du matériau……...……13
II.4. Vérification de la tenue des poteaux……14
II.5. Calcul de la charge critique de flambement……..17
II.6. Calcul des flèches……..18
II.7. Assemblage des éléments……..19
II.8. Réalisation pratique du portique……....19

III. CAPTEURS ET VANNE……21
III.1. Mesure de la pression……21
a. Capteur……...21
b. amplification et conversion A/D……...22
- *Amplification*……23
- *conversion A/D*……..27
- *Alimentation et tension de référence*……29
c. Réalisation du circuit imprimé…….....29
d. Calibration du capteur……...……30
III.2. Mesure de la contraction……...32
a. Comparaison et choix de capteurs……..32
b. Méthode de mesure……35
c. Calibration du capteur……36
III.3. Vanne pneumatique

IV. PROGRAMMATION DU MICROCONTROLEUR MOTOROLA 68HC11….38
IV.1. Introduction……....38
IV.2. Principes et rôle du microcontrôleur……..38
IV.3. Introduction à la programmation……...……44
IV.4. Programme en assembleur……47
a. Introduction……...47
b. Organigrammes……..48
c. Communication SCI……...54
d. Communication SPI……...56
e. Timer……..58

V. INTERFACE VISUAL BASIC……....62
V.1. Introduction à Visual Basic……...…62
V.2. Partie visuelle……63
V.3. Programme……69

VI. CONCLUSION……....76

VII. ANNEXES……...78
VII.1. Informations Complémentaires……79
- Robot Lucy……...80
- Pièces de fixation du muscle artificiel……..81
- Calibration du capteur de pression……...83
- Tableaux comparatifs des capteurs de contraction……...84
- Plan 2D des pièces usinées……...85
- Calibration du capteur de contraction……...89
- Description générale des microcontrôleurs……..90
- Interface Visual Basic……...92
VII.2. Programmes……...…....94
- Programme Assembleur……...95
- Interface Visual Basic……..96
VII.3. Données techniques……..97

VIII. BIBLIOGRAPHIE……..98

I. INTRODUCTION

I.1. Principe de fonctionnement des muscles artificiels pneumatiques :

Les muscles artificiels pneumatiques, développés à la VUB (Vrije Universiteit Brussel), présentent de multiples caractéristiques intéressantes [DAERDEN].
Ces actionneurs ont été conçus à la base dans le cadre du robot bipède Lucy (voir Annexes page 80). Mais leur utilisation a été étendue à d'autres applications, comme par exemple dans le but d'actionner un robot manipulateur à trois degrés de liberté.

En pratique, chaque type d'actionneur (pneumatique, hydraulique, électrique) présente ses propres avantages et inconvénients. Certains seront donc plus adaptés que d'autres suivant l'application pour laquelle ils sont utilisés.

Les muscles artificiels pneumatiques plissés sont des organes gonflables. Ils développent un mouvement et une force unidirectionnels mais qui, lorsqu'ils sont associés par deux à une articulation, peuvent effectuer des mouvements bidirectionnels.

Ces actionneurs pneumatiques présentent les caractéristiques suivantes :

- Un rapport puissance/poids de l'ordre de 1kW/kg. Par comparaison les actionneurs électriques classiques présentent un rapport de l'ordre de 100W/kg. A puissance égale, les actionneurs électriques sont donc beaucoup plus lourds que nos actionneurs pneumatiques.

- Une capacité de fonctionner sur une large plage de pression. Il est alors possible d'exercer de faibles ou d'importantes forces de traction.

- Les muscles artificiels pneumatiques plissés peuvent réaliser un déplacement allant jusqu'à 50% de la taille initiale de la membrane de l'actionneur.

- Ces actionneurs présentent une caractéristique Force-Déplacement non linéaire et décroissante, ainsi qu'un comportement similaire à un ressort non linéaire.

- Enfin, du point de vue de la sécurité, les muscles artificiels pneumatiques ne présentent pas de risques d'explosion ou d'inflammation.

Le principe de fonctionnement est le suivant :
les muscles artificiels pneumatiques sont des actionneurs contractants dont la partie principale est une membrane qui se gonfle. La membrane est constituée d'un tissu de polymère imperméabilisé. On retrouve également des cordes de polymère de type para-aramide parcourant le muscle de bout à bout. Celles-ci reprennent l'effort et soutiennent la charge. Lorsqu'ils sont gonflés, ces actionneurs pneumatiques se rétrécissent et développent une force de traction.

La figure 1.1 montre le muscle dans son état de fonctionnement et dans son état de repos.

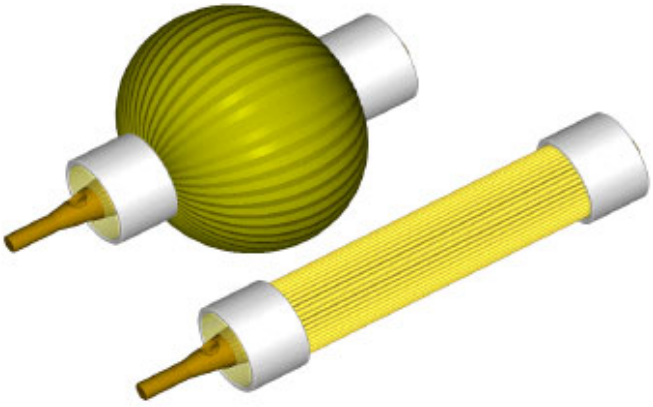


Fig. 1.1 : Muscle à l'état de fonctionnement et de repos

La membrane présente une grande résistance à la traction et est extrêmement flexible.
Cette membrane est plissée uniformément dans le sens de la longueur ce qui lui permet de contenir l'excédant déploiera au gonflage.
Aux deux extrémités, la membrane est maintenue par deux pièces qui contiennent les conduits d'alimentation et d'évacuation d'air comprimé.

De par la disposition des plis au niveau de la membrane, l'influence de la friction et l'hystérésis qui en découle sont négligeables.

Les muscles artificiels pneumatiques plissés, développés à la VUB, ne sont pas les seuls actionneurs de ce type, le muscle de McKibben en est l'exemple.

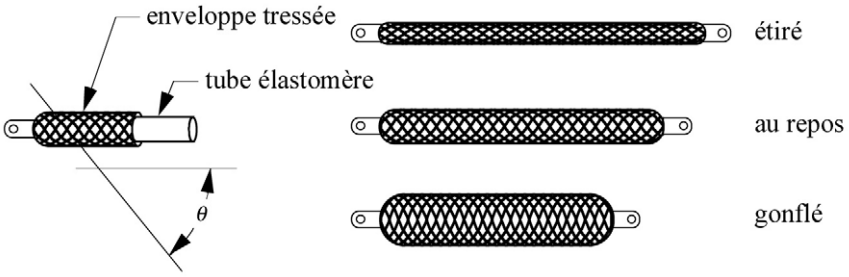


Fig 1.2 : Muscle de McKibben

Celui-ci est constitué d'un tube en élastomère entouré d'un tressage synthétique. L'avantage de ce type d'actionneur est sa simplicité de réalisation. Mais le muscle de McKibben présente néanmoins un inconvénient majeur :

- La présence de frottement entre le tube de caoutchouc et la tresse synthétique entraîne une hystérésis dans la caractéristique Force-Allongement. Ceci a pour conséquence un effet néfaste sur le comportement de l'actionneur.

Voici une représentation de l'hystérésis du muscle de McKibben pour différents niveaux de pression [CHOU]:

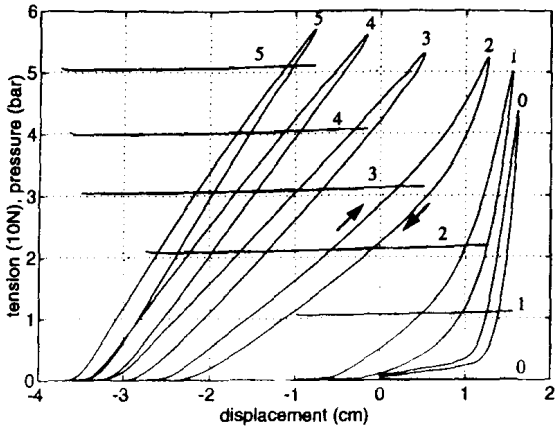


Fig. 1.3 : Hystérésis du muscle de McKibben

- Le muscle de McKibben présente un déplacement réduit par rapport aux actionneurs pneumatiques plissés. En effet il est capable de développer un déplacement total de l'ordre de 20 à 30% de la taille initiale.

- Pour le muscle de McKibben, le seuil de pression minimale pour avoir un effet utile notable est plus élevé que celui du muscle artificiel plissé. Il vaut en général 1 bar. Tandis que pour le muscle plissé ce seuil est inférieur à 0,1 bar. Ce dernier pourra donc fonctionner pour de très faibles valeurs de pression.

- La déformation de l'élastomère nécessite une certaine énergie. Ceci aura pour conséquence une réduction allant jusqu'à 60% de la force développée par l'actionneur.
  Dû à l'absence de friction, le déploiement du muscle artificiel pneumatique plissé ne nécessite pas d'énergie considérable. Il n'y aura donc pas de diminution de la force développée par l'actionneur.

Le muscle de FESTO est autre exemple d'actionneur pneumatique de ce type. En voici une illustration :

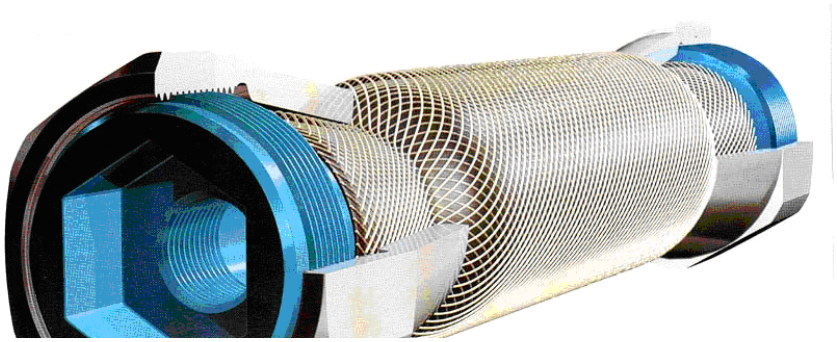


Fig. 1.4 : Muscle FESTO

Toutes les caractéristiques des muscles artificiels plissés : la forme, le volume, le diamètre et la contraction maximale, peuvent être déterminées précisément à partir d'un modèle mathématique. Le modèle est celui d'une membrane symétrique suivant un axe et soumise à une pression interne uniforme. Celle-ci développe des forces axiales à ses deux extrémités.
La membrane évoluera graduellement d'une forme cylindrique, à contraction nulle, vers la forme d'une sphère elliptique, à contraction maximale.
La figure 1.5 représente ces deux formes aux états extrêmes.

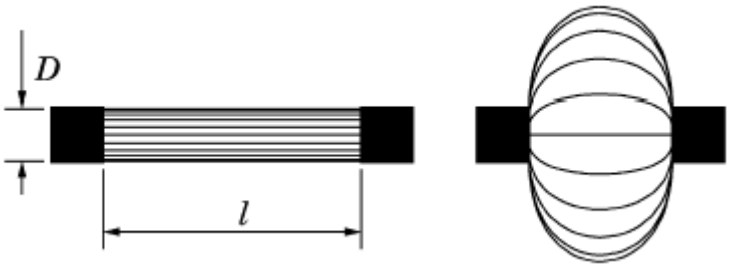


Fig. 1.5 : formes de l'actionneur

La dilatation est donc la plus importante au milieu de la membrane et diminue progressivement pour être nulle aux extrémités (fixations). Les caractéristiques telles que la géométrie, l'extension de la membrane, ainsi que la force de contraction dépendent de :

- la taille initiale (longueur l) de la membrane
- son diamètre initial D
- du degré de contraction ε
- de la pression appliquée p
- d'un paramètre a qui tient compte de l'élasticité longitudinale de la membrane.

Les expressions de la force (F), du diamètre maximum (D_m), ainsi que le volume (V) sont les suivantes :

$$F = p.l^2.f\left(\varepsilon, \frac{l}{D}, a\right)$$

$$D_m = l.d\left(\varepsilon, \frac{l}{D}, a\right)$$

$$V = l^3.v\left(\varepsilon, \frac{l}{D}, a\right)$$

avec f, d et v : des fonctions sans dimensions dépendant du rapport $\frac{l}{D}$ et de l'élasticité.

L'utilisation d'une membrane rigide en traction entraîne une influence de l'élasticité négligeable.

Les formules deviennent alors :

$$F = p.l^2.f\left(\varepsilon, \frac{l}{D}\right)$$

$$D_m = l.d\left(\varepsilon, \frac{l}{D}\right)$$

$$V = l^3.v\left(\varepsilon, \frac{l}{D}\right)$$

Le diagramme représentant l'évolution de la force en fonction de la contraction à pression constante est représenté à la figure 1.6 (la force déployée a été limité à 3500N). On observe que la force développée par les muscles artificiels pneumatiques varie avec le déplacement à pression constante.

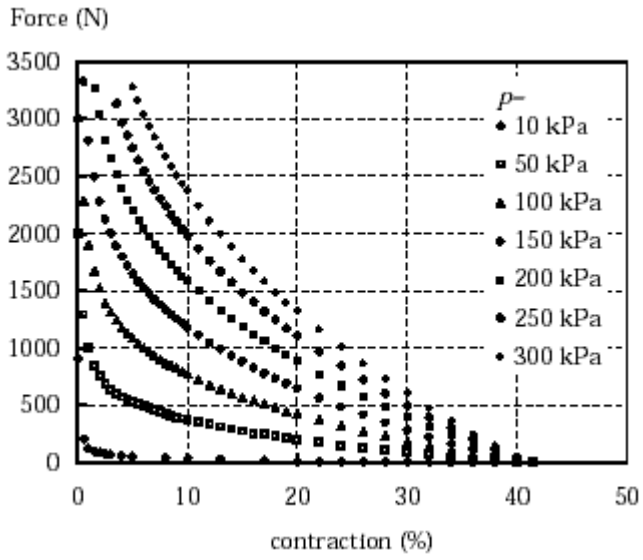


Figure 1.6 : Caractéristique Force-Contraction

Lorsque le muscle au repos est soumis à une certaine pression à l'intérieur de la membrane, une force de traction très importante est développée. Cette force de traction diminue lorsque la contraction augmente, et devient nulle lorsque la contraction est maximale.
De plus, ce même diagramme illustre le fait que le muscle artificiel pneumatique plissé peut travailler sur une large plage de pression. En effet celui-ci est utilisé pour une pression allant de 10kPa à 300kPa.

I.2. Objectifs :

Le principe de fonctionnement des muscles artificiels pneumatiques plissés est bien connu et l'utilisation de ces actionneurs est maîtrisée. Leur durée de vie ainsi que leur comportement dans le temps suscitent néanmoins encore des interrogations.
En effet, les caractéristiques de ces actionneurs pneumatiques n'ont pas encore été rigoureusement étudiées. La connaissance de celles-ci reste toutefois importante pour l'utilisateur. Un banc d'essai sera donc conçu et réalisé à cette fin.
Ce banc d'essai fournira notamment diverses informations telles que la contraction et la pression régnant à l'intérieur de l'actionneur pneumatique.
La conception et la réalisation de ce banc d'essai comprendront :

- le dimensionnement ainsi que le montage de la structure mécanique.

- le choix et l'implantation des capteurs pour la mesure de la contraction ainsi que de la pression régnant à l'intérieur de l'actionneur.

- la programmation du microcontrôleur MOTOROLA 68HC11 qui supervisera le test. Il récoltera les différentes mesures et enverra celles-ci à l'ordinateur via le port série RS232.

- La dernière étape consistera en la réalisation d'une interface sous Visual Basic. Celle-ci collectera les données venant du microcontrôleur et permettra leur visualisation.

Le choix de différents paramètres pour la phase de test se fera également à partir de l'interface.

Le test consistera en une utilisation répétée de l'actionneur. Une charge sera suspendue au muscle artificiel pneumatique et celui-ci sera soumis à un grand nombre d'étapes de contraction/détente.

La figure 1.7 représente le schéma de principe de fonctionnement du banc d'essai.

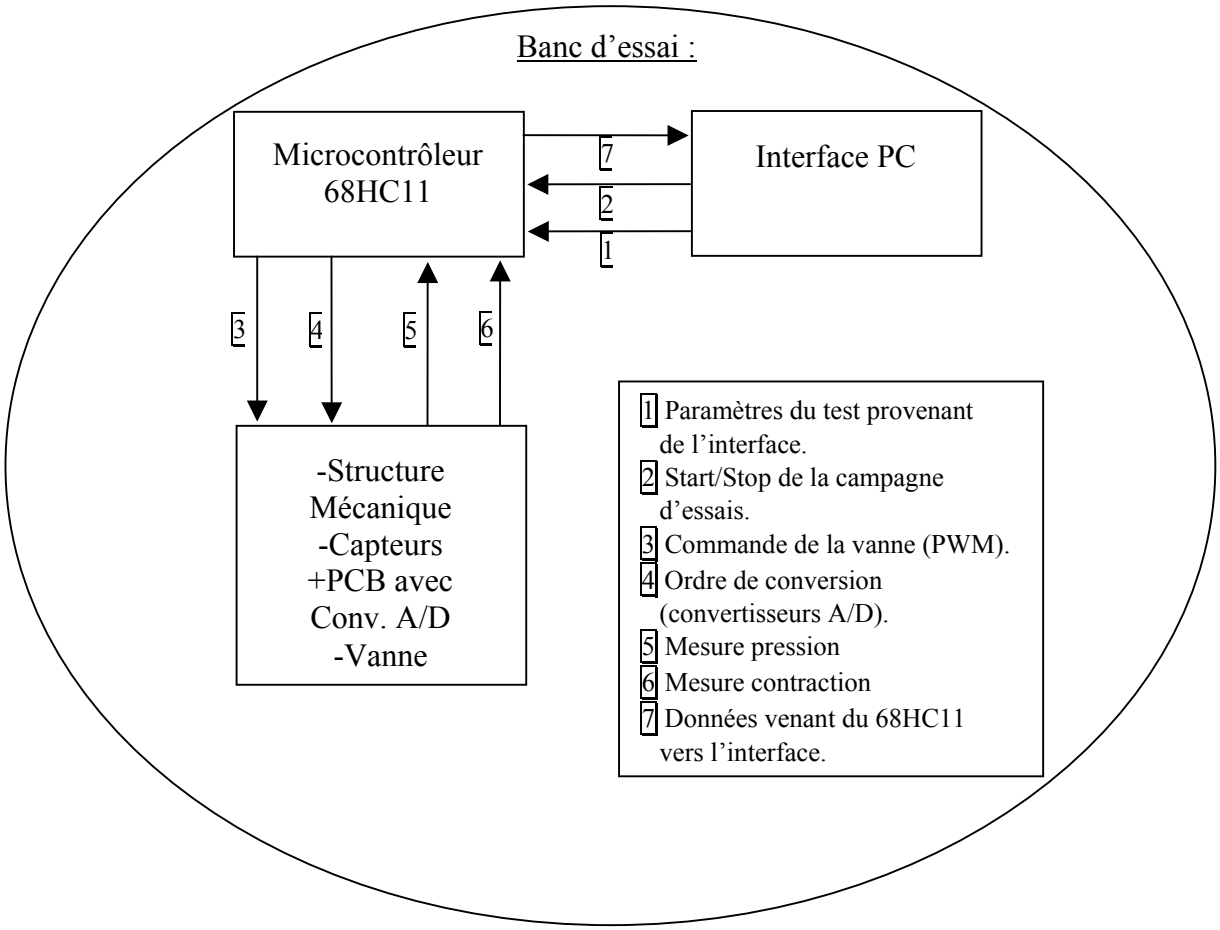


Fig. 1.7 : Principe de fonctionnement du banc d'essai.

II. DIMENSIONNEMENT DE LA STRUCTURE MECANIQUE :

II.1. Introduction :

Dans un premier temps, il a fallu choisir le type de structure à dimensionner.
Le muscle artificiel pneumatique ainsi que les différents capteurs seront fixés à cette structure. La charge, ayant un certain encombrement, sera suspendue à la partie inférieure de l'actionneur pneumatique. Une structure de type portique a donc été choisie, celle-ci permettra aisément le mouvement relatif de l'actionneur ainsi que celui de la charge. Après le dimensionnement du portique, le choix du mode d'assemblage des différents éléments sera étudié afin de répondre aux spécifications.

II.2. Calcul des efforts internes :

La charge appliquée sur la structure mécanique est une charge ponctuelle, essentiellement verticale. Elle est dirigée vers le bas et centrée sur la poutre transversale.
La charge suspendue au muscle, sera au maximum de 5000N.
L'ensemble des organes de fixation du muscle et de la charge, le muscle en lui-même, ainsi que les divers capteurs, augmenteront l'effort exercé sur la structure. La structure supportera donc une charge maximale totale de 5050N.

Etant donné que le portique ne sera pas boulonné ou fixé d'une quelconque manière dans le sol, le modèle du portique à deux montants encastrés est à proscrire. On choisira alors le modèle du portique à deux montants articulés.

Les dimensions du portique ont été choisies arbitrairement dans le but d'avoir une structure pas trop encombrante et transportable.

Les moments fléchissants d'un tel modèle pour une charge concentrée sont les suivants :

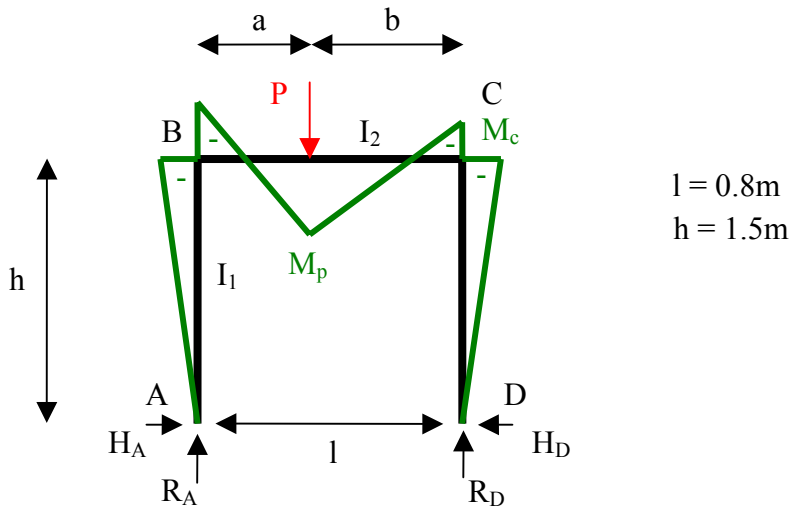


Fig. 2.1 : Diagramme des moments fléchissant

Dans notre cas, la charge P (5050N) est parfaitement centrée sur la poutre transversale ;

D'où $a = b = 0{,}4\text{m}$

Les expressions des moments fléchissants et réactions d'appui sont alors les suivantes :

$$R_A = R_D = \frac{P.a}{l}$$

$$H = H_A = H_D = \frac{3Pab}{2lh(2k+3)} \quad \text{avec } k = \frac{I_2.h}{I_1.l}$$

$$M_B = M_C = -H.h$$

$$M_P = \frac{4k+3}{2k+3}.\frac{Pab}{2l}$$

[GOULET]

Intuitivement, on remarque que I2 > I1. Mais pour des facilités de montage et également pour l'esthétique de la structure, on prendra les mêmes profils pour les poteaux ainsi que pour la poutre.

D'où $I_1 = I_2$

Résolution numérique :

$$\rightarrow k = \frac{h}{l} = \frac{1.5}{0.8} = 1{,}875 \qquad \text{[s.d.]}$$

La poutre transversale sera la plus sollicitée à l'endroit où on applique la charge. On va donc calculer la contrainte en ce point. La contrainte dans une poutre sollicitée en flexion vaut :

$$\sigma = \frac{M}{I/v} \leq \sigma_{adm}$$

avec M = moment fléchissant.
I = moment d'inertie de la section droite par rapport à l'axe autour duquel se fait la flexion.
v = distance entre centre de gravité et fibre extrême.

La contrainte admissible (σ_{adm}) vaudra la limite d'élasticité (F_y) du matériau divisé par un cœfficient de sécurité (s). Dans la pratique s = 1,5…2…

$$\rightarrow \sigma = \frac{M}{I_2/v_2} \leq \sigma_{adm} = \frac{F_y}{1.5}$$

Un acier couramment utilisé est le s235 pour lequel $F_y = 235\ N/mm^2$.

$$M_P = \frac{4.1{,}875+3}{2.1{,}875+3}.\frac{5050.0{,}4.0{,}4}{2.0{,}8} = 785\text{Nm}$$

$$\rightarrow \sigma = \frac{785}{I_2/v_2} \leq \frac{235}{1.5}\ N/mm^2$$

On trouve donc un module de résistance à la flexion élastique, suivant l'axe fort, de :

$$W_{el\ y\text{-}y} = \frac{I_2}{v_2} \geq 5\text{cm}^3$$

Cette valeur va nous permettre de choisir le profil qui supportera la charge.
La contrainte admissible a été fixée en considérant la sollicitation comme statique. En effet, les phases de vidange de l'actionneur pneumatique se font progressivement.
Les valeurs d'accélération de la masse sont donc réduites. Il en est de même pour l'effort résultant de ces accélérations.

II.3. Choix du profilé et du matériau :

Le matériau choisi pour la fabrication de notre portique est l'acier. Nous aurions pu prendre l'aluminium pour gagner en légèreté de la structure, mais le choix s'est arrêté à l'acier pour des raisons de coût et de flèches. En effet, pour un profil en aluminium dimensionné pour une même charge, ses déformations sont plus importantes que pour un profil en acier.
Les avantages de l'acier sont les suivants :

- Qualité et légèreté : la fabrication de l'acier est bien contrôlée, les produits sidérurgiques sont donc fiables. L'acier permet des ossatures légères et offre une résistance élevée pour une faible section.

- Facilité et rapidité de montage : Le montage sur site par boulonnage ou par soudage est aisé.

- Possibilité de transport et de déplacement en raison du poids peu élevé.

Le type de profil retenu est de type creux carré. L'avantage des profils carrés est qu'ils offrent la même résistance à la flexion suivant les axes x-x et y-y. Nous n'avons donc pas d'axe faible :

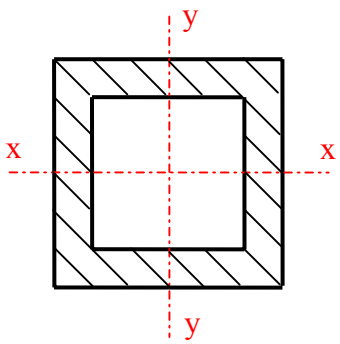


Fig. 2.2 : section du profil

De plus, ce profil creux va nous permettre de réaliser une économie de matière et donc un gain de poids sur la structure.
Pour connaître le profil qui va correspondre à nos critère de résistance, il suffit de prendre un profil dont le module de résistance à la flexion ($\frac{I}{v}$) est supérieur ou égal à celui calculé.
Ainsi, le catalogue HENSFERSTAUX nous propose un profil de 40x40x4 qui nous offre un module $\frac{I}{v}$ de 5.54 cm^3.

II.4. Vérification de la tenue des poteaux :

Le diagramme des moments fléchissants nous indique que chaque poteau est soumis à un effort de flexion (M_B) ; mais ceux-ci supportent également une charge les sollicitant en compression :

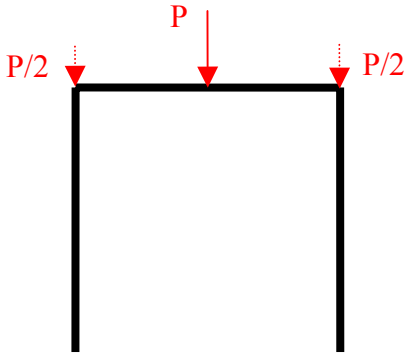


Fig. 2.3 : effort de compression

Chaque poteau reprend ainsi la moitié de la charge P. La contrainte résultant de l'effort de compression et de l'effort de flexion est la somme des contraintes des deux sollicitations.
En effet, l'effort de compression entraîne des contraintes normales σ_C .
La figure 2.4 représente une section droite soumise en compression sous une charge P/2 :

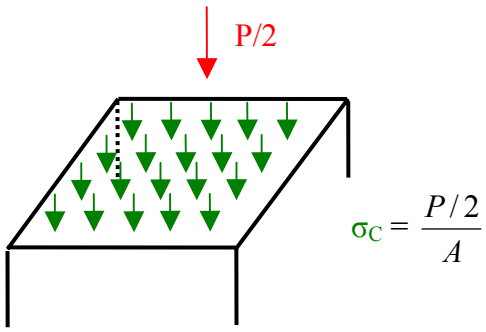


Fig. 2.4 : contrainte normale due à la compression

$$\rightarrow \sigma_C = - \frac{2525}{A} = -4{,}72 \, N/mm^2$$

Avec A = section de notre profil 40x40x4 = 5,35cm²

L'effort de flexion va lui aussi faire apparaître des contraintes normales (σ_{fl}). Une partie du profil sera sollicitée en traction et une autre partie en compression :

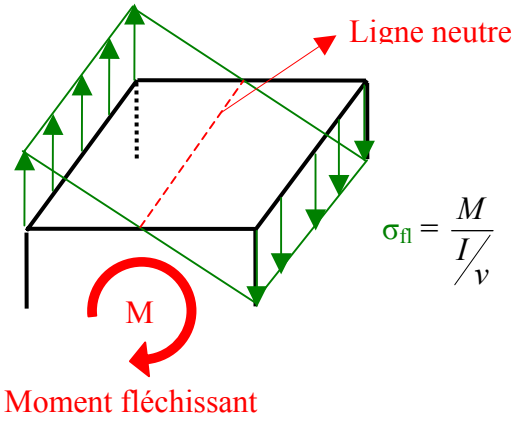


Fig. 2.5 : contrainte normale due à la flexion

Le profil choisi nous offre un module de résistance à la flexion $\frac{I}{v}$ de 5.54cm^3.

M = moment de flexion maximum

$$= M_B = \frac{-3Pabh}{2lh(2k+3)}$$

$$= \frac{-3.5050.0,4.0,4}{2.0,8.(2.1,875+3)} = -224 \text{ Nm}$$

La section du poteau la plus sollicitée est au sommet de celui-ci. En effet, le moment fléchissant (M_B) y est maximum et l'effort de compression est constant tout au long du poteau, à condition de considérer son poids propre négligeable.

$$\sigma_{total} = \sigma_C + \sigma_{fl}$$

$$\sigma_{total} = -4,72 - \frac{224.10^3}{5,54.10^3} \quad N/mm^2$$

$$\sigma_{total} = -4,72 - 40,4 \approx 45 \quad N/mm^2$$

L'acier couramment utilisé est le s235 ; la limite d'élasticité est donc de 235 N/mm^2 et en prenant comme cœfficient de sécurité s = 1,5 :

$$\sigma_{total} \leq \sigma_{adm} = \frac{235}{1,5} = 157 \quad N/mm^2$$

→ ***Le poteau est donc assez résistant pour supporter la charge appliquée au portique.***

II.5. Calcul de la charge critique de flambement :

Le flambement est un brusque dérobement latéral provoqué par un effort de compression. Etant donné que les poteaux du portique ont un certain élancement par rapport à leur section droite, il est intéressant de calculer la charge critique (F_C) à ne pas dépasser pour l'éviter.
Soit notre portique :

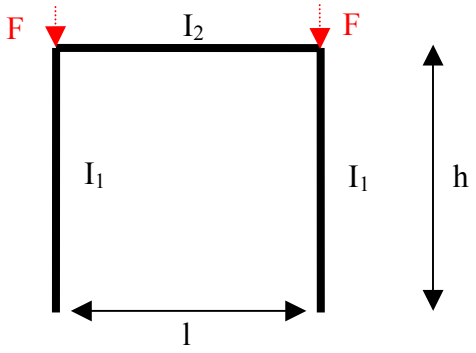


Fig. 2.6 : effort de compression provoquant le flambement

L'expression de la charge critique de flambement est la suivante [XIONG] :

$$F_C = \left(\frac{\pi}{hn}\right)^2 EI_1$$

$$\text{Avec } n = 2\sqrt{1+0{,}4.\frac{I_1 L}{I_2 h}}$$

I_1 et I_2 en mm^4

E = module d'élasticité de l'acier en $N/mm^2 = 210000N/mm^2$

Dans notre cas, les profils pour les poteaux et la poutre transversale étant les mêmes, $I_1=I_2=11{,}07cm^4$.

$$\rightarrow n = 2\sqrt{1+0{,}4.\frac{0{,}8}{1{,}5}} = 2{,}203$$

$$F_C = \left(\frac{\pi}{1500.2{,}203}\right)^2 210000.11{,}07.10^4$$

$$= 21011N \approx 21kN$$

Il faut donc une charge de 21kN appliquée au poteau pour provoquer le flambement. La charge verticale que reprend le poteau (2525N) est bien inférieure à F_C.

II.6. Calcul des flèches :

Afin de déterminer si les déformations ne sont pas trop importantes lors de l'application de la charge P sur le portique, un calcul des flèches, au point d'application de la charge, est nécessaire.
Ainsi la flèche verticale au centre de la poutre transversale vaut [DROUET] :

$$f = \frac{P_f . l^3}{77EI} \qquad [mm]$$

$$I = 11{,}07 cm^4$$
$$E = 210\,000\ N/mm^2$$
$$L = 0{,}8m$$

$$\text{Avec } P_f = 1.6\ P + 9{,}6\frac{M_B}{l} \text{ et } M_B = -224Nm \text{ (voir pt. 4)}$$

$$\rightarrow Pf = 1{,}6.5050 + 9{,}6.\frac{-224}{0{,}8}$$

$$= 5392N$$

$$\rightarrow f = \frac{5392.800^3}{77.210000.11{,}07.10^4} = 1{,}54mm$$

On aura donc une flèche verticale de 1,54mm au centre de la poutre transversale :

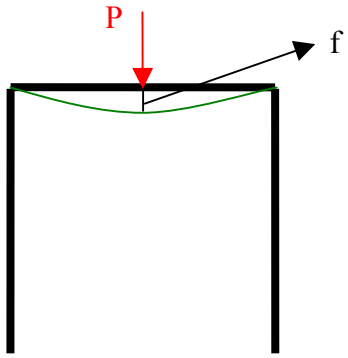


Fig. 2.7 : flèche verticale

II.7. Assemblage des éléments :

Maintenant que tous les éléments du portique sont dimensionnés, il reste à choisir le mode d'assemblage. Pour assembler des éléments métalliques on peut procéder soit par boulonnage soit par soudage.
L'assemblage par soudage présente plusieurs avantages intéressants, à savoir :

- une meilleure continuité des pièces assemblées et donc une meilleure transmission des efforts.

- pour les assemblages soudés, des pièces de transfert ne sont pas nécessaires (alors qu'elles sont indispensables pour les assemblages boulonnés).

Les assemblages soudés présentent néanmoins un inconvénient qui sera déterminant.
En effet, ils ne peuvent être démontés une fois assemblés. Or le portique doit pouvoir être transportable. On va donc choisir un assemblage par boulonnage pour notre structure afin de pouvoir effectuer un démontage et assemblage aisé et à souhait.

II.8. Réalisation pratique du portique :

Le stock disponible à la VUB nous a limité dans le choix des profils.
Les profils correspondant aux critères de résistance ($W_{el} \geq 5cm^3$), sont de dimensions 50x50x3 (mm).
En effet, suivant le catalogue HENSFERSTAUX, ces profils présentent un module de résistance à la flexion élastique W_{el} valant 7,98 cm^3 ($I = 19.94cm^4$).
Le portique a donc été réalisé avec ces profils.

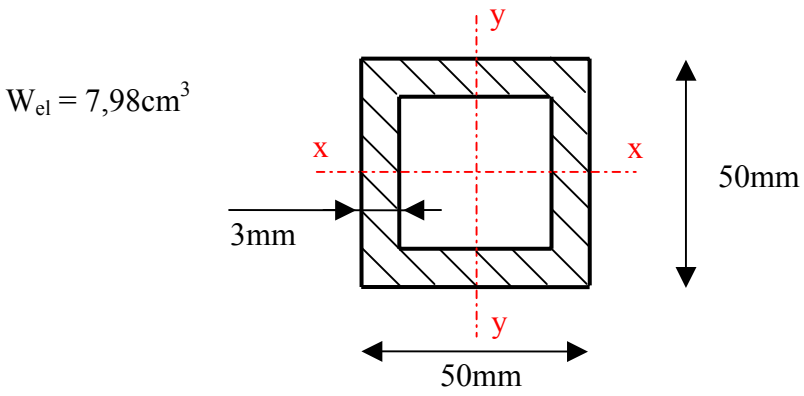


Fig. 2.8 : dimensions du profil utilisé

Calcul de la déformée :

La flèche aura, à charge maximale (5050N) et avec le profil 50x50x3, pour valeur :

$$f = \frac{5392.800^3}{77.210000.19,94.10^4} = 0,86mm$$

Fig. 2.9 : portique assemblé

III. CAPTEURS ET VANNE

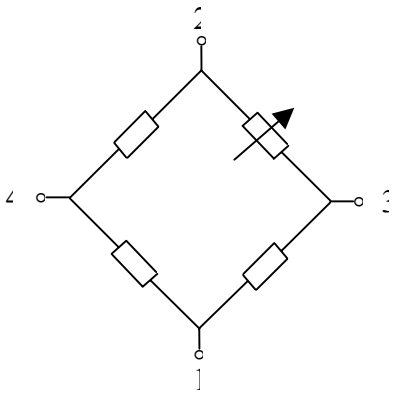
III.1. Mesure de pression :

a) Capteur :

Concernant la mesure de pression, le capteur doit être le moins encombrant possible. En effet, il sera placé directement à l'intérieur du muscle artificiel. On veillera également à un faible coût. Le capteur a été fourni par la VUB, il s'agit d'un capteur provenant de chez HONEYWELL. Il porte la référence *CPC100*.
Le principe de fonctionnement de celui-ci est basé sur l'effet piezorésistif, c'est-à-dire basé sur une variation de résistance sous l'effet d'une contrainte mécanique.
Dans notre cas, la variation de résistance sera provoquée par une variation de pression à l'intérieur du muscle artificiel.
La pression mesurée par le capteur est une pression absolue ; Etant donné que c'est la pression relative qui nous intéresse, on veillera à retirer la valeur de la pression atmosphérique. En effet, la relation unissant ces deux grandeurs est la suivante :

$$p_{absolue} = p_{relative} + p_{atm}$$

Le capteur de pression est compensé en température et également calibré. Sa fiche technique complète est disponible en Annexes.
Afin de bien comprendre le fonctionnement du capteur de pression, on peut représenter celui-ci par son schéma équivalent :



1. Alimentation V-
2. Alimentation V+
3. Out+
4. Out-

Fig 3.1 : Schéma équivalent du capteur de pression

On fournira à ce capteur une alimentation de 12V (bornes 1 et 2). Lors d'une pression absolue non nulle, on verra apparaître, à ses bornes 3 et 4, une différence de potentiel proportionnelle au mesurande.
C'est cette différence de potentiel, image de la pression absolue, que l'on viendra mesurer.

b) Amplification et conversion A/D :

La différence de potentiel entre les bornes 3 et 4 du capteur de pression est une grandeur analogique. Ce type de grandeur est très sensible à l'environnement et peut être perturbée par les interférences d'autres appareils : proximité d'une alimentation, moteur électrique, etc…
On veillera donc à convertir cette grandeur analogique en grandeur numérique. Celle-ci n'est en effet pas sujette à ce genre de perturbations.
Cette conversion vers le domaine numérique n'est pas réalisée dans l'unique objectif de limiter les effets des perturbations et bruits sur le signal utile. En effet, les données provenant du capteur vont être recueillies par un microcontrôleur puis envoyées vers un ordinateur. Afin de manipuler ces informations, la conversion en grandeur numérique est nécessaire.

Alimenté en 12 VDC, notre capteur de pression délivre une différence de potentiel de 100mV pour une pression absolue de 100 PSI (14.5 PSI = 1 bar). On va donc amplifier cette différence de potentiel avant de la convertir en grandeur numérique. Cette conversion se fera le plus près possible du montage d'amplification. Tout le montage sera réalisé sur le même circuit imprimé placé directement à l'intérieur du muscle artificiel.

Le circuit d'amplification et de conversion A/D, fourni par la VUB, est repris à la figure 3.2. Nous allons en expliquer les différentes parties.

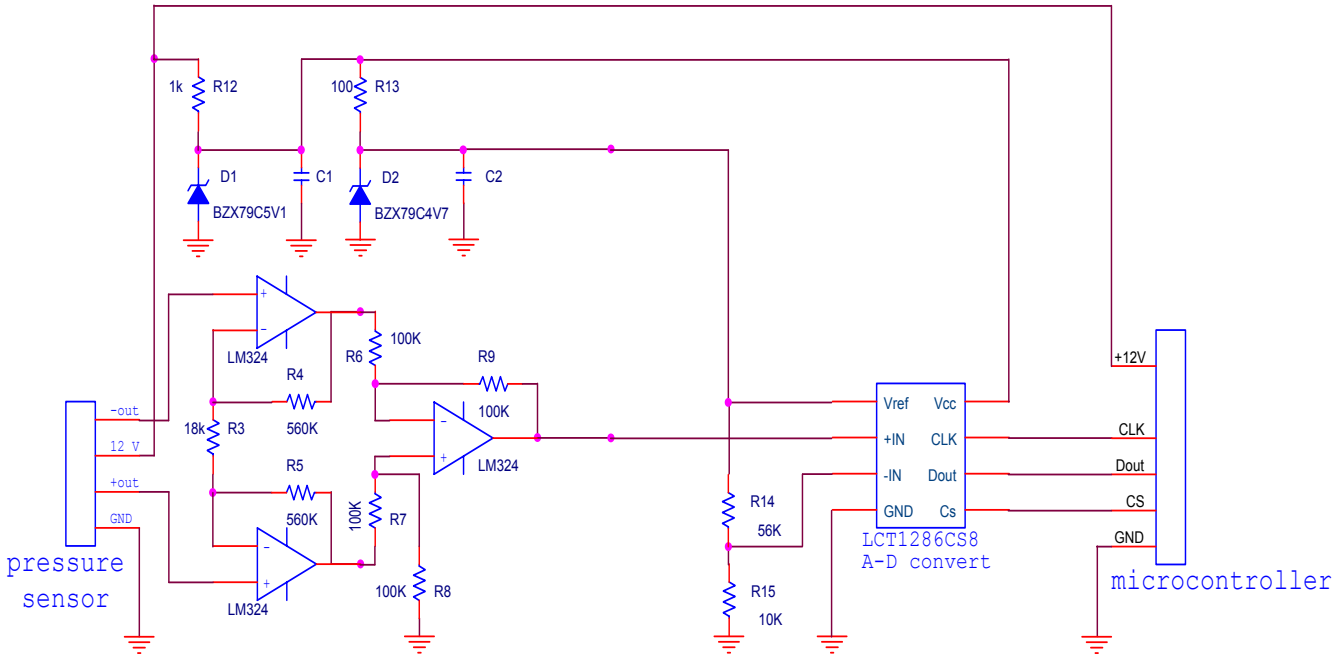


Fig. 3.2 : Circuit d'amplification et de conversion A/D

- *Amplification*

Comme nous l'avons vu, la grandeur image du mesurande aux bornes de notre capteurde pression est de faible niveau. Pour travailler avec une bonne précision, il est nécessaire de l'amplifier avant de la mesurer (dans notre cas, avant de la convertir en grandeur numérique). Cette grandeur analogique contient le signal utile (qui nous intéresse) mais aussi une tension parasite (interférence) et également une tension dite de mode commun. On choisira donc un montage amplificateur à fort taux de rejection de mode commun afin de n'amplifier que le signal utile.
Le montage choisi porte le nom d'amplificateur d'instrumentation. Celui-ci est constitué de 3 amplificateurs opérationnels et de plusieurs résistances.
Son schéma de montage est le suivant :

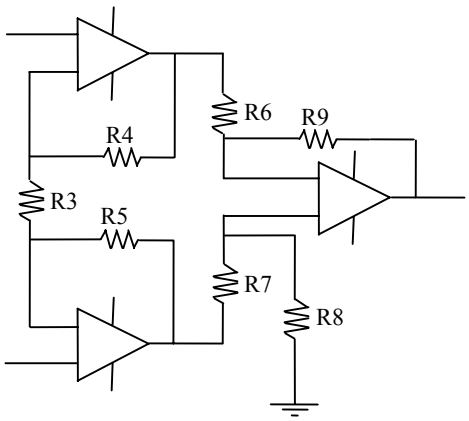


Fig 3.3 : Amplificateur d'instrumentation

C
e type de montage existe en circuit intégré mais peut être également réalisé à l'aide d'éléments discrets. Le choix se portera sur la deuxième alternative.

Pour calculer le gain d'un tel montage, étudions d'abord le 1er étage d'amplification :

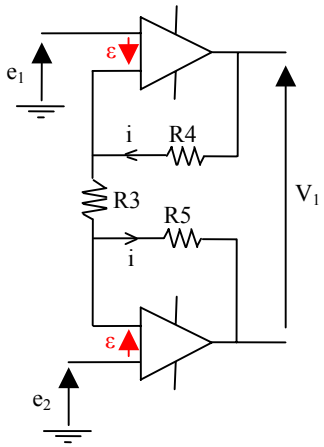


Fig 3.4 : 1er étage d'amplification

En supposant que les amplificateurs opérationnels fonctionnent en régime non saturé, on peut admettre que ε tend vers zéro et donc $V_+ \approx V_-$.
De plus, on considère l'impédance d'entrée des amplificateurs opérationnels valant l'infini.
On peut alors écrire les équations suivantes :

$$e_1 - e_2 = R_3.i$$
$$\rightarrow i = \frac{e_1 - e_2}{R_3}$$

$$V_1 = (R_3 + R_4 + R_5).i$$
$$\rightarrow V_1 = (R_3 + R_4 + R_5).\frac{(e_1 - e_2)}{R_3}$$

En posant que $R_4 = R_5$

On trouve : $V_1 = (1 + \frac{2R_4}{R_3})(e_1 - e_2)$

Le gain d'un circuit valant le rapport de la sortie sur l'entrée, on trouve comme gain du premier étage :

$$G_1 = 1 + \frac{2R_4}{R_3}$$

On étudie alors le deuxième étage d'amplification :

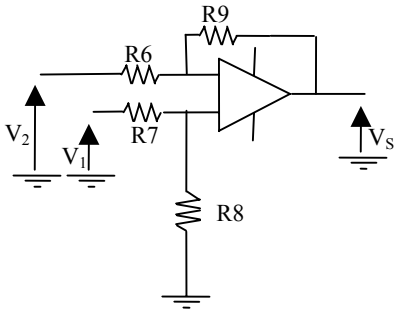


Fig 3.5: 2ème étage d'amplification

En considérant l'impédance d'entrée de l'amplificateur opérationnel infinie, on peut donc écrire que :

$$V_+ = V_1 \frac{R_8}{R_7 + R_8}$$
$$V_- = \frac{V_2 R_9 + V_S R_6}{R_9 + R_6}$$

En posant les même hypothèses que pour le 1er étage d'amplification : ε → 0 :

$$V_+ = V_-$$

$$V_1 \frac{R_8}{R_7 + R_8} = \frac{V_2 R_9 + V_S R_6}{R_9 + R_6}$$

En isolant la tension de sortie de cet étage d'amplification, on trouve :

$$V_S = V_1 \frac{R_8}{R_6} \frac{(R_9 + R_6)}{(R_7 + R_8)} - V_2 \frac{R_9}{R_6}$$

$$\text{Si } \frac{R_9}{R_6} = \frac{R_8}{R_7}$$

$$\text{On trouve alors : } V_S = \frac{R_9}{R_6}(V_2 - V_1)$$

Le gain du deuxième étage d'amplification vaut donc :

$$G_2 = \frac{V_S}{(V_2 - V_1)} = \frac{R_9}{R_6}$$

Pour le montage complet avec les deux étages d'amplification, on a donc un gain total valant le produit des gains de chaque étage c'est-à-dire :

$$G_T = (1 + \frac{2R_4}{R_3}) \frac{R_9}{R_6}$$

L'expression du gain total du montage de l'amplificateur d'instrumentation indique que celui-ci ne dépend que des valeurs des résistances du montage. Il convient donc de choisir celles-ci afin de fixer le gain approprié à notre utilisation.
Le choix de la valeur du gain d'amplification se fera de la manière suivante : en sortie de l'amplificateur, on va tenter d'obtenir une correspondance entre la tension et la pression absolue afin que l'on ait 1 Volt pour 1 Bar mesuré.
Les données techniques du capteur de pression piézorésistif nous indiquent que celui-ci fournit une différence de potentiel à ses bornes 3 et 4 valant 100mV pour une pression absolue de 100PSI (14.5 PSI = 1bar). On a donc 1 mV par PSI.
La valeur du gain assurant la correspondance souhaitée est donc de 69. La réalisation du montage (gain de l'amplificateur) se fera avec les résistances SMD disponibles dans le laboratoire ; on arrivera ainsi à fixer un gain de 63 pour G_T.

Etant donné que notre capteur de pression mesure une pression absolue, on ne descendra jamais en dessous d'une certaine valeur de tension (valant environ 1V).
Du point de vue du convertisseur A/D, on perd donc une partie de la plage de tension (voir point *Conversion A/D*).
On va donc effectuer une compensation électrique afin de combler cette perte.
On va en fait relever le potentiel de la borne -*IN* du convertisseur. Pour cela, on va établir un diviseur de tension à l'aide de résistance :

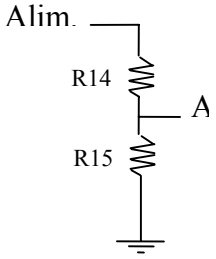


Fig. 3.6 : Diviseur de tension

Le point *A* sera donc connecté à une des bornes d'entrée *–IN* du convertisseur A/D.
On choisira les valeurs des résistances afin d'avoir un potentiel au point *A* inférieur à 1 V. En effet, la pression atmosphérique fluctue selon les conditions météorologiques, il est donc risqué de prendre exactement 1V. Sachant que l'alimentation du diviseur de tension est de 4.7V (voir figure 3.2), on a choisi, pour les valeurs de R14 et R15, 56KΩ et 10KΩ. Le potentiel du point A vaut :

$$V_A = 4{,}7.\frac{10}{56+10} = 0{,}7V$$

- *conversion A/D*

Le convertisseur A/D va donc nous permettre de convertir une grandeur analogique en une grandeur digitale. Comme vu précédemment, cette conversion vers le domaine numérique est nécessaire pour limiter l'influence des perturbations extérieures sur le signal utile. Mais également afin de pouvoir manipuler les données par le microcontrôleur ainsi que par l'ordinateur.
Le convertisseur utilisé pour notre application est le LTC1286 de LINEAR TECHNOLOGY. Il s'agit d'un convertisseur 12 bits, basé sur la méthode d'approximation successive.
Le principe de cette méthode est le suivant : Le convertisseur analogique – numérique reçoit à son entrée une tension v_i. Il délivre en sortie un mot de 12 bits correspondant à la valeur numérique N associé à v_i. Ce convertisseur est donc caractérisé par une plage de tension analogique convertible qui est fixé par une tension de référence (V_{ref}).

Comme l'information en sortie du convertisseur est codée sur 12 bits, on peut délivrer, au maximum, 2^{12} mots distincts pour numériser la plage de tension analogique définie par V_{ref}.
A chaque mot correspond donc une tension analogique. La différence entre deux tensions analogiques successives est appelée quantum q. La valeur de ce quantum est donc de :

$$q = \frac{V_{ref}}{2^{12}}$$

Pour convertir on procède de la manière suivante : à la tension v_i correspond une valeur numérique N dont l'expression générale en binaire, codée sur 12 bits, vaut :

$$N = b_{11}.b_{10}....b_1.b_0$$

Par comparaison de v_i avec 12 valeurs de tension convenablement choisies, on détermine successivement la valeur 1 ou 0 des 12 bits constituant N. La première comparaison teste le bit b_{11} qui est le seul mis à 1 ; la valeur N_1 vaut donc $100...00$ et correspond à une tension va_1 :

$$va_1 = N_1.q$$
$$\text{Si } v_i \geq va_1 \text{ alors } N \geq N_1 \text{ et } b_{11} = 1$$
$$\text{Si } v_i < va_1 \text{ alors } N < N_1 \text{ et } b_{11} = 0$$

La valeur trouvée pour b_{11} est fixée définitivement.

On procède ensuite à la deuxième comparaison durant laquelle on va tester le bit b_{10} qui est à son tour mis à 1 ; la valeur numérique N_2 valant alors $b_{11}100...00$, correspond à une tension va_2 :

$$va_2 = N_2.q$$
$$\text{Si } v_i \geq va_2 \text{ alors } N \geq N_2 \text{ et } b_{10} = 1$$
$$\text{Si } v_i < va_2 \text{ alors } N < N_2 \text{ et } b_{10} = 0$$

On a donc trouvé également la valeur du bit_{10}. On procède de la même manière jusqu'au dernier bit. Les valeurs des 12 bits constituant *N* sont donc trouvées par une suite de 12 comparaisons successives.
Les valeurs v_a sont fournies par un convertisseur numérique – analogique, interne au convertisseur A/D. Celui-ci fournit une tension proportionnelle à la valeur numérique du mot d'entrée :

$$v_a = N.q$$

Un convertisseur A/D externe a été utilisé à la place du convertisseur A/D du microcontrôleur Motorola pour deux raisons : Premièrement, le convertisseur du microcontrôleur est un convertisseur 8 bits. On a donc gagné en résolution en utilisant un convertisseur externe. De plus l'utilisation du convertisseur MOTOROLA impliquerait la présence de fils d'une certaine longueur pour connecter la sortie de l'amplificateur d'instrumentation (grandeur analogique) à sont entrée. Par cette manière, les grandeurs analogiques auraient été fortement perturbées par les interférences externes.
Voici un schéma des entrées/sorties du convertisseur LTC1286 ainsi que leurs rôles :

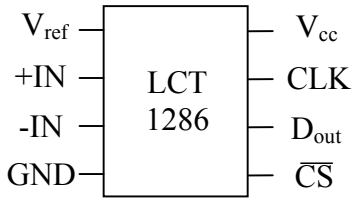


Fig 3.7 : Convertisseur A/D LCT1286

La borne d'entrée *+IN* sera connectée à la sortie de l'amplificateur d'instrumentation.
La borne *-IN* sera connectée au point *A* (0,7V) du diviseur de tension, comme expliqué au paragraphe précédent. Ainsi la plage de tension définie entre les bornes *+IN* et *–IN* doit être incluse dans la plage de tension définie entre la borne V_{ref} et la masse (*GND*).

Pour commencer la conversion, le convertisseur A/D doit détecter un flanc descendant sur sa pin *~CS* (Chip Select). Il va ensuite coder la grandeur numérique sur 12 bits.
Le transfert des données numériques du convertisseur vers le microcontrôleur se fait au travers de l'interface SPI (Serial Peripheral Interface). Il s'agit d'un transfert synchronisé sur le signal *CLK* provenant du microcontrôleur. La fréquence de ce signal d'horloge peut être égale à celle de l'horloge interne du microcontrôleur (2Mhz), ou à une fraction de celle-ci (voir Chap. IV). Ce transfert de données se fera via la borne de sortie D_{out}. Il devra se réaliser en deux étapes. En effet, le bus de données du microcontrôleur MOTOROLA est un bus 8 bits.
Celui-ci ne saurait donc recevoir et véhiculer un paquet de données codé sur 12 bits. Le convertisseur enverra donc les 12 bits de la donnée en deux étapes.

- *Alimentation et tension de référence*

Pour fournir une tension d'alimentation ainsi qu'une tension de référence stable et fixe, on va utiliser des diodes ZENER ainsi que des condensateurs.

Voici la caractéristique classique d'une diode ZENER [NOEL] :

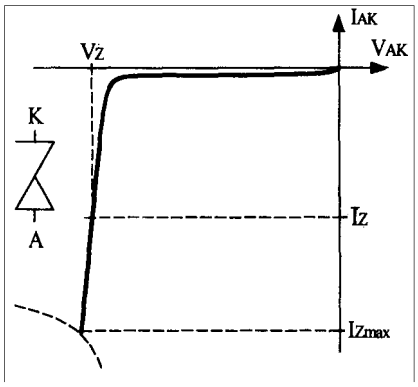


Fig 3.8 : Caractéristique d'une diode ZENER

On travaille dans la zone d'avalanche, ce phénomène est non destructif à condition de respecter les limites en puissances du composant ; pour ce faire on place une résistance (R_{12} et R_{13}) afin de limiter le courant dans la diode (voir fig. 3.2).
La tension de claquage (V_z) de la diode ZENER est fixe et vaut pour l'alimentation 5,1V et pour la tension de référence 4,7V.

c) Réalisation du circuit imprimé :

Le circuit imprimé a été réalisé sous TRAXMAKER. Il s'agit d'un circuit imprimé double face. Les composants à implanter sont les suivants :

- La puce contenant les amplificateurs opérationnels pour la réalisation de l'amplificateur d'instrumentation, il s'agit de la LM324 de chez NATIONAL SEMICONDUCTOR.
  Les données techniques de cette puce sont disponibles en Annexes.
- La puce du convertisseur A/D LTC1286
- Les résistances SMD
- Les deux condensateurs pour l'alimentation et la tension de référence du convertisseur A/D
- Les deux diodes ZENER utilisées pour la même cause
- Les deux résistances classiques qui limitent le courant dans les diodes ZENER

La difficulté pour la réalisation de ce circuit imprimé était la limitation de son encombrement. En effet, le capteur de pression va se placer directement à l'intérieur du muscle.

Le circuit imprimé supportant tout le montage devait donc être réalisé afin de pouvoir occuper cette position et être fixé sur l'attache inférieur du muscle prévue pour accueillir également le circuit imprimé (voir annexe page 82).

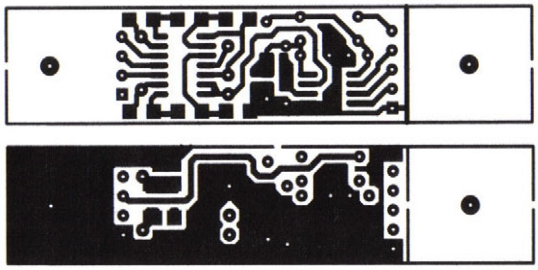


Fig.3.9 : face avant et face arrière du circuit imprimé

d) Calibration du capteur :

La calibration du montage comprenant le capteur de pression et le convertisseur a été réalisée. On connaît ainsi exactement son comportement lors des mesures de pression. C'est-à-dire savoir exactement quelle information sera fournie par le convertisseur A/D pour les différentes valeurs de pression possibles.

Cette calibration est faite de la manière suivante : la vanne est connectée à un volume clos indéformable. A ce même volume est connecté le capteur de pression piezorésistif monté sur son circuit imprimé, de même qu'un deuxième capteur de pression. Il s'agit du PN2024 de IFM ELECTRONIC (données techniques en Annexes). Ce capteur va nous fournir la mesure de pression (pression relative), cette mesure de pression peut servir de référence vue la précision de ce capteur. A la valeur indiquée par le capteur PN2024, on ajoute la pression atmosphérique mesurée avec un baromètre. Ceci va donc nous permettre d'établir la correspondance entre la pression régnant à l'intérieur du volume clos et l'information fournie par le convertisseur A/D.
Voici un schéma du montage :

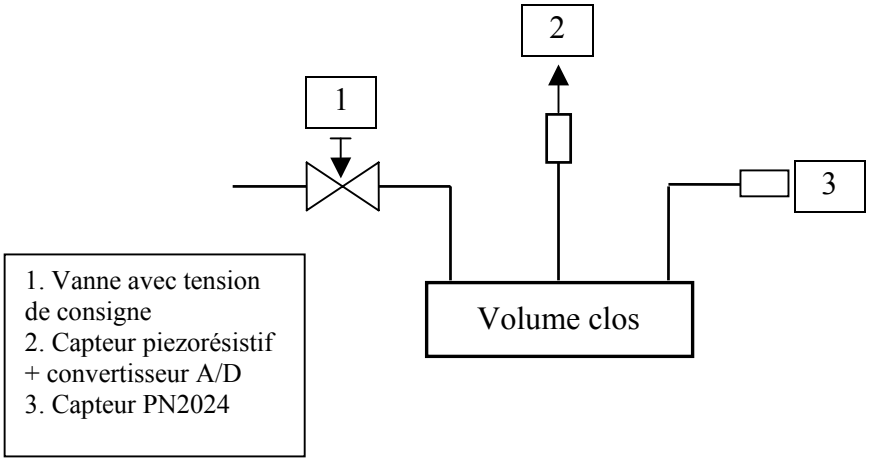


Fig. 3.10 : montage pour la calibration du capteur de pression

Le principe de la manipulation est le suivant : on applique une tension de consigne à la vanne. Celle-ci va soumettre le volume clos à une certaine pression. On recueille la valeur de cette pression à l'aide du capteur PN2024 ainsi que l'information du montage 2. On pourra alors en établir la caractéristique.
La correspondance entre ces deux grandeurs nous sera utile dans l'interface Visual Basic.

On obtient alors la caractéristique suivante :

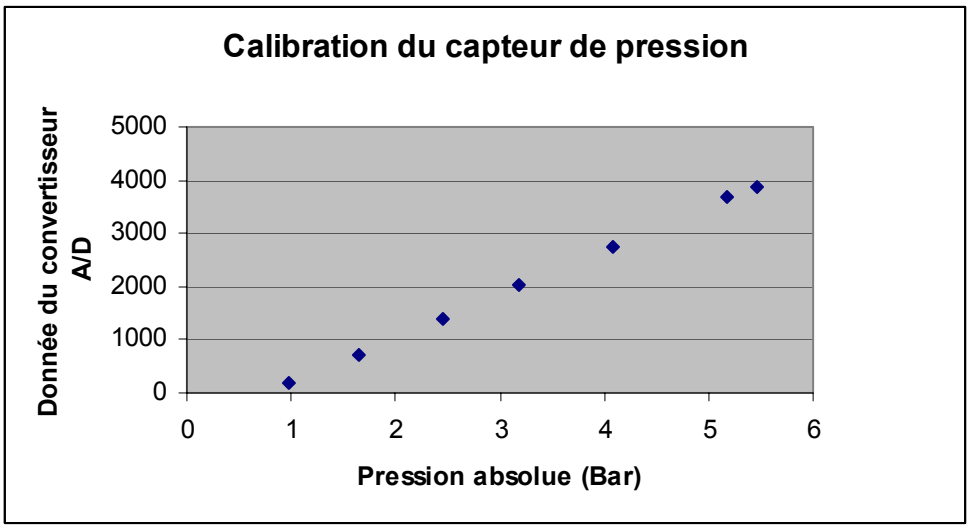


Fig. 3.11 : Calibration du capteur de pression

Toutes les valeurs relatives à cette manipulation sont reprises dans un tableau situé en Annexes page 83.
On va identifier cette suite de point à l'aide d'une régression linéaire (droite des moindres carrés). L'équation de cette droite vaut :

$$y = 830{,}13x - 638{,}25$$

Le carré du cœfficient de corrélation est également fourni avec la droite des moindres carrés. Celui-ci vaut 0,9999, ce qui indique une linéarité quasi parfaite des points de mesure.
Une superposition de la droite et des mesures a été réalisée en Annexes page 83.

III.2. Mesure de la contraction :

a) Comparaison et choix du capteur :

Il convient donc de choisir le capteur qui va permettre de mesurer la deuxième grandeur utile de notre banc d'essai : la contraction.
La contraction de l'actionneur pneumatique est un mouvement unidimensionnel. Une multitude de capteurs sont adaptés à ce type de mesure. Un tableau comparatif a été établi entre les différentes solutions possibles. Il est disponible en Annexes page 84.

Le choix du capteur de contraction s'est fait à base d'un critère économique. C'est donc le capteur linéaire potentiométrique de chez ELAP (PLS100) qui a été sélectionné (données techniques en Annexes). La mesure de contraction se fait donc à l'aide d'un diviseur résistif.

De plus, les données techniques du capteur sélectionné indiquent que sa durée de vie est adéquate pour le type d'utilisation auquel il sera destiné.

Identiquement à la mesure de pression, on va effectuer une conversion de la grandeur analogique (tension) en une grandeur digitale à l'aide d'un convertisseur A/D (LTC1286).
On a ainsi repris les mêmes circuits pour l'alimentation et la tension de référence du convertisseur A/D. Le capteur de contraction sera alimenté en 5V, suivant les conseils du constructeur. Le courant circulant dans le capteur doit être au maximum de 1 mA (résistance interne = 5kΩ). L'alimentation de ce capteur proviendra de la carte de développement du microcontrôleur MOTOROLA 68HC11.

b) Méthode de mesure :

Le capteur de contraction sera positionné parallèlement au muscle artificiel. Il sera fixé à l'aide de ses pièces de fixation (figure 3.12 pièce 1 et figure 3.13 pièce 2).
Pour ne pas gêner la mesure de contraction lors de la dilatation du muscle artificiel, on va déporter le mouvement vertical de l'actionneur parallèlement à lui-même. On a donc réalisé une pièce cylindrique (figure 3.14 pièce 3) qui s'adapte sur la fixation inférieure du muscle. Toutes les pièces usinées ont été réalisées en aluminium (plans 2D disponibles en Annexes page 86). Une représentation en perspective des pièces de fixation de l'actionneur est présente également en Annexes page 81.
Le mouvement vertical de l'actionneur peut être accompagné de mouvements parasites tels que des balancements de la masse suspendue. Afin de préserver le capteur contre tout dégât matériel, le couplage de ce dernier et de la pièce 3 s'est fait à l'aide d'un système constitué d'une tige rigide et de deux cardans.

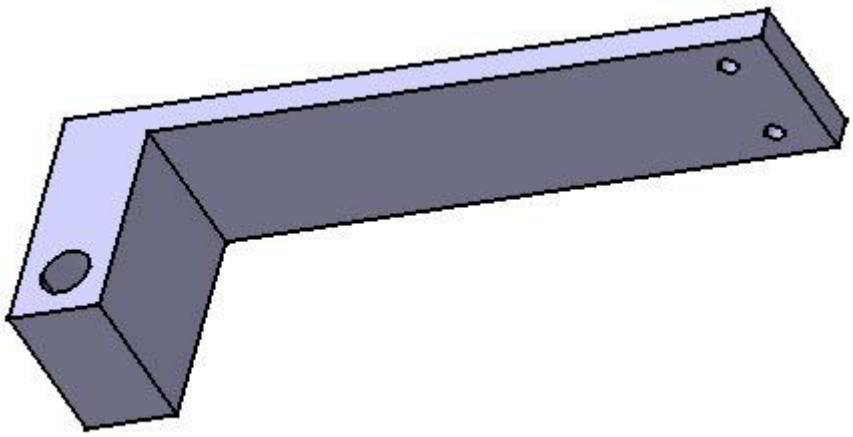


Fig. 3.12 : Pièce 1

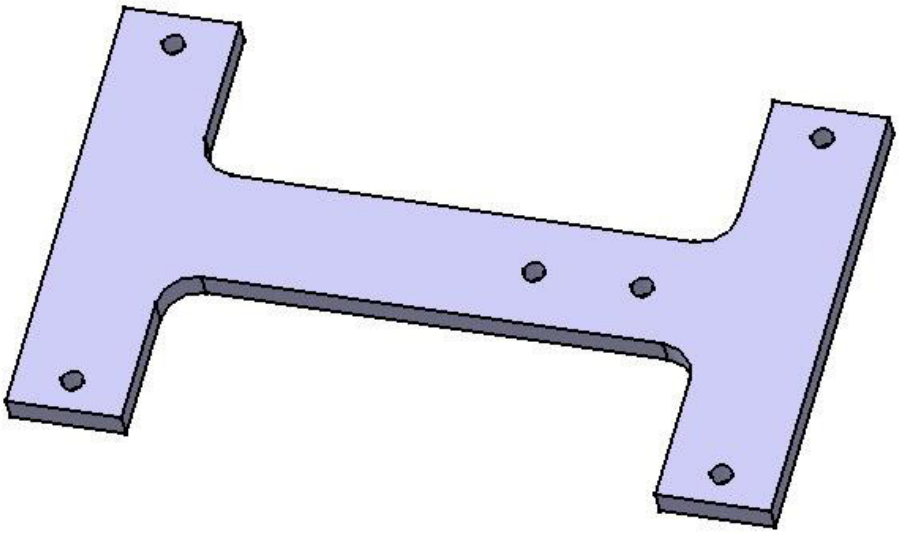


Fig. 3.13 : Pièce 2

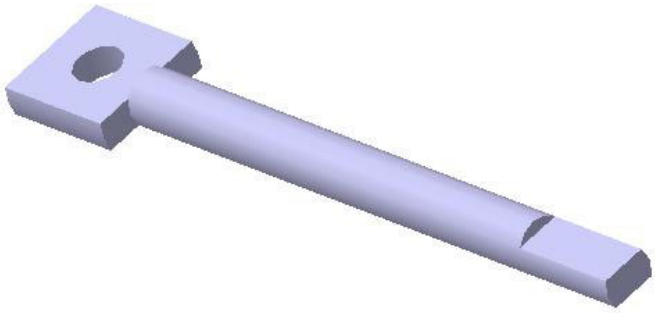


Fig. 3.14 : Pièce 3

Voici une représentation des différentes pièces assemblées. Le schéma ne comporte pas le système de cardans :

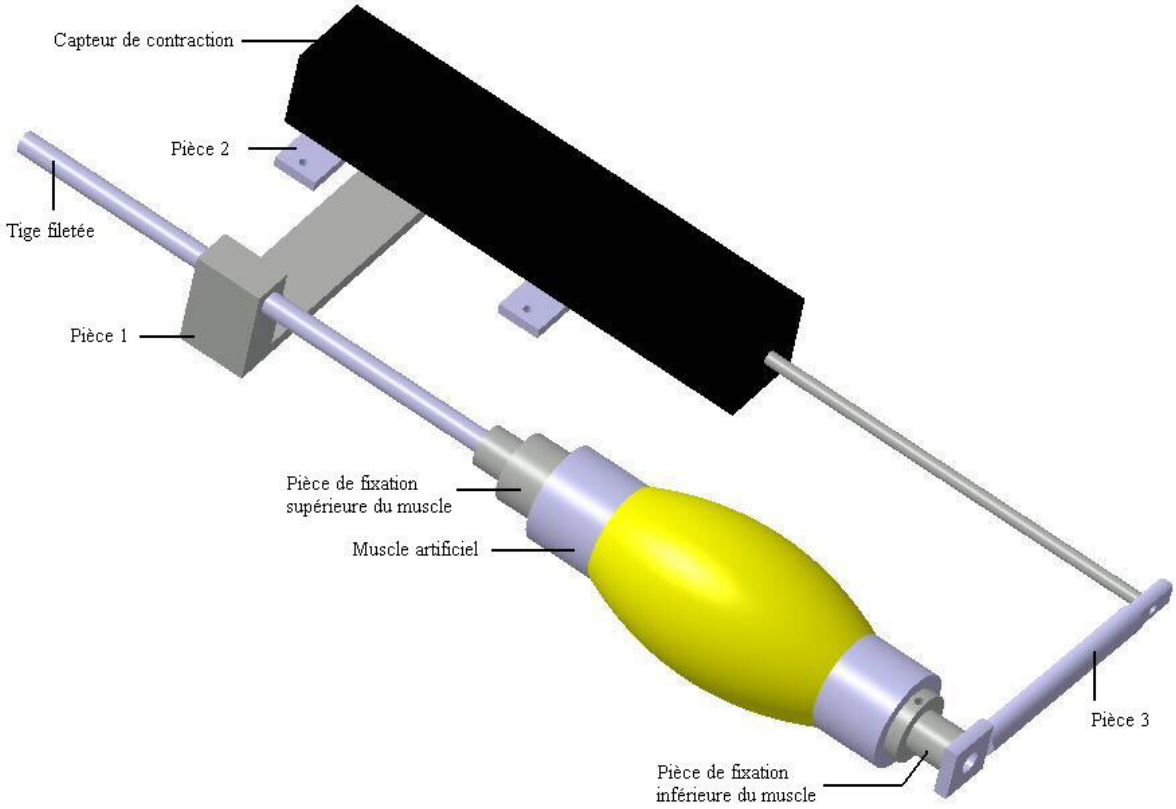


Fig. 3.15 : Pièces assemblées

Ce type de capteur comporte des inconvénients : il y a un contact entre le piston du capteur et le palier dans lequel celui glisse. De plus, la mesure de position se fait par déplacement d'un pointeau sur une surface résistive (diviseur de tension). Il y a également contact entre le pointeau et la dite surface. Lorsque deux corps sont en contact, afin de provoquer le déplacement relatif d'un des corps par rapport à l'autre, il faut vaincre une force d'arrachement. Celle-ci s'oppose donc au déplacement du corps.

Il existe donc dans notre capteur potentiométrique, une force d'arrachement à vaincre pour déplacer le piston et le pointeau le long de la surface résistive. Néanmoins, comparée aux forces engendrées par l'actionneur pneumatique, cette force est négligeable.

c) Calibration du capteur :

Afin de connaître la correspondance entre les données du convertisseur et la position, on a également procédé à une calibration. Les données relatives à cet essai sont disponibles en Annexes page 89.

On obtient alors le graphique suivant :

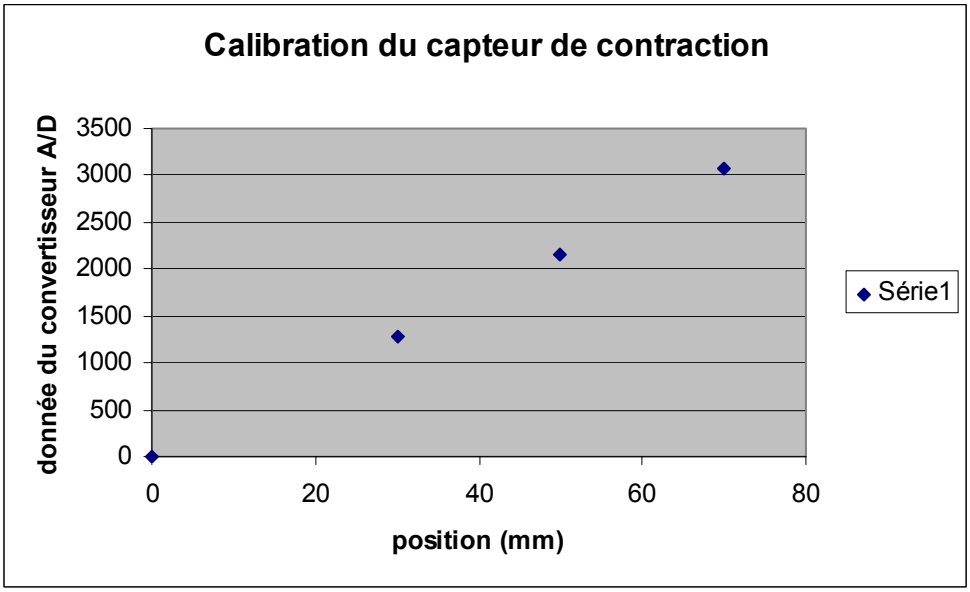


Fig. 3.16 : Calibration du capteur de contraction.

On a déterminé pour ces points une droite de régression par les moindres carrés ; l'équation de celle-ci vaut :

$$y = 43{,}831x - 15{,}907$$

Le carré du cœfficient de corrélation vaut 0.9997 et atteste encore de la bonne linéarité des mesures.
La superposition de la droite des moindres carrés et les points de mesures a été établie en Annexes page 89.

III.3. Vanne pneumatique :

La vanne pneumatique, fournie par la VUB, est une vanne provenant de chez FESTO. C'est une vanne munie d'un régulateur de pression mécanique (de qualité inférieure). Ses données techniques sont disponibles en Annexes.
On applique à ses bornes d'entrée une tension (0-10V) et la pression est alors proportionnelle à cette tension de consigne.
Lorsque l'on observe le fonctionnement de cette vanne, on remarque une hystérésis lors des phases de montée et de descente en pression.

Une illustration de cette hystérésis est représentée à la figure 3.17.

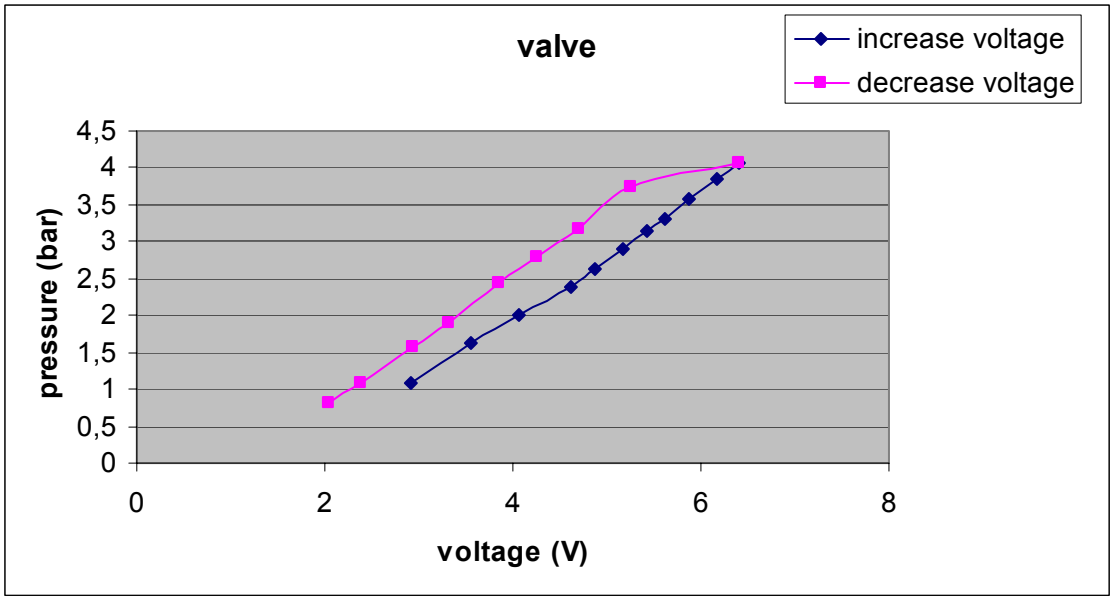


Figure 3.17 : Hystérésis de la vanne pneumatique

Une étude plus fine de la vanne montre que celle-ci réagissait à partir d'un niveau de tension de 2Volt. Il est intéressant de pouvoir ajouter au signal de commande de la vanne cette constante de 2Volt. Ce relèvement de tension est réalisé sur la carte de développement du microcontrôleur fournie par la VUB. Ce relèvement se fait à l'aide d'un circuit sommateur.
Ce dernier est constitué d'un amplificateur opérationnel.

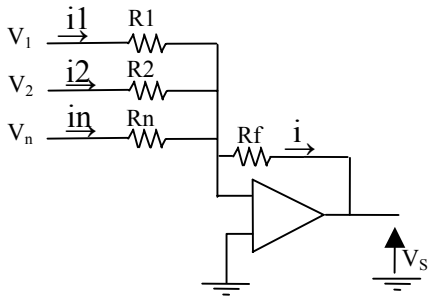


Fig. 3.18 : Amplificateur opérationnel en montage sommateur

En considérant le fonctionnement de l'amplificateur opérationnel en régime non saturée et l'impédance d'entrée de celui-ci comme infinie, on peut écrire que :

$$V_s = -R_f . i$$
$$i = i_1 + i_2 + ... + i_n$$

On obtient alors : $V_s = -R_f \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + ... + \frac{V_n}{R_n} \right)$

On constate donc que la tension de sortie vaut donc la somme des tensions d'entrées. Mais cette tension de sortie est de signe opposé aux entrées c'est pourquoi on connecte la sortie du sommateur à l'entrée d'un circuit inverseur, lui aussi constitué d'un amplificateur opérationnel.

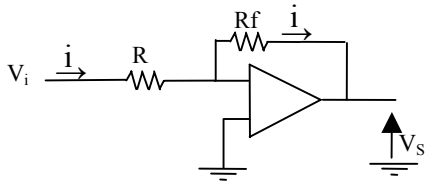


Fig. 3.19 : Amplificateur opérationnel en montage inverseur

En posant les mêmes hypothèses et en appliquant la même logique de résolution nous trouvons :

$$V_s = -\frac{R_f}{R} v_i$$

En posant $R_f = R$, on a : $V_s = -v_i$

La combinaison du montage sommateur et inverseur permet d'écrire :

$$V_s = R_f \left(\frac{V_1}{R_1} + \frac{V_2}{R_2} + ... + \frac{V_n}{R_n} \right)$$

C'est donc de cette manière que un signal constant de 2Volt est ajouté au signal PWM. Il est est obtenu à partir d'un simple diviseur résistif.

Dans un deuxième temps cette vanne sera remplacée par une vanne plus précise et comportant un régulateur de pression.
Il faudra tenir compte de ce cas de figure dans le programme assembleur ainsi que dans l'interface Visual Basic.

IV. PROGRAMMATION DU MICROCONTRÔLEUR MOTOROLA 68HC11

IV.1. Introduction :

L'étape suivante à réaliser dans le déroulement du projet est la programmation du microcontrôleur. Avant d'expliquer en détails les différentes étapes du programmes et les diverses sous-routines utilisées pendant son l'exécution, il est intéressant de rappeler brièvement les principes de fonctionnement du microcontrôleur et son rôle dans notre projet.

IV.2. Principes et rôle du microcontrôleur :

Une description générale des microprocesseurs a été faite en Annexes page 89. Nous allons introduire le rôle du MOTOROLA 68HC11 dans notre application et expliquer brièvement quelques-unes des ses caractéristiques.

Le microcontrôleur MOTOROLA 68HC11 va jouer un rôle-clef dans le fonctionnement de notre banc d'essai. En effet, tout le déroulement des différentes étapes de l'essai d'un muscle artificiel pneumatique sera supervisé par le microcontrôleur. C'est donc au travers de celui-ci que l'on pilotera la vanne pneumatique en lui fournissant la tension adéquate à ses bornes d'entrées. C'est également au travers du microcontrôleur que l'on va pouvoir effectuer les mesures de pression et de contraction de l'actionneur pneumatique. Les grandeurs analogiques (tensions), images des deux mesurandes converties en grandeur numérique, sont aisées à recueillir avec ce type d'équipement. On pourra les stocker ou les envoyer vers un autre terminal tel qu'un ordinateur.

Le microcontrôleur agira en conséquence, en fonction des éléments extérieurs précisés par l'utilisateur. Tel que les pressions maximum et minimum qui règneront à l'intérieur de l'actionneur, la période du signal fourni à la vanne…

Il communiquera donc en permanence avec un ordinateur car c'est par ce biais que l'utilisateur va paramétrer les conditions d'essai de l'actionneur.

La figure 4.1 illustre le brochage du MOTOROLA 68HC11 (version PLCC 68 broches).

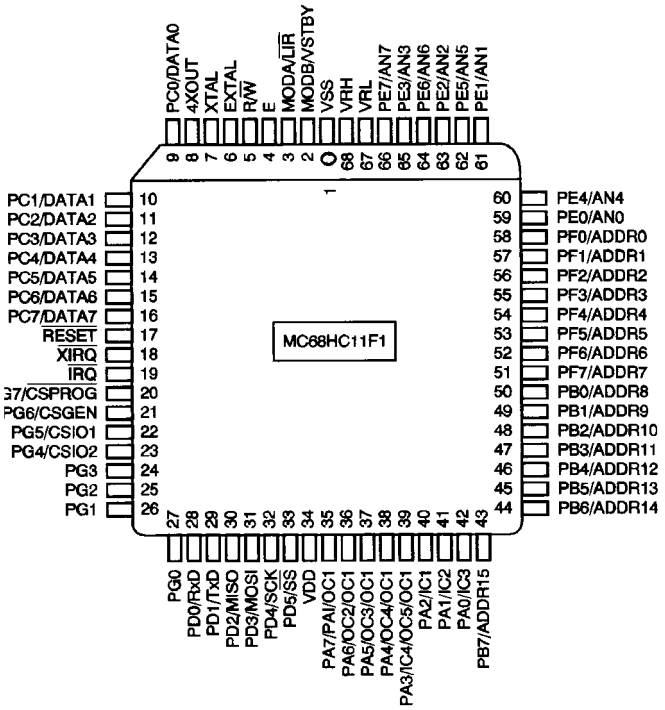


Fig. 4.1 : Brochage du MOTOROLA 68HC11

On remarque donc la présence de 54 broches qui permettent d'accéder directement à tous les octets des 6 ports de la machine.

Voici une brève description de ceux que l'on va utiliser pour notre application :

- Le port *A* (PA_0 à PA_7) est un port d'usage général qui permet les entrées/sorties sur 8 bits. Il dispose de 3 entrées (PA_0 à PA_2) de 3 sorties (PA_4 à PA_6) et de 2 broches bidirectionnelles (PA_3 et PA_7). Afin de définir le sens de fonctionnement de ces 2 broches bidirectionnelles, le MOTOROLA 68HC11 dispose d'un registre approprié : le registre *DDRA* (Data direction Register for port *A*). Pour des applications temporisées, les fonctions des broches du port *A* sont partagées avec un compteur *Timer* (Output Compare & Input Compare), interne au MOTOROLA 68HC11. Lorsque les broches adéquates sont utilisées en sortie, l'information est maintenue par des verrous internes au microcontrôleur.

- Le port *D* est, quant à lui, codé sur 6 bits (PD_0 à PD_5). Il possède, identiquement au port *A,* un registre interne permettant de définir le sens de transfert des données sur les différentes broches : il s'agit du registre DDRD.
  Comme le port *A*, le port *D* peut être lu à tout moment et lorsque ses broches sont utilisées en sortie l'information est maintenue par des verrous internes au microcontrôleur Motorola.
  Il est important de rajouter que le port *D* partage ses fonctions avec les deux interfaces permettant la communication série entre divers équipements et le microcontrôleur.

Ces deux interfaces sont les suivantes : *SCI* (Serial Communication Interface) et *SPI* (Serial Peripheral Interface). Celles-ci vont servir respectivement à communiquer avec l'ordinateur et donc l'interface développée à cet effet, et à communiquer avec les deux convertisseurs A/D utilisés, afin de collecter les informations fournies par ceux-ci. Nous décrirons plus tard le fonctionnement exact de ces interfaces et la manière de les utiliser.

- Le port *E* (PE_0 à PE_7) est quant à lui utilisable uniquement en entrée. C'est par celui-ci qu'on accédera au convertisseur A/D interne au MOTOROLA 68HC11. Mais pour deux raisons évoquées précédemment (bruitage des signaux et codage sur 8 bits uniquement) nous utiliserons des convertisseurs externes (LTC1286) dans le cadre de la mesure des grandeurs qui nous intéressent.

  Les autres ports du microcontrôleur ne sont pas utilisés pour notre application.

La figure 4.2 représente les ports du MOTOROLA 68HC11 et les différentes interfaces qui sont associées.

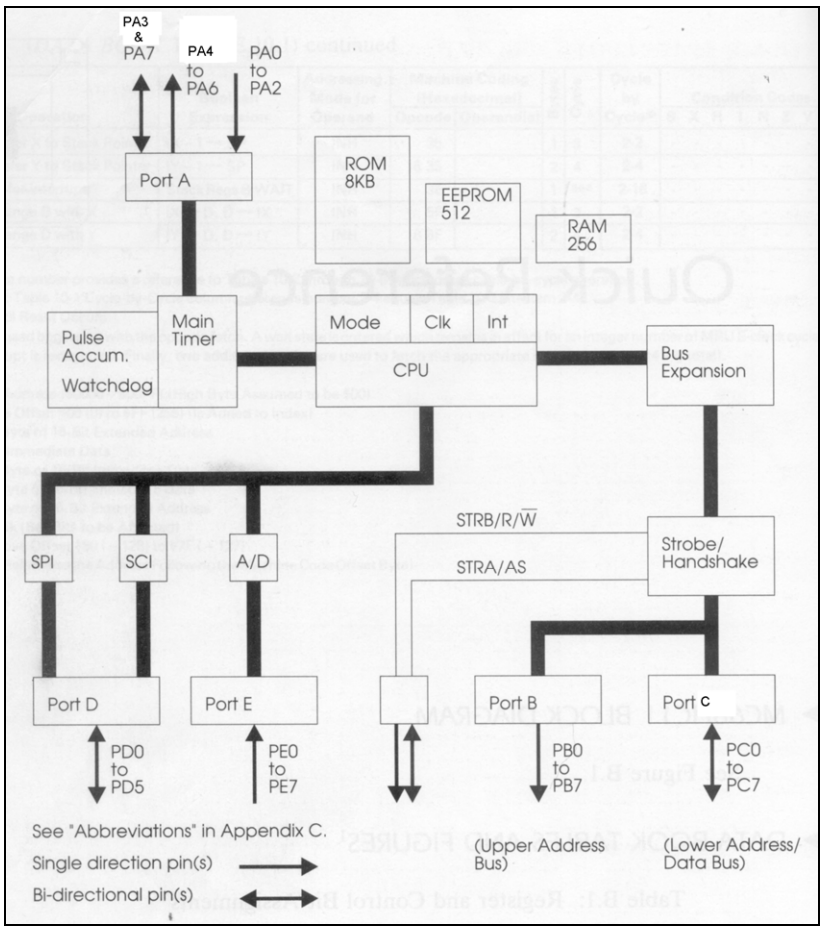


Fig. 4.2 : Ports et interfaces du MOTOROLA 68HC11

La description de n'importe quel microcontrôleur passe obligatoirement par une description des registres internes à celui-ci.

La figure 4.1 donne une représentation des ces registres internes.

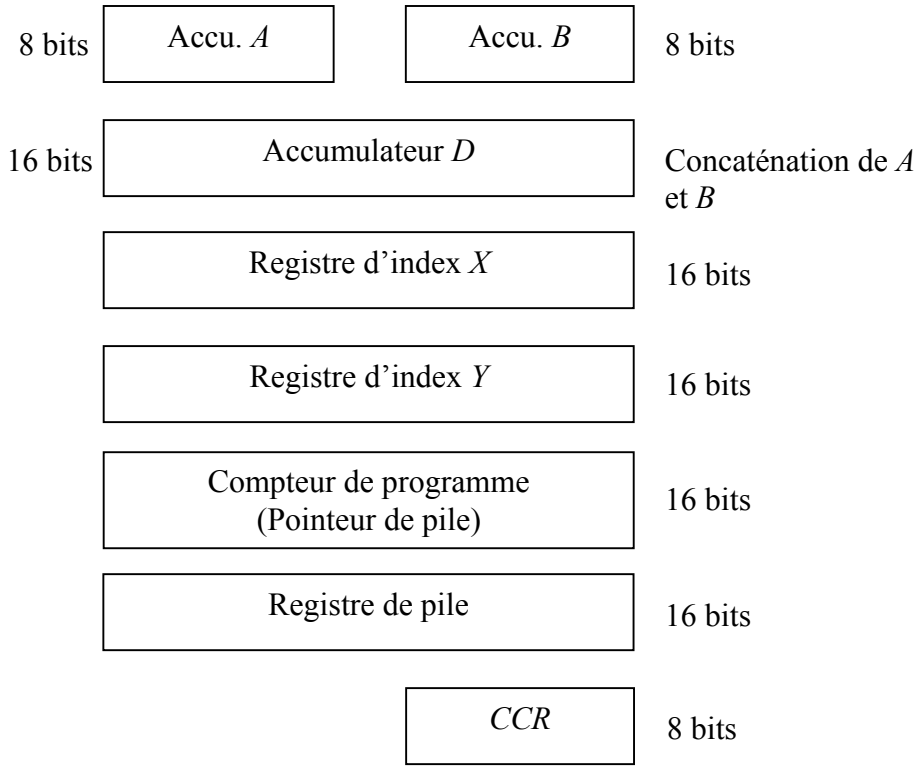


Fig. 4.1 : Registres internes

Voici une description de ces différents registres :

- Les accumulateurs *A*, *B* et *D* :
  les registres *A* et *B* sont des registres codés sur 8 bits chacun. Ce sont des registres à usage général, ils vont permettre de contenir les opérandes des instructions, ainsi que les résultats d'opérations logiques ou arithmétiques ou encore de manipulation de données.
  Parfois il est nécessaire de traiter avec des informations codées sur 16 bits, on utilise alors le registre *D* qui résulte en fait de la concaténation
  des registres *A* et *B*.

- Les registres d'index *X* et *Y* :
  les registres X et Y, utile notamment pour l'adressage indexé (voir point 3. « *Introduction à la programmation* »), sont codés sur 16 bits. Pour définir une adresse d'un périphérique où aller chercher une information, on ajoute, à la valeur contenue dans le registre X par exemple, un offset codé par exemple sur 8 bits.

- Le compteur de programme :
  le compteur de programme est en fait un registre codé sur 16 bits, il contient l'adresse de la prochaine instruction à exécuter. Il sera donc incrémenté automatiquement à chaque exécution d'une instruction.

- Le registre de pile :
  la pile est utile dans le déroulement du programme pour y sauver des données ou des adresses lors d'appels de sous-routine au cours de l'exécution. On définira la taille de la pile avec prudence car celle-ci ne doit pas déborder sur les informations sauvées dans la RAM. Ceci entraînant de grave erreur dans l'exécution du programme (*Stack overflow*).

- et le registre d'état :
  Celui-ci est codé sur 8 bits. Il contient en fait les indicateurs (flags), très utiles pour le microcontrôleur et les prises de décision que celui-ci doit faire.

S	X	H	I	N	Z	V	C

C = Carry
V = Overflow
Z = Zero
N = Negative
I = Interrupt mask
H = Half carry
X = Interrupt mask
S = Stop

Fig. 4.2 : Registre CCR

Le bit C est mis à 1 lorsque l'ALU effectue une retenue, il sert de neuvième bit pour les opérations de décalage, il est également indicateur d'erreur pour les opérations de multiplication et de division

Le bit V est mis à 1 lorsque une opération arithmétique entraîne un dépassement. Il n'est pas affecté autrement et vaut alors 0.

Le bit Z vaut quant à lui 0 lorsque une opération de décalage, arithmétique ou logique donne un résultat nul. Ce bit est par exemple utile avec les instructions de comparaison de deux grandeurs. Ces instructions (CMP) effectuent une soustraction.Si le résultat est nul, les deux grandeurs sont donc identiques et le bit Z a alors été mis à 0.

Le bit N est mis à 1 lorsque le résultat d'une opération arithmétique logique ou encore de manipulation est négatif.

Le bit I est un bit qui va en fait déterminer de manière globale si les interruptions sont masquées ou non. L'instruction pour mettre le bit à 1 est *SEI* et celle pour le mettre à 0 est *CLI.*

Le bit H est affecté par une retenue du bit 3 vers le bit 4 lors d'une opération arithmétique, son utilisation porte essentiellement sur des opérations avec des nombres BCD (Binary Coded Decimal).

Le bit X inhibe ou non les interruptions venant de la broche XIRQ (XIRQ déclenche une demande d'interruption après un reset).

Le bit S interdit, lorsqu'il est mis à 1 la mise en veille du microcontrôleur par l'instruction STOP. Si le microcontrôleur rencontre cette instruction et que le bit S est mis à 1 il va remplacer STOP par NOP (No Operation) et le déroulement du programme se fera.
Si par contre le bit S est à 0, le microcontrôleur se mettra en veille.

IV.3. Introduction à la programmation :

A l'heure actuelle les microcontrôleurs sont incapables de comprendre le langage humain. Ils utilisent en effet un langage particulier appelé langage machine.
Ce langage machine n'est pas aisé d'utilisation (0 ou 1) c'est pourquoi nous allons utiliser un langage de programmation. C'est ce dernier qui sera converti en langage machine afin de permettre au microcontrôleur de comprendre ce qui lui est demandé et ce qu'il doit exécuter. Cette conversion porte le nom de compilation.
Le langage machine évoqué ci-dessus n'est pas universel et utilisé identiquement par tous les microcontrôleurs existants. Chaque type de microcontrôleur possède une liste d'instruction qu'il comprend et est capable d'exécuter. Ces instructions sont donc une suite de chiffres binaires (0 ou 1) codée suivant une séquence déterminée et c'est cette séquence qui indiquera quel action le microcontrôleur doit faire.
Voici un exemple d'instruction en langage machine ; sous cette forme les instructions nous sont difficilement compréhensible et exploitable :

%1001100

Comme cité précédemment, nous utiliserons un langage de programmation qui nous permettra de manipuler aisément ces instructions et de les comprendre. Un de ces langages est l'assembleur.
Voici la même instruction évoquée plus haut et qui ordonne au 68HC11 d'incrémenter d'une unité la valeur contenue dans l'accumulateur *A*.

INCA

Généralement les instructions que le microcontrôleur va décoder sont constituées de deux parties : le code opération et l'opérande.
Le code opération, formant la première partie de l'instruction, indique en fait à l'unité centrale de traitement (CPU) ce qu'elle doit faire, quelle opération exécuter. L'opérande indique quant à lui sur quelle donnée l'opération doit agir. Nous traitons donc des instructions codées sur deux octets : un premier octet pour le code opération et un second pour l'opérande.

Ex : LDAA $1008

LDAA est donc le code opération et indique de charger une donnée située à l'adresse hexadécimale 1008 dans l'accumulateur *A*.
L'opérande dans ce cas ci est l'adresse hexadécimal 1008 indiquant ou aller chercher la donnée à charger.
Pour information, le caractère « $ » indique que la donnée qui suit est codée en hexadécimal. On rencontrera également « % » qui indique cette fois que celle-ci est codée en binaire.

Il existe également des instructions ne possédant pas d'opérande (ex. INCA) ou encore en possédant deux ou trois. A cela s'ajoute également des instructions dont le code opération est codé sur 2 octets afin d'augmenter le nombre possible d'instructions.

Afin de compléter notre introduction à la programmation, il est utile de parler des différents modes d'adressage que l'on peut utiliser.

Il existe l'adressage inhérent : pour ce type d'adressage, il n'y a pas d'adresse fournie avec l'instruction, celle-ci s'applique en fait toujours sur le même registre.

Ex : INCA
Incrémenter le contenu de l'accumulateur *A*.

Ex : CLRB
Efface le contenu de l'accumulateur *B*.

Ensuite vient le mode d'adressage immédiat : dans ce cas, la donnée sur laquelle s'applique l'instruction est contenue dans l'octet qui suit immédiatement le code opération. On occupe donc au minimum deux octets en mémoire puisqu'on doit stocker le code opération (un octet) et la donnée (un octet ou plus).

Ex : LDAA #%10101010
Charger dans l'accumulateur la valeur qui suit codée en binaire.

Ex : ADDA #$58
Additionner la valeur qui suit, codée en hexadécimal, au contenu de l'accumulateur *A*.

Pour le mode d'adressage direct, l'adresse de l'opérande est contenue dans un seul octet qui suit le code opération. On peut donc atteindre uniquement les mémoires comprises entre les adresses hexadécimales $00 et $FF.

Ex : LDAA $5C
Charger dans l'accumulateur *A* la donnée se trouvant à l'adresse $5C.

Il y a également le mode d'adressage étendue : cette fois-ci l'adresse n'est plus codée sur un seul octet comme pour le mode d'adressage direct mais sur deux octets.

Ex : LDAA $105C
Charger dans l'accumulateur *A* la donnée se trouvant à l'adresse $105C.

Pour l'adressage indexé, un décalage ou offset, codé sur un octet est ajouté à la valeur contenu dans un des registres d'index (*X* ou *Y*) pour former l'adresse ou se situe l'opérande.

Ex : LDAA $02, X
On ajoute ainsi la valeur hexadécimale $02 au contenu de *X* et le résultat forme l'adresse où se trouve la donnée à charger dans l'accumulateur *A*. Bien entendu, on devra charger l'accumulateur *X* avec une valeur adéquate. Ceci à l'aide d'instruction LDX par exemple.

Le dernier mode d'adressage est l'adressage relatif : ce mode ci est utilisé pour les instructions de branchement. Si la condition de branchement est vraie, un décalage codé sur un octet inclus dans l'instruction est ajouté au compteur de programme pour former l'adresse à laquelle doit se poursuivre l'exécution du programme

Ex : BEQ adr8
On compare par exemple deux nombres. Si ceux-ci sont égaux, la condition de branchement est vraie (Branch if Equal). Le programme continue alors à l'adresse formée par la somme de *adr8* et du compteur de programme.

Avant d'entamer les explications détaillées du programme en assembleur, il est important de différencier le saut à un sous-programme (branchement) et l'appel d'un sous-programme.
Dans le premier cas le programme effectue un branchement conditionnel ou non à l'adresse indiquée ; ceci sans sauvegarder les contenus des différents registres dans la pile. Si nécessaire, cette sauvegarde doit être réalisée par le programmeur.
Dans le second cas, le microcontrôleur sauvegarde les registres internes dans la pile, effectue le branchement à l'adresse indiquée, exécute le code contenu dans la sous-routine et revient à l'adresse suivant l'appel de la sous-routine avec l'instruction RTS (Return To Sub-Routine), ceci en récupérant les valeurs des registres à partir de la pile.

IV.4. Programme en Assembleur :

a) introduction :

On a vu que dans le cas où on utilisait la vanne FESTO, il était nécessaire d'ajouter un signal constant de 2V au signal PWM de commande de vanne. Il est également intéressant d'effectuer une procédure d'initialisation. Cette procédure va déterminer le rapport cyclique exact donnant le niveau de pression maximum désiré par l'utilisateur. On procède de la manière suivante : on part d'un rapport cyclique de départ (correspondant à un certain niveau de tension). On ajoute une valeur afin de l'augmenter. La tension au borne de la vanne va donc augmenter ainsi que la pression. On arrête lorsque l'on a atteint la pression désirée. Il en est de même pour le niveau de pression minimum. La pression régnant à l'intérieur de l'actionneur est bien entendu connue grâce au capteur de pression placé à l'intérieur du muscle.
Etant donné que la vanne utilisée est lente, on va exécuter une boucle d'attente après chaque modification du rapport cyclique. On lui laisse donc le temps de réagir à la tension de consigne. Cette boucle d'attente dure un peu plus d'une demi seconde.

Cette étape d'initialisation ne sera pas nécessaire lors de l'utilisation d'une vanne avec un régulateur de pression précis. Ce dernier garantit un niveau de pression pour une consigne de tension (généralement 1V/1Bar). On sait donc que si on applique par exemple 5 V aux bornes de la vanne, une pression de 5 bars régnera à l'intérieur de l'actionneur (aux incertitudes près).
On a donc défini deux mode de fonctionnement : le mode A s'associant à la vanne FESTO utilisé actuellement. Le mode B correspond à un test dont la vanne comporte un régulateur précis.

Le choix du mode de déroulement du test de l'actionneur se fait sur l'interface. Celle-ci enverra le caractère *a* ou *b* au microcontrôleur suivant le choix du mode et permettra ainsi à celui-ci d'adapter le programme en conséquence.

Pour solliciter l'actionneur, on souhaite le soumettre à une pression variant périodiquement. L'allure du cycle de pression sera de type triangulaire. On partira d'une pression minimale pour atteindre la pression maximale, ceci sur une durée valant la demi-période. La pression minimale désirée sera ensuite rétablie, toujours en une demi-période.

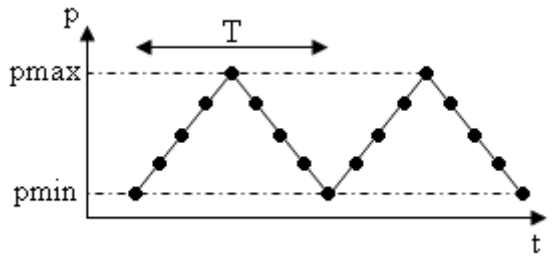


Fig. 4.3 : Evolution souhaitée de la pression dans l'actionneur

Afin de faire varier la pression dans l'actionneur, on fera varier la tension de commande de la vanne et donc le rapport cyclique du signal PWM. Voici l'allure d'un signal de ce type :

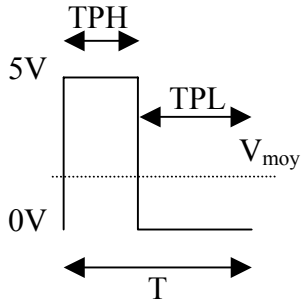


Fig. 4.4 : signal PWM

La tension moyenne est déterminé par :

$$Vmoy = 5.\frac{TPH}{T}$$

Pour notre application la période du signal PWM a été choisie constante et vaut 1ms.
Tout au long du cycle, des mesures de pression et de contraction de l'actionneur seront réalisées et envoyées à l'interface. Le nombre de mesures par cycle est paramétré dans cette dernière.

b) organigrammes :

Pour une compréhension plus aisée des différentes étapes du programme en assembleur, une représentation de celui-ci sous forme d'organigramme a été établie. De plus, dans les étapes, les actions ou comparaisons sont exprimées en pseudo langage, c'est-à-dire un langage à mi chemin entre le langage humain et le langage de programmation.
Voici les différentes variables utilisées dans l'organigramme, il s'agit des mêmes variables utilisées dans le programme en assembleur :

- *Patm* et *LongIni* : ils correspondent respectivement aux mesures de pression atmosphérique et de longueur initiale du muscle. Ces mesures sont effectuées par le microcontrôleur.

- *Pmax* et *Pmin* : il s'agit des consignes de pression maximale et minimale. Ces valeurs sont encodées par l'utilisateur dans l'interface Visual Basic.

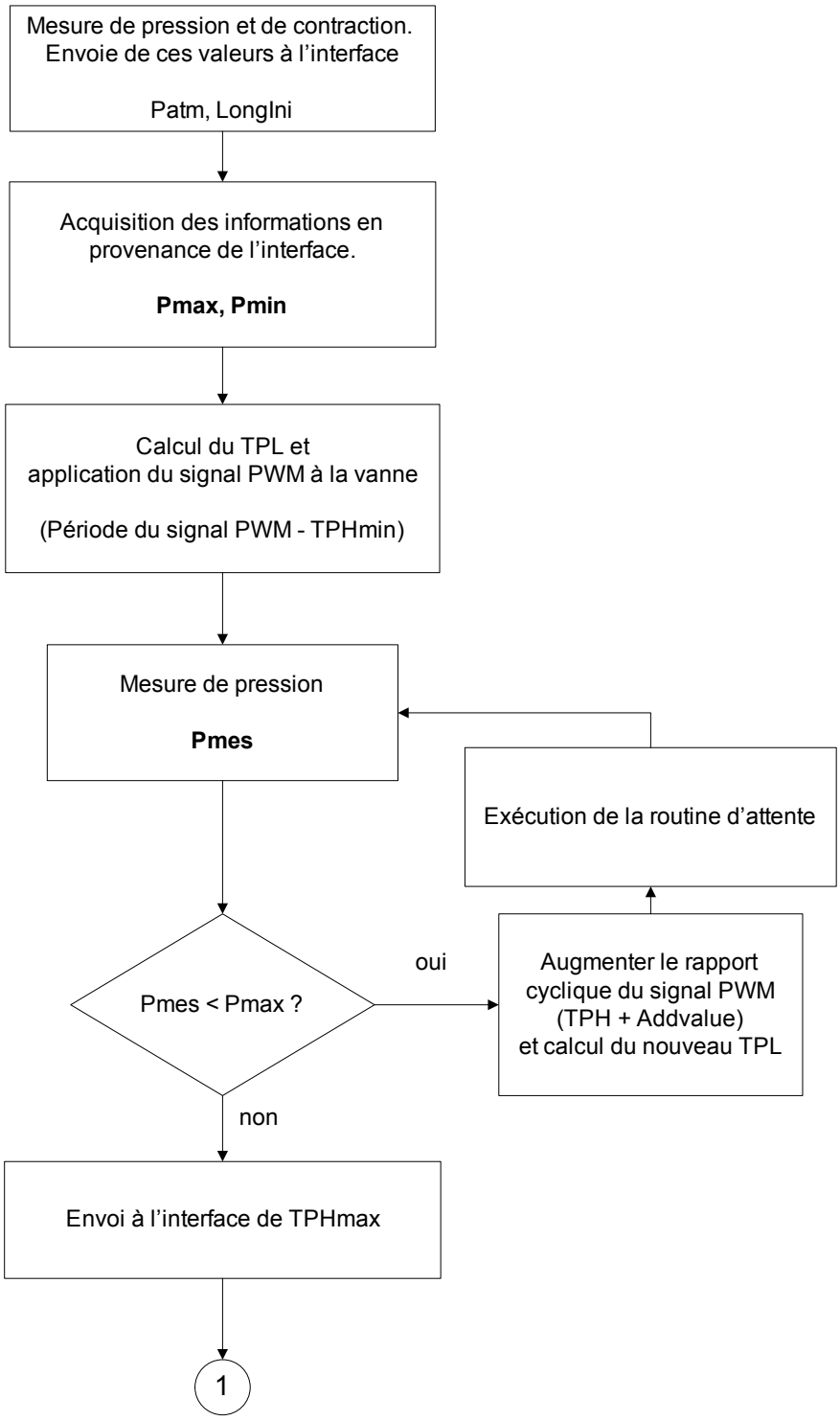
- *TPH* : (= Time Pin High) il s'agit en fait du temps passé à l'état haut (5 Volt) par la broche sur laquelle sera établi le signal PWM (voir figure 4.4). Le *TPHmin* correspond au TPH donnant le niveau de pression minimale. Idem pour *TPHmax* et la pression maximale.

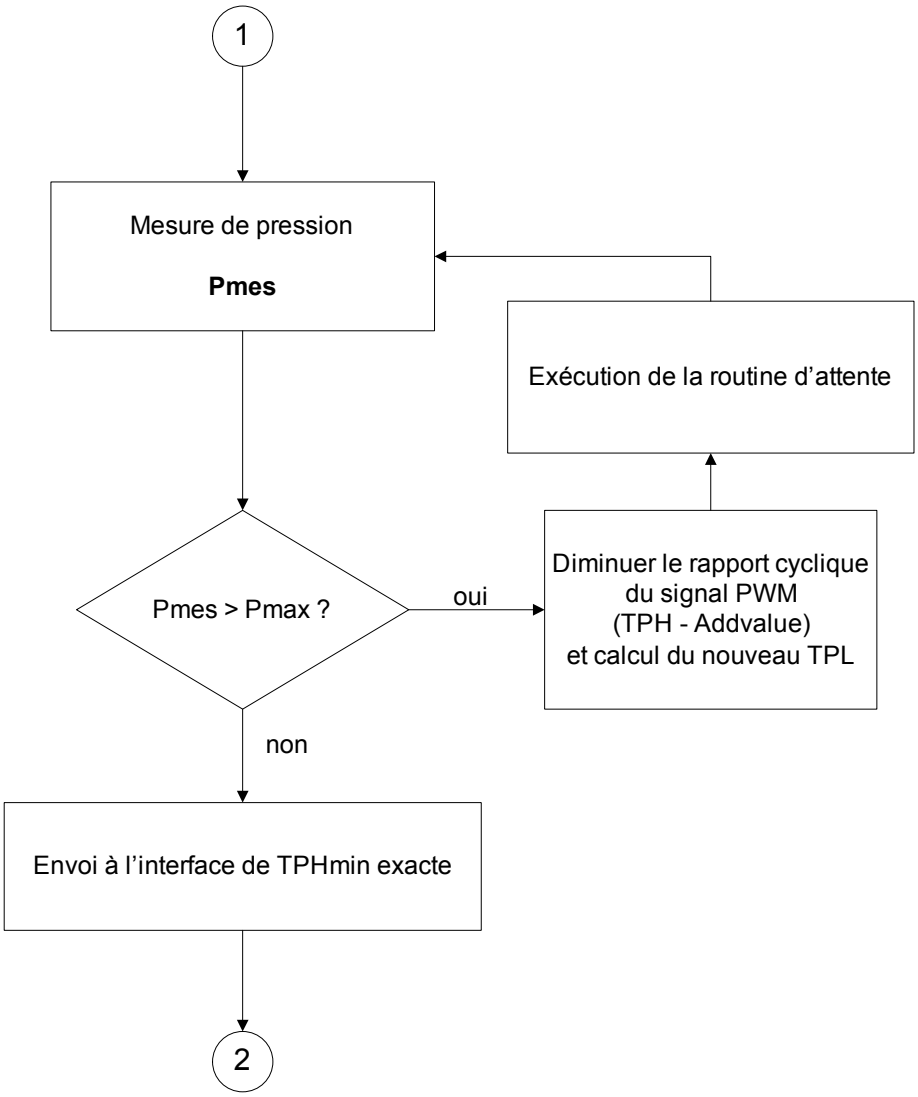
- *Addvalue* : il s'agit de la valeur qui sera ajoutée ou soustraite à *TPH* afin de modifier le rapport cyclique du signal PWM et donc la tension moyenne. Cette valeur est calculée dans l'interface.

- *TPL* : (= Time Pin Low) il s'agit cette fois du temps passé à l'état bas (0 Volt) par la broche sur laquelle sera établi le signal PWM (voir figure 4.4). Celui-ci est calculé dans le microcontrôleur. On soustrait la valeur de *TPH* à la période du signal PWM. TPH et TPL sont exprimé en nombre de cycles d'horloge du microcontrôleur. Chaque cycle dure 0,5µs.

- *Pmes* : Il s'agit de la pression mesurée à l'intérieur du muscle artificiel. L'information est fournie par le microcontrôleur et est codée sur 12 bits.

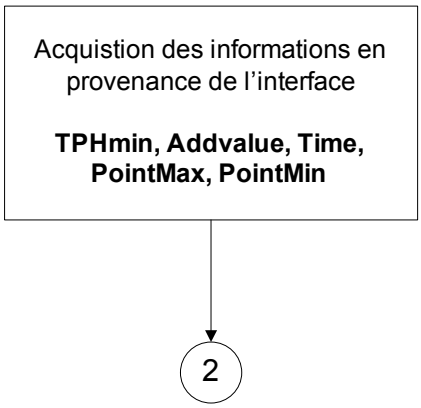
Si l'utilisateur choisi le mode de fonctionnement avec la vanne FESTO actuelle, autrement dit le mode A, voici le déroulement de l'étape d'initialisation :





Le microcontrôleur attend maintenant des données qui lui sont nécessaires pour le déroulement de la suite du programme. A cet instant, le microcontrôleur exécute une partie identique au mode de fonctionnement *B*.

Par contre, si l'utilisateur choisit le mode de fonctionnement *B*, l'étape d'initialisation n'est plus nécessaire. Le microcontrôleur se contente alors d'acquérir les informations nécessaires à la suite du programme.



La suite du programme est commune aux deux modes de fonctionnement, on peut alors définir certaines variables qui vont nous servir au cours du déroulement de celui-ci.

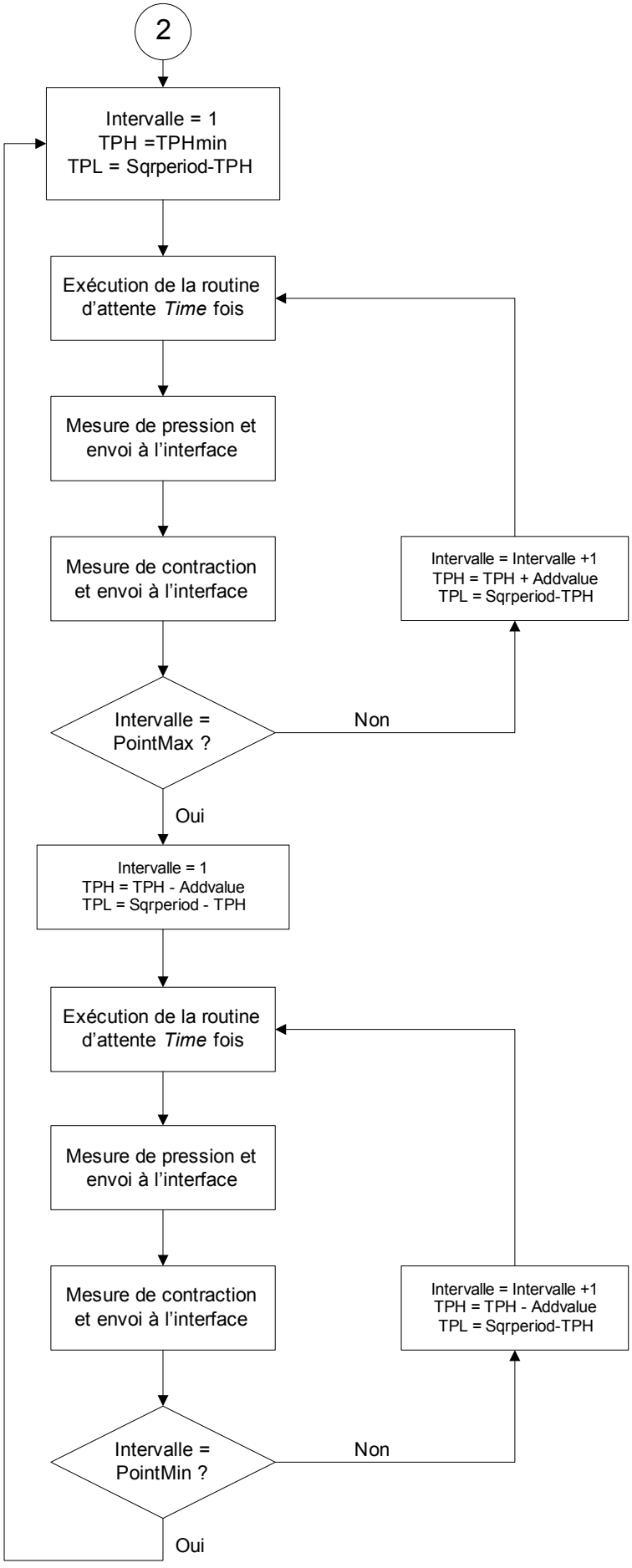
- *PointMax* : le cycle de pression est divisé en un certain nombre d'intervalles. *PointMax* correspond alors au $X^{ième}$ intervalle à partir duquel le rapport cyclique du signal PWM sera diminué au lieu d'être augmenté. On part donc de Pmax pour atteindre Pmin à l'intérieur de l'actionneur. Cette valeur est calculée dans l'interface et vaut :

  PointMax = (Nombre de Point par cycle / 2) + 1

- *PointMin* : cette variable joue un rôle identique à la précédente ; il s'agit du $X^{ième}$ intervalle après laquelle on entamera un accroissement de la pression et non plus une diminution de celle-ci. PointMin est également calculé dans l'interface et vaut :

  PointMin = (Nombre de Point par cycle / 2) – 1

- *Time* : Comme cité plus haut, les divers intervalles constituant le cycle de pression durent un certain temps. Dans le programme une sous-routine qui dure un temps déterminé a été établie. Cette sous-routine va être exécuté *Time* fois afin que l'intervalle dure le temps nécessaire.
  Time est calculé dans l'interface.


2
Intervalle = 1
TPH =TPHmin
TPL = Sqrperiod-TPH
Exécution de la routine d'attente *Time* fois
Mesure de pression et envoi à l'interface
Mesure de contraction et envoi à l'interface
Intervalle = Intervalle +1
TPH = TPH + Addvalue
TPL = Sqrperiod-TPH
Intervalle = PointMax ?
Non
Oui
Intervalle = 1
TPH = TPH - Addvalue
TPL = Sqrperiod - TPH
Exécution de la routine d'attente *Time* fois
Mesure de pression et envoi à l'interface
Mesure de contraction et envoi à l'interface
Intervalle = Intervalle +1
TPH = TPH - Addvalue
TPL = Sqrperiod-TPH
Intervalle = PointMin ?
Non
Oui

Il faut préciser que c'est l'interface qui arrêtera le déroulement du programme. Soit parce qu'on a atteint le nombre de cycles désiré par l'utilisateur, soit parce qu'une fuite a été détectée sur base des mesures de pression. Cet arrêt est réalisé en envoyant au microcontrôleur le caractère *s*. Une détection de la réception de ce caractère est réalisée plusieurs fois au cours du programme.

Maintenant qu'une description générale du déroulement du programme a été réalisée, on peut détailler les différentes sous-routines du programme. Le programme complet et commenté est présent en Annexes page 95. On remarque au début de celui-ci la définition de toutes les variables et des espaces mémoires réservés.

Une étape primordiale à réaliser est la communication série entre le microcontrôleur et l'ordinateur sur lequel s'exécutera l'interface Visual Basic ; cette communication s'établit à travers l'interface SCI.

c) Communication SCI :

Le système SCI possède un registre de donnée *SCDR* (Serial Communication Data Register). C'est dans ce registre que les données à transmettre vont être inscrite mais également les données reçues. Lorsqu'une donnée doit être transférée sur le port série, elle passe du registre *SCDR* à un registre tampon (shift register). C'est par ce même registre que les données arrivent avant d'être transférées dans le registre de données *SCDR*. Lorsque l'on veut émettre une donnée, le registre *SCDR* prend l'appellation *TDR* (Transmit Data Register).

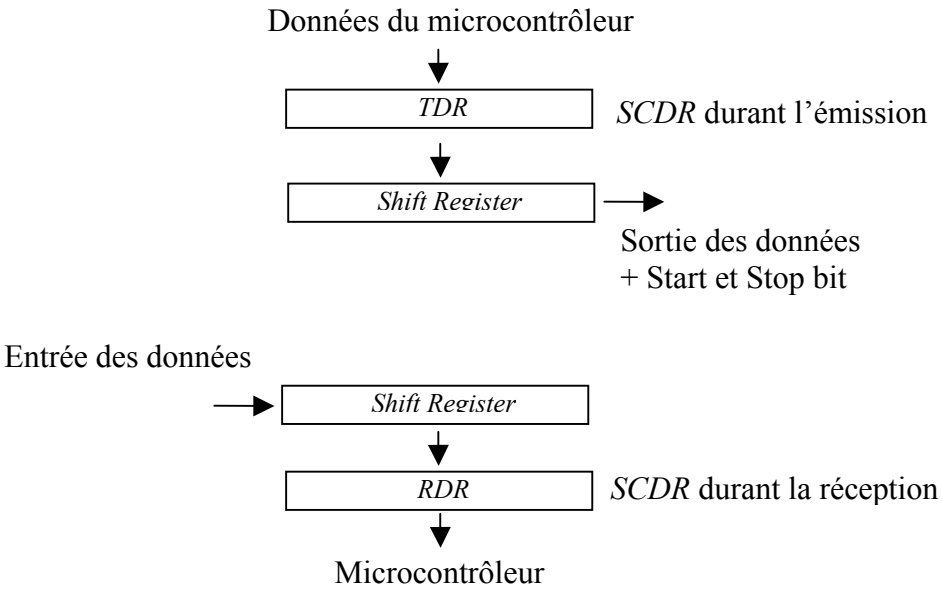


Fig 4.5 : Registre SCDR et Shift Register

Opération de transmission :

Pour émettre, le système écrit donc un octet dans le registre *TDR*. Cet octet est à son tour est transféré au registre de sortie (*shift register*). Ce dernier envoie automatiquement un bit de départ (Start Bit), les bits de données et pour clôturer un bit de fin (Stop bit).
Le système met également un indicateur (flag) à l'état haut chaque fois qu'il transfert les données du registre *TDR* au registre de sortie. Il s'agit du *TDRE* (Transmit Data Register Empty flag). Le programme détecte alors que le *TDRE* est mis à l'état haut et on peut alors écrire un autre octet dans le registre *TDR*. Une inscription d'un octet dans le registre *TDR* met l'indicateur *TDRE* à l'état bas.

Opération de réception :

Le transfert de la donnée du registre intermédiaire (*shift register*) au registre de donnée *SCDR* met l'indicateur *RDRF* (Receive Data Ready Flag) à l'état haut. Le programme sait donc que la donnée est prête à être lue. Lorsque la donnée reçue est lue dans le registre *RDR*, l'indicateur *RDRF* est remis à l'état bas. L'octet suivant peut alors être transféré vers le registre *RDR* à son tour.

Pour établir la communication, il faut au préalable configurer certains paramètres. Comme la communication n'est pas synchronisée, sur un signal d'horloge par exemple, il faut que les vitesses d'émission et de réception soient identiques sur les deux stations qui communiquent. On doit également informer les stations sur le nombre de bits constituant la donnée, le nombre de bits d'arrêt etc…
On doit autoriser l'émission et la réception de données pour pouvoir utiliser les entrées et sorties du port D (*TxD* et *RxD*) pour la communication série. Sans ceci les entrée/sorties sont utilisées en usage général. Voici les instructions qui permettent ces actions :

```
LDAA    #$30          A ← $30
STAA    BAUD,X        BAUD,X ← A : bits 4 et 5 à 1
BSET    SCCR2 ,X $C   SCCR2 : bits 2 et 3 à 1
BCLR    SCCR1,X $10   SCCR1 : bit 4 à 0
```

On utilise donc l'adressage indexé, le registre *X* contient la valeur hexadécimale $1000 (instruction *LDX*). Les variables *BAUD*, *SCCR2* et *SCCR1* contiennent les offsets à ajouter au registre *X* pour atteindre les registres dans lesquels on doit mettre à l'état haut ou bas certains bits. Ces variables et leurs contenus sont définis en tête du programme. Les deux premières instructions définissent une vitesse de transmission de 9600 Kbps. La troisième instruction autorise l'émission et la réception d'octets à travers l'interface SCI. Ceci se fait en mettant à 1 les bits 2 et 3 du registre *SCCR2* ($C = %1100). La quatrième instruction agit sur le registre *SCCR1*. Elle met le quatrième bit à l'état bas afin de choisir la configuration d'échange avec 1 bits de départ, 8 bits de donnée et 1 bit d'arrêt.
La sous-routine permettant de charger l'octet du registre *SCDR* dans l'accumulateur *A* porte le nom de *SCIWAITA*, voici les instructions qu'elle comporte :

```
BRCLR   SCSR,X $20 *  Bit 5 à 0 ?
LDAA    SCDR,X        A ← SCDR,X
RTS                   Return from Subroutine
```

L'astérisque à la fin de la première ligne d'instruction signifie que tant que le bit 5 du registre *SCSR* (*RDRF*) est à l'état bas, on revient au début de cette ligne. Dès que la donnée a été reçue, l'indicateur *RDRF* est mis à l'état haut, on passe alors à la deuxième instruction qui indique de charger dans l'accumulateur *A* le contenue du registre *SCDR*. L'instruction *RTS* est nécessaire pour retourner à l'endroit ou l'on a appelé la sous-routine. Une sous-routine identique chargeant la valeur du registre *SCDR* dans l'accumulateur *B* est utilisé et porte le nom de *SCIWAITB*.
Ces deux sous-routines sont utilisées pour acquérir les données en provenance de l'interface.
Pour envoyer les contenus des accumulateurs *A* et *B* à l'interface, on utilise alors les sous-routine *SENDA2PC* et *SENDB2PC*. Voici la première :

```
BRCLR    SCSR,X $80 *   Bit 7 à 0?
STAA     SCDR,X         SCDR,X ← A
RTS                     Return from Subroutine
```

Avec la première instruction on attend que l'indicateur *TDRE* soit mis à l'état haut, ce qui indiquerait que le registre *SCDR* est libre. On charge ensuite le contenue de l'accumulateur *A* (ou *B*) dans le registre *SCDR* (instruction 2).

d) Communication SPI :

L'interface SPI (Serial Peripheral Interface) va permettre la communication entre les convertisseurs A/D des mesures de pression et de contraction et le microcontrôleur. Ce système est assez pratique puisque il peut être utilisé avec des périphériques numériques venant de différents fabricants.
L'échange des données sera synchronisé sur un signal d'horloge. La fréquence de ce signal est égale à celle du microcontrôleur ou à une fraction de celle-ci. Pour ces échanges d'informations, un périphérique doit jouer le rôle de maître et les autres d'esclaves. Dans notre cas, le maître sera le MOTOROLA 68HC11 et les esclaves seront les deux convertisseurs LTC1286.
Pour l'interface SPI, certains registres vont jouer un rôle-clef. On peut alors citer le registre de contrôle *SPCR* (Serial Peripheral Control Register), le registre d'état *SPSR* (Serial Peripheral Status Register) et le registre de donnée *SPDR* (Serial Peripheral Data Register).
Il y a également le registre permettant de définir la direction des informations, il s'agit du *DDRD* (Data Direction Register for port D), celui-ci jouera un rôle essentiel dans le système SPI.
Les différents signaux qui interviendront vont donc s'établir sur certaines broches du port D. En effet, la broche PD2 supportera le signal *MISO* (Master In Slave Out). La broche PD3 se chargera du signal *MOSI* (Master Out Slave In). Sur la broche PD4 s'établira le signal *SCK* (Serial Clock), il s'agit du signal d'horloge. Pour finir, la broche PD5 sera utilisée pour le signal *~SS* (Slave Select). Comme nous utilisons 2 convertisseurs externes, la broche PD5 sera remplacée par les broches PA3 et PA4. Elles serviront respectivement pour le convertisseur A/D relatif à la contraction et pour celui de la pression.

```
LDAA    #$38        A ← $38
STAA    DDRD,X      DDRD,X ← A : bits 3, 4 et 5 à 1
LDAA    #$54        A ← $54
STAA    SPCR,X      SPCR,X ← A : bits 2, 4 et 6 à 1
```

Les deux premières lignes activent les bornes *MISO* et *SCK*, la borne ~*SS* est utilisée comme broche à usage général. Ceci se fait en mettant, dans le registre *DDRD*, les bits 2 et 4 à 1 et le bit 5 à 0. Les deux dernières mettent à 1 les bits 4 et 6 du registre *SPCR*. Ceci a pour conséquence l'activation de l'interface SPI et la mise du microcontrôleur en mode maître.
Voici comment se déroule l'échange de données : La borne D_{out} du convertisseur A/D est connecté à la borne *MISO*, la borne ~*CS* du LTC1286 sera connectée soit à la broche PA3 soit à la broche PA4.
Pour commencer à envoyer les données numériques, le convertisseur doit détecter sur sa borne ~*CS* un flanc descendant. Celui-ci doit être suivi par un train d'impulsion provenant de la borne *SCK* connectée à la borne *CLK* du convertisseur. Dans notre cas la borne *MISO* ne sera pas utilisée mais pour que le signal d'horloge soit établi (train d'impulsions) on doit y envoyer un octet.
Dès que la donnée du convertisseur est reçue, l'indicateur *SPIF* (bit 7) du registre *SPSR* est mis à l'état haut. On peut alors lire la donnée dans le registre *SPDR*, cette lecture à pour conséquence de remettre l'indicateur à l'état bas. La donnée complète en provenance du convertisseur est codée sur 12 bits, on la recevra donc en deux étapes et un petit traitement de celle-ci sera nécessaire.
Voici la routine qui permet de collecter les informations à travers l'interface SPI ; avant l'appel de cette sous-routine, un flanc descendant sur la broche PA3 ou PA4 est effectué suivant le fait que ce soit la contraction ou la pression que l'on veut mesurer :

```
GETSPI  LDAA    #$A5            A ← $A5
        STAA    SPDR,X          SPDR,X ← A : envoi d'un octet
POLL2   TST     SPSR,X          Test du bit 7 de SPSR,X
        BPL     POLL2           Si bit 7 = 0 alors on va à POLL2
        LDAA    SPDR,X          A ← SPDR,X
        STAA    SPIMSB          SPIMSB ← A
        LDAA    #$A5            A ← $A5
        STAA    SPDR,X          SPDR,X ← A : envoi d'un octet
POLL    TST     SPSR,X          Test du bit 7 de SPSR,X
        BPL     POLL            Si bit 7 = 0 alors on va à POLL
        BSET    PORTA,X $18     PA3 et PA4 à l'état haut
        LDAA    SPDR,X          A ← SPDR,X
        STAA    SPILSB          SPILSB ←  A
        LDAA    SPIMSB          A ← SPIMSB
        ANDA    #%00111111      ET logique
        LDAB    SPILSB          B ← SPILSB
        LSRD                    Translation à droite
        RTS                     Return from Subroutine
```

Comme expliqué plus haut, une donnée doit être envoyé via la borne *MOSI*. C'est le rôle des deux premières instructions. La valeur de la donnée envoyée n'a pas d'importance.
On attend ensuite que le bit 7 du registre *SPSR* soit mis à l'état haut ; alors, le 1er octet qui est reçu (partie haute de la donnée) est chargé du registre *SPDR* vers l'accumulateur *A*. On renvoie ensuite une donnée pour permettre au deuxième train d'impulsions du signal d'horloge de s'établir et ainsi pouvoir recevoir la partie basse de la donnée.
Dès que celle-ci est reçue on remet les broches PA3 et PA4 à l'état haut (peu importe celle qui était à l'état bas) et on charge le contenu du registre *SPDR* vers l'accumulateur *B*.
On efface ensuite les deux derniers bits de la partie haute, et on effectue une translation vers la droite de la donnée complète.
On a donc l'information complète codée sur 12 bits et contenue dans l'accumulateur *D* qui découle de la concaténation des accumulateurs *A* et *B*.

e) Timer :

Les boucles d'attente ainsi que la gestion du signal PWM nécessitent l'utilisation du Timer. Le cœur de ce dernier est un registre de 16 bits, *TCNT*, qui s'incrémente à chaque pulsation de l'horloge. On peut venir lire la valeur de ce registre en permanence mais il nous est impossible de venir la modifier, seul le signal d'horloge peut l'incrémenter. Le facteur *prescale,* défini par les bits *PR1* et *PR0*, détermine combien de pulsations d'horloge sont nécessaires pour incrémenter d'une unité le registre *TCNT*. Ces bits PR1 et PR0 sont protégés contre toute modification et doivent alors être modifié durant les 64 premiers cycles après un reset. Un programme de base dans lequel on utilise le système Timer est similaire aux autres programmes : il faut en premier lieu configurer les registres de contrôle, ensuite écrire dans un registre de données si nécessaire ; après cela, on attend qu'un indicateur soit mis dans un certain état, on efface ensuite celui-ci et on vient écrire ou lire dans un registre de données si nécessaire.

Le module *Output Compare*, appartenant au système Timer, va nous permettre de sortir des signaux carrés tel qu'un signal PWM. On peut également déclencher une exécution d'une séquence sans pour autant générer un signal sur une sortie. Voici un schéma simplifié qui va nous permettre de comprendre ce qui se passe avec le module Output Compare :

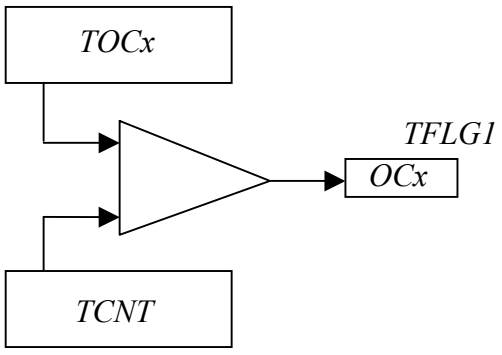


Fig. 4.10 : Module Output Compare

On vient écrire durant le programme dans le registre *TOCx*, le registre *TCNT* compte les pulsations de l'horloge. Dès que la valeur du registre *TCNT* atteint la valeur du registre *TOCx*, la comparaison est réussie et l'indicateur *OCxF* est mis à l'état haut. Ces indicateurs sont situés dans le registre *TFLG1*. La valeur inscrite dans le registre *TOCx* va être comparé au registre *TCNT*, il faut donc qu'elle soit codée sur 16 bits.
Si la comparaison est réussie et que donc l'indicateur *OCxF* est mis à l'état haut, il se passe un événement qui dépend des registres *TCTL1* et *TMSK1*. La configuration des bits *OMx* et *OLx* dans le registre *TCTL1* détermine comment la broche de sortie *OCx* est affectée. Le bit d'interruption associé *OCxI* dans le registre *TMSK1* détermine si une interruption est générée dans le cas d'une comparaison réussie.

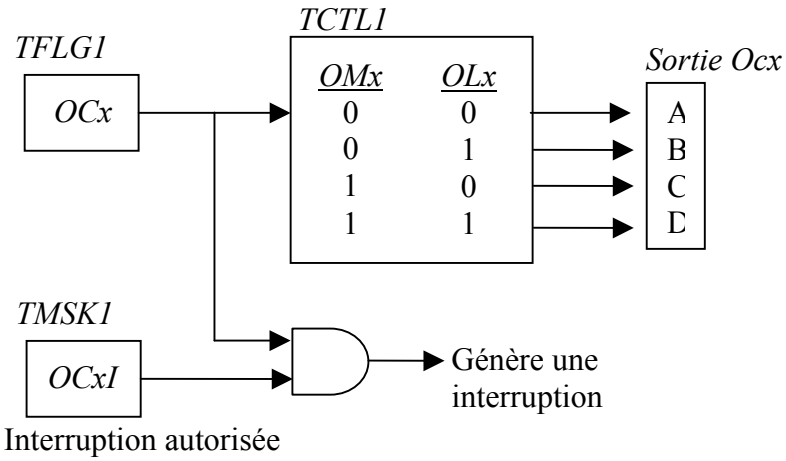


Fig. 4.11 : Cas d'une comparaison réussie

Le cas A n'entraîne pas d'action sur la sortie. Le cas B engendre un basculement de la sortie dans l'autre état. Le cas C conduit la sortie à l'état bas et le cas D à l'état haut.
Le module *Output Compare* est rattaché au port *A*, et c'est donc une de ces broches qui sera utilisée pour émettre le signal PWM.
Voici la configuration utilisée dans notre programme assembleur :

```
LDAA    #$10        A ← $10
STAA    TCTL1,X     TCTL1,X ← A : bit 4 à 1
LDAA    #$20        A ← $20
STAA    TMSK1,X     TMSK1,X ← A : bit 5 à 1
```

On va utiliser le module Output Compare 2 dans les boucles d'attente et le module Output Compare 3 pour la gestion du signal PWM. Ainsi lorsqu'une comparaison entre les registre *TOC2* et *TCNT* est logiquement vraie, il ne doit pas y avoir d'action sur la broche du port *A* associé à ce module (*PA6*) car on utilise uniquement ce module pour avoir une référence dans le temps (Fig. 3.10 cas A). Par contre lorsque la comparaison entre le registre *TOC3* et le registre *TCNT* est logiquement vraie, il doit se produire un basculement de la sortie *PA7* du port *A* (Fig. 3.10 cas B). On doit donc mettre le bit *OL3* à 1 et les bits *OM2*, *OL2* et *OM3* à 0, c'est le rôle des deux premières instructions agissant sur le registre *TCTL1*.

Comme le signal PWM doit s'exécuter en permanence, on va gérer le basculement d'un état à un autre dans une routine d'interruption.
L'interruption, comme son nom l'indique, interrompt le programme pour lancer l'exécution d'une série d'instructions. A la fin de ces instructions, le programme reprend alors son déroulement à partir de là ou il a été interrompu.
Il faut donc indiquer que le module Output Compare associé au signal PWM (OC3) doit avoir la faculté d'interrompre le programme lorsqu'une comparaison entre le registre *TOCx* et *TCNT* est logiquement vraie. Cette action se fait en configurant donc les bits *OCxI*. Le bit *OC3I* a donc été mis à 1 afin de générer l'interruption au moment voulu. C'est le rôle des deux dernières instructions mettant le cinquième bit du registre TMSK2 à l'état haut.
Maintenant que tout est configuré pour utiliser les modules Output Compare correctement, on peut décrire la sous-routine permettant de générer un signal PWM en sortie du microcontrôleur :

```
OC3INT      PSHA                        PUSH A
            PSHB                        PUSH B
            PSHX                        PUSH X
            LDX         #BASE           X ← BASE
            LDAA        #$20            A ← $20
            STAA        TFLG1,X         TFLG1,X ← A : bit 5 à 0
            LDD         TCNT,X          D ← TCNT,X
            BRSET       PORTA,X $20     Si PA5 est à 1 on va à
                        DRIVELOW        DRIVELOW
            ADDD        TPL             D ← D + TPL
            STD         TOC3,X          TOC3,X ← D
            PULX                        PULL X
            PULB                        PULL B
            PULA                        PULL A
            RTI                         Return from interrupt
DRIVELOW    ADDD        TPH             D ← D + TPH
            STD         TOC3,X          TOC3,X ← D
            PULX                        PULL X
            PULB                        PULL B
            PULA                        PULL A
            RTI                         Return from interrupt
```

Il ne faut pas perdre de vue que la sous-routine, commençant à l'étiquette *OC3INT*, est exécutée uniquement lorsqu'une interruption générée par l'Output Compare 3 apparaît.
Puisque l'on empêche le déroulement « normal » du programme, il est donc important de sauvegarder les valeurs des registres internes. En entrant dans l'interruption, ceux-ci peuvent en effet contenir des informations utiles au programme lorsque celui-ci quittera la routine d'interruption. Pour sauvegarder ces registres, on utilise la pile du microcontrôleur, mais il faut savoir que les données sauvegardées dans la pile le seront de la manière suivante : LIFO (Last In First Out). Cela signifie que si on sauvegarde en dernier lieu le registre *X*, par exemple, ce sera celui là qu'il faudra extraire en premier lieu car il se trouve au sommet de la pile. Les instructions PSH (Push) placent le contenu d'un registre dans la pile. Les instructions PUL (Pull) extraient une valeur de celle-ci pour la replacer dans le registre. C'est ce qui est fait au début et à la fin de la routine

d'interruption. Comme pour rentrer dans cette routine, le microcontrôleur a dû détecter que l'indicateur *OC3F* était à l'état haut, il faut remettre cet indicateur à l'état bas. C'est le rôle des instructions 5 et 6. Ensuite on teste si la broche *PA5* du port *A* était à l'état haut : si c'est le cas, on passe à l'étiquette *DRIVELOW* et elle est mise à l'état bas. On ajoute alors (*ADDD*) à la valeur du registre *TCNT* (chargée dans *D* par l'instruction *LDD*), la valeur de *TPL*. Le résultat de cette somme est stocké dans le registre *TOC3* afin d'être comparé dans la suite. Si par contre la sortie *PA5* était à l'état bas, elle est mise à l'état haut et on ajoute alors au contenu du registre *TCNT* la valeur de *TPH*.

On procède d'une manière plus simple pour les boucles d'attente. Elles font partie intégralement du programme et ne sont donc pas exécutées en cas d'interruption, il ne faudra donc pas sauver le contenu des registres sur la pile. Ensuite, une comparaison logiquement vraie n'entraîne pas de basculement d'une sortie.
On attend juste que le compteur TCNT arrive à la valeur contenue dans TOC2 et on continue la suite du programme. Voici la boucle d'attente utilisée lors de l'étape d'initialisation, après chaque modification du rapport cyclique :

```
INIWAIT1    LDY     #0              Y ← 0
BOUCLE      LDAA    #$40            A ← $40
            STAA    TFLG1,X         TFLG1,X ← A : bit 6 à 0
            LDD     TCNT,X          D ← TCNT,X
            ADDD    #$EA60          D ← D + $EA60
            STD     TOC2,X          TOC2,X ← D
WAITWAIT    BRCLR   TFLG1,X $40 *   Test du bit 6
            INY                     Y ← Y + 1
            CPY     #20             Y = 20 ?
            BLS     BOUCLE          Si Y ≤ 20 on va à BOUCLE
            RTS                     Return from Subroutine
```

Le registre *Y* est utilisé pour exécuter un certain nombre de fois cette boucle d'attente (10 dans notre cas). Lorsque l'on rentre dans cette routine on initialise la contenu du registre *Y* avec la valeur 0. Ensuite avant de faire une opération avec le registre *TOC2*, on met l'indicateur *OC2F* à l'état bas (instructions 2 et 3). Dès que cela est fait on charge dans l'accumulateur *D* la valeur du compteur *TCNT*. On lui ajoute alors un certain nombre ($EA60 = 60 000 cycles d'horloge) et on place le résultat dans le registre *TOC2*. Ensuite à l'étiquette *WAITWAIT* on attend que l'indicateur *OC2F* passe à l'état haut car l'instruction rebranche le compteur au début de cette instruction tant que cet indicateur est à l'état bas. Quand l'attente a été faite, on incrémente le registre *Y* et on le compare à la valeur 10, si c'est inférieur on passe à l'étiquette *BOUCLE* sinon on retourne à l'endroit ou la sous-routine à été appelée.
Les boucles d'attente utilisées dans la phase de test (*CONFIGWAIT1* et *CONFIGWAIT2*) fonctionnent de la même manière sauf que le nombre de fois ou la boucle d'attente est exécutée (ici *Y*), qui est un paramètre déterminé par l'interface suivant les desideratas de l'utilisateur. Si la boucle est exécutée une seule fois, un délai de 10ms s'est écoulé.

V. INTERFACE VISUAL BASIC

V.1. Introduction à Visual Basic :

Maintenant que nous avons utilisé la programmation séquentielle (programme assembleur), nous allons nous orienter vers une programmation dite événementielle.
Les instructions ne seront plus exécutées les unes à la suite des autres, mais un événement déterminé déclenchera l'exécution d'une suite d'instructions.
De plus, la programmation en Visual Basic ne s'arrête pas à écrire simplement des instructions dans un langage de programmation, on a recours à des outils visuels.
En effet, dans les systèmes d'exploitation tel que Windows, les programmes doivent être capable d'interagir graphiquement avec l'écran, le clavier, la souris et au besoin l'imprimante.
Les langages plus anciens tel que le Basic, le Turbo Pascal, sont idéaux dans des environnements purement textuels. Ils deviennent néanmoins tout à fait inappropriés aux interfaces graphiques des ordinateurs actuels.

L'objectif de cette étape est donc de réaliser une interface graphique, facile d'utilisation, qui va permettre à l'utilisateur de définir les différents paramètres relatifs au test de l'actionneur pneumatique. En plus d'envoyer ces paramètres nécessaires au microcontrôleur, cette interface va recueillir toutes les mesures envoyées par le MOTOROLA 68HC11 et va utiliser ces données.
On va donc rechercher les extremums tout au long des cycles de tests, aussi bien pour la contraction et pour la pression. Lorsque l'on aura atteint un certain nombre de maximums, on calculera la moyenne sur cet échantillon, cette valeur moyenne sera ensuite stockée dans un fichier Excel créé à cette attention. On procédera identiquement pour les minimums.
Le fichier Excel, contenant les valeurs moyennes, sera régulièrement sauvegardé afin de conserver toutes les données en cas de problème.
On procédera également à une représentation graphique des cycles en cours, afin de voir le bon déroulement du test de l'actionneur.

Afin de programmer efficacement et d'obtenir rapidement des résultats, il est intéressant de respecter une certaine marche à suivre pour la réalisation de notre interface :

1. Déterminer ce que notre application doit faire.

2. Créer la partie visuelle de notre interface.

3. Ajouter le code associé aux éléments visuels, celui-ci automatisera d'une certaine manière notre application.

4. Tester l'application afin d'éliminer les erreurs éventuelles.

5. Une fois que le fonctionnement correct est assuré, on peut compiler le programme et en extraire un fichier exécutable afin de l'utiliser facilement.

V.2. Partie visuelle :

Dans cette étape, nous allons élaborer la partie visible du programme à laquelle l'utilisateur sera confronté.

Au départ, nous disposons d'une feuille vide, cette feuille constitue l'espace sur lequel on viendra ajouter tous des éléments. Ceux-ci peuvent être visibles ou invisibles à l'utilisateur lors de l'exécution du programme. Ces éléments ajoutés portent le nom de contrôle. On peut citer, pour exemple, les boutons, cases à cocher etc. comme contrôle visible et les contrôles tel que le timer ou encore le contrôle MSComm comme contrôle invisible.
Voici à quoi ressemble la feuille vierge de tout contrôle. Il s'agit du point de départ de l'application :

Fig. 5.1 : Feuille vide

A chaque contrôle est associé une série de propriétés utiles pour en définir les différents états. Il est inutile de toutes les détailler sauf celle qui nous concernent.
La propriété permettant de modifier le nom visible du contrôle est définie par :

Nomducontrôle.caption = « … »

La chaîne de caractère comprise entre les guillemets sera ainsi affichée sur le contrôle.

Un contrôle peut être rendu visible ou invisible suivant les évènements, la propriété qui permet de gérer cette action est exprimée sous la forme suivante :

Nomducontrôle.visible = *true*
= *false*

Mettre cette propriété à l'état logique *true* autorise au contrôle d'être visible. Dans le cas ou cette propriété est mise à *false*, le contrôle devient alors invisible à l'utilisateur.
Cette propriété nous sera très utile. Par exemple, pour que l'utilisateur passe d'une page à une autre, on rend alors la page visible ou invisible selon le choix de l'utilisateur.

Pour qu'un contrôle apparaisse à l'utilisateur en grisé, on utilise la propriété suivante :

Nomducontrôle.enabled = true
= false

Lorsque cette propriété est mise à *false*, le contrôle en question est mis en grisé ; celui-ci est alors inutilisable par l'utilisateur (mais bien visible). On utilisera cette propriété afin

d'empêcher l'utilisateur d'utiliser un contrôle. Par exemple, cela nous sera utile pour guider l'utilisateur : on veut que celui-ci clique d'abord sur un bouton avant de pouvoir cliquer sur un autre.

Lorsque l'on souhaite l'apparition d'une bulle informative avec la position de la souris au dessus d'un contrôle, on utilise :

Nomducontrôle.ToolTipText = « … »

On placera entre les guillemets le texte à afficher.

Toutes ces propriétés sont accessibles par la fenêtre des *Propriétés* lorsque l'on sélectionne le contrôle que l'on a mis sur la feuille ; néanmoins elles peuvent également être modifiées dans le code Visual Basic en les introduisant sous la forme citée.

Voici la feuille avec tous les contrôles, nous allons décrire ces derniers ainsi que leur rôles dans notre interface :

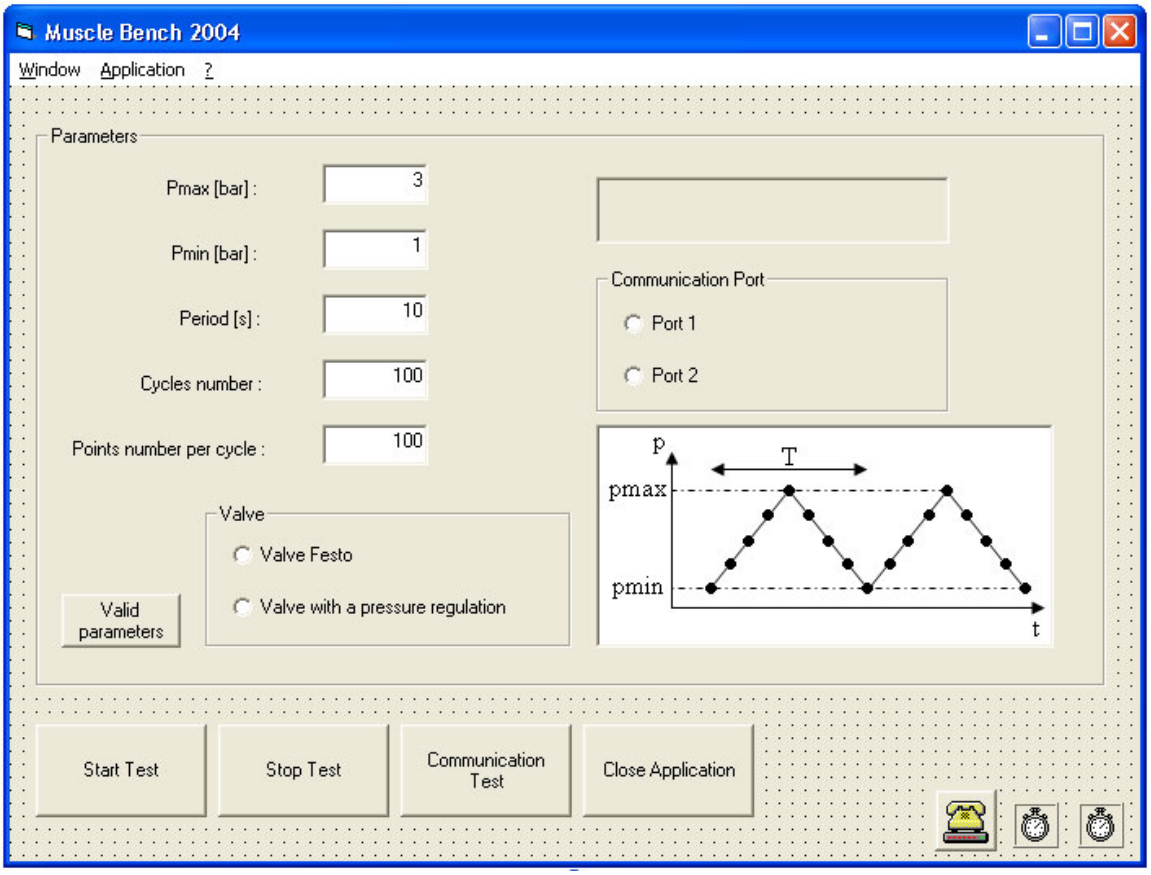


Fig. 5.2 : Feuille avec contrôles

Dans la partie supérieure gauche, on remarque un menu. Lorsque l'on clique sur le titre *Window* on peut choisir entre *Parameters* et *Graphics*. Cela va nous permettre de choisir quelle partie de l'interface on veut afficher : soit la partie ou l'on peut paramétrer tout le test (cas de la figure en Annexes page 92) soit la partie on l'on pourra visionner le tracé des différents points de mesures dans un graphique.

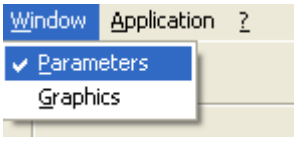


Fig. 5.3 : Menu de l'interface

On distingue que la fenêtre (*Parameters* ou *Graphics*) qui est affichée est caractérisée par un signe indiquant de quelle fenêtre il s'agit.
Le menu « Application » contient uniquement « Exit application » qui va nous permettre de quitter l'interface. Pour finir, le menu d'aide contient de l'information sur le développement de l'interface.

Lorsque l'on a introduit la présence des menus ainsi que leur rôle dans notre interface, nous avons indiqué qu'ils servaient à basculer d'une fenêtre à une autre.
En réalité, ces menus ne sont pas liés à des fenêtres mais à des contrôles conteneur, appelés également *frames*. Les *frames* sont des zones délimitées par un cadre, dans lesquelles on va déposer d'autres contrôles. Tous les contrôles déposés dans ces *frames* sont étroitement liés à ces zones. Donc, lorsque l'on rend une *frame* invisible (propriété *visible = false*) tous les contrôles lui appartenant deviennent également invisibles. Donc, en cliquant sur le menu *Window* et en sélectionnant la fenêtre « Parameters » on rend la *frame* associée visible (*frame* appelée *Parameters*) et on rend celle associé à « Graphics » invisible ; et vice versa.
Une *frame* peut contenir d'autres *frame*s qui lui seront associées. A la figure 5.2, on peut observer la *frame Parameters* contenant, elle-même, deux autres *frames* : *valve* et *Communication port*.

Dans la *frame Parameters*, on distingue 5 contrôles identiques appelés *Zones de Texte*. Ces contrôles servent donc lorsque l'utilisateur doit taper quelque chose ; dans le cas échéant, les valeurs des différents paramètres du test.
Il est plus aisé pour l'utilisateur de se voir proposer une valeur par défaut. Visual Basic nous permet de le faire, avec la propriété *Text* associée au contrôle.
Lorsque l'on viendra saisir le contenu des ces zones de texte pour entamer l'exécution du programme, Visual Basic les enregistre comme des chaînes de caractères, même si ces dernières ne contiennent que des nombres. On va donc les convertir en valeurs numériques, mais pour que tout se passe correctement, ces chaînes de caractères ne doivent rien contenir d'autres que des chiffres. Il va donc falloir réaliser un test des divers contenus de ces zones de texte. C'est le rôle du code associé au bouton *Valid Parameters*. Celui-ci vérifie également que les valeurs encodées sont cohérentes avec les capacités du banc d'essai.

Nous avons ainsi introduit la présence de contrôles tel que les boutons de commandes.
En plus de celui cité dans le paragraphe précédent, l'application en contient 4 autres. Ceux-ci ne sont pas associés à la *frame Parameters*. Donc lorsque cette dernière sera rendue invisible via le menu, ces 4 boutons resteront visibles.

On utilisera fréquemment la propriété *enabled* pour ces contrôles afin de guider l'utilisateur dans les actions qu'il doit faire ou qu'il ne peut plus faire pour le bon déroulement du programme.
Le fonctionnement de ces contrôles est simple : lorsque l'utilisateur va cliquer sur un bouton, la procédure événementielle lié à ce bouton va s'exécuter.
Dans le contrôle conteneur *Parameters*, on remarque la présence de deux autres *frames*. Chacune contient des contrôles appelés boutons d'option. Ces contrôles ont la particularité suivante : Dans chaque *frame*, un et un seul bouton d'option peut être sélectionné à la fois. La sélection d'un bouton d'option dans une *frame* entraîne donc la désélection des autres boutons d'option présents dans la *frame*. Ces contrôles vont nous servir à choisir le port de communication sur lequel le microcontrôleur MOTOROLA sera connecté ainsi que le choix de la vanne utilisée. L'utilisation d'une vanne avec régulateur précis ne nécessite pas de procédure d'initialisation.

Le schéma, représentant l'évolution de la pression au cours du temps, est présent afin d'aider l'utilisateur à comprendre la signification des divers paramètres qu'il doit définir. Ce schéma a été ajouté à l'aide du contrôle appelé PictureBox, permettant, entre autre, d'insérer dans l'application des fichiers images.

Au dessus de la *frame Communication Port*, on distingue un cadre. Il s'agit simplement d'un *label*. Les labels vont permettre d'afficher du texte simplement. Ils ont déjà été utilisés pour afficher du texte à coté des zones de texte (Pmax, Pmin...)

Voici la deuxième *frame* intitulée *Graphics* :

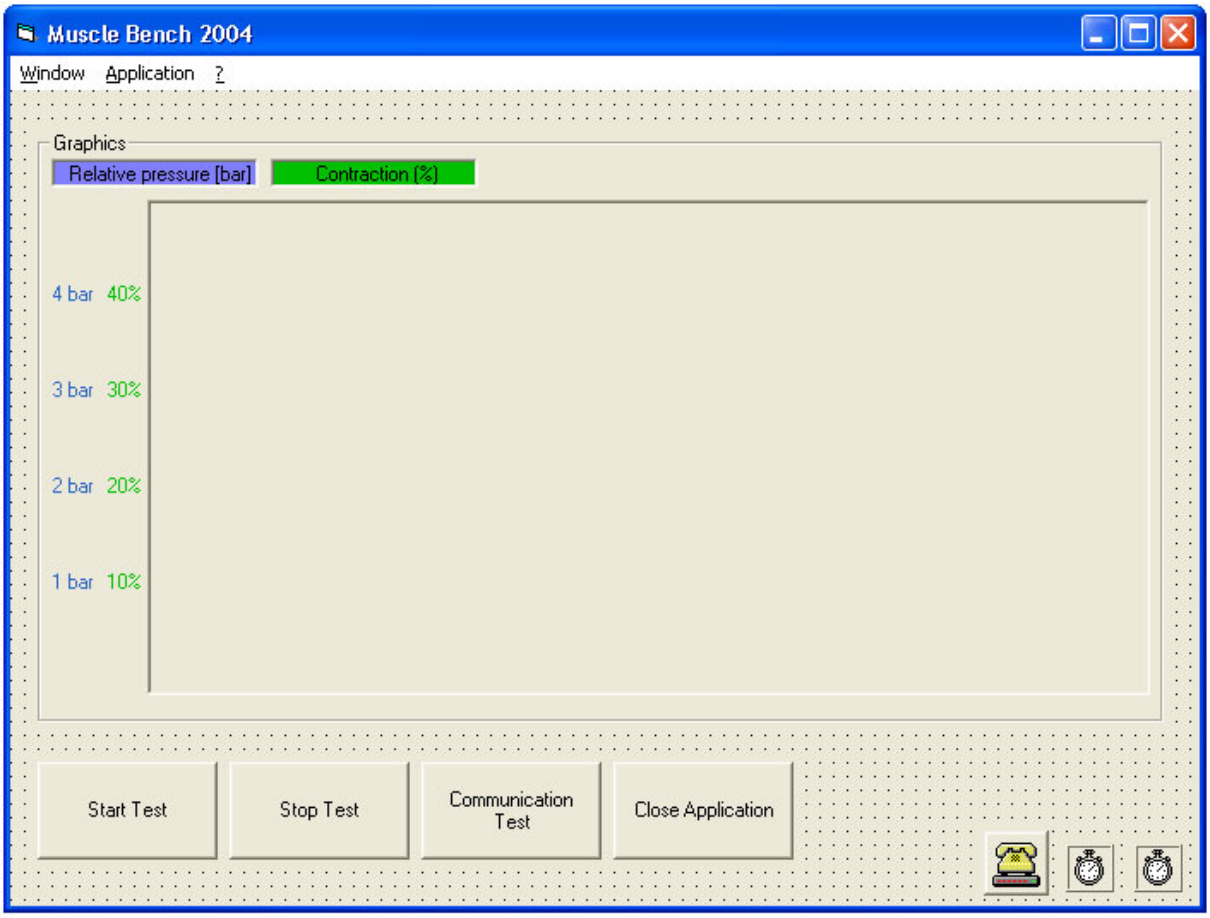


Fig. 5.4 : Frame Graphics

Dans cette *frame*, on réalisera une représentation graphique des points de mesure à l'aide du contrôle PictureBox.

Il nous reste à décrire quelques contrôles un peu plus particuliers ; le premier est appelé Timer, il est représenté par l'élément suivant:

Fig. 5.4 : Contrôle Timer

Bien qu'il soit déposé sur la feuille, ce contrôle est invisible à l'utilisateur une fois que l'application est lancée. Le Timer va permettre d'exécuter un code au bout d'un certain laps de temps par exemple.
On définit dans la propriété *interval*, relative à ce contrôle, un délai (défini en millisecondes). Lorsque le contrôle est activé (avec la propriété *enabled*), il va générer un événement à chaque fois que le délai est écoulé. À cet événement est déclenché une procédure événementielle appropriée. Elle porte le nom suivant :

NomduTimer_Timer()

On va utiliser ce contrôle lors du test de communication entre l'ordinateur et le microcontrôleur. Un deuxième Timer sera utilisé pour le rafraîchissement du graphique de la *frame Graphics*.

Le second contrôle qui sera lui aussi invisible à l'utilisateur quelles que soient les conditions porte le nom de MSComm ; il sera représenté par :

Fig. 5.5 : Contrôle MSComm

Il va permettre de gérer la communication série. Pour rappel, cette communication (RS232) va être utilisée pour les échanges d'informations entre le microcontrôleur et l'interface.

Le contrôle MSComm est associé à de nombreuses propriétés, nous allons décrire celles qui nous seront utiles.
Il faut en premier lieu choisir le port de communication. Sur la majorité des ordinateurs, il y a deux ports de communication série. On choisira le port à l'aide de cette propriété :

MSComm1.CommPort = 1 ou 2

MSComm1 est donc le nom du contrôle que l'on a déposé sur notre feuille.

Afin que la communication s'établisse correctement, il va falloir la configurer. Il faut en effet que les paramètres que l'on définit dans l'interface soient identiques à ceux du microcontrôleur. Pour ce faire, on utilise la propriété suivante :

MSComm1.Settings = "W,X,Y,Z"

W est la vitesse de transmission des données, elle est exprimée en Bauds. Le paramètre *X* impose si le paquet de données comporte ou non un bit de parité. *Y* exprime le nombre de bits constituant la donnée et *Z* le nombre de bit d'arrêt fermant la trame.
Tous les octets reçus par le port série sont stockés dans une mémoire tampon. Il existe une propriété indiquant le nombre d'octets non lus résidant dans cette mémoire :

MSComm1.InBufferCount = N

où *N* est un entier précisant le nombre de caractères non lus dans la mémoire tampon.

Lorsque l'on veut lire les données contenues dans la mémoire tampon, on doit encore préciser la longueur de celle-ci. Sinon Visual Basic lira tout ce qui est contenu dans cette mémoire lorsque l'on donnera l'ordre de lecture. Ce paramètre est définit avec la propriété suivante :

MSComm1.InputLen = M

où *M* est un entier précisant le nombre de caractères à lire dans la mémoire tampon.

Il peut être utile dans le programme de générer un événement lorsque l'on reçoit une donnée sur le port série. Pour permettre l'exécution du code lié à cet événement, il faut mettre une propriété à 1 : il s'agit de :

MSComm1.RThreshold = 1

A chaque fois qu'une donnée est reçue par le port série, Visual Basic exécute la procédure suivante : *MSComm1_OnComm()*. Cette procédure va nous être utile pour tester la communication entre l'interface et le microcontrôleur.
On peut établir quelque chose de similaire à l'envoie d'une donnée.

Pour lire la donnée dans la mémoire tampon, on utilise :

MSComm1.Input

On place donc le caractère lu dans la variable approprié.

Pour envoyer un caractère sur le port série, on utilise cette fois :

MSComm1.Output

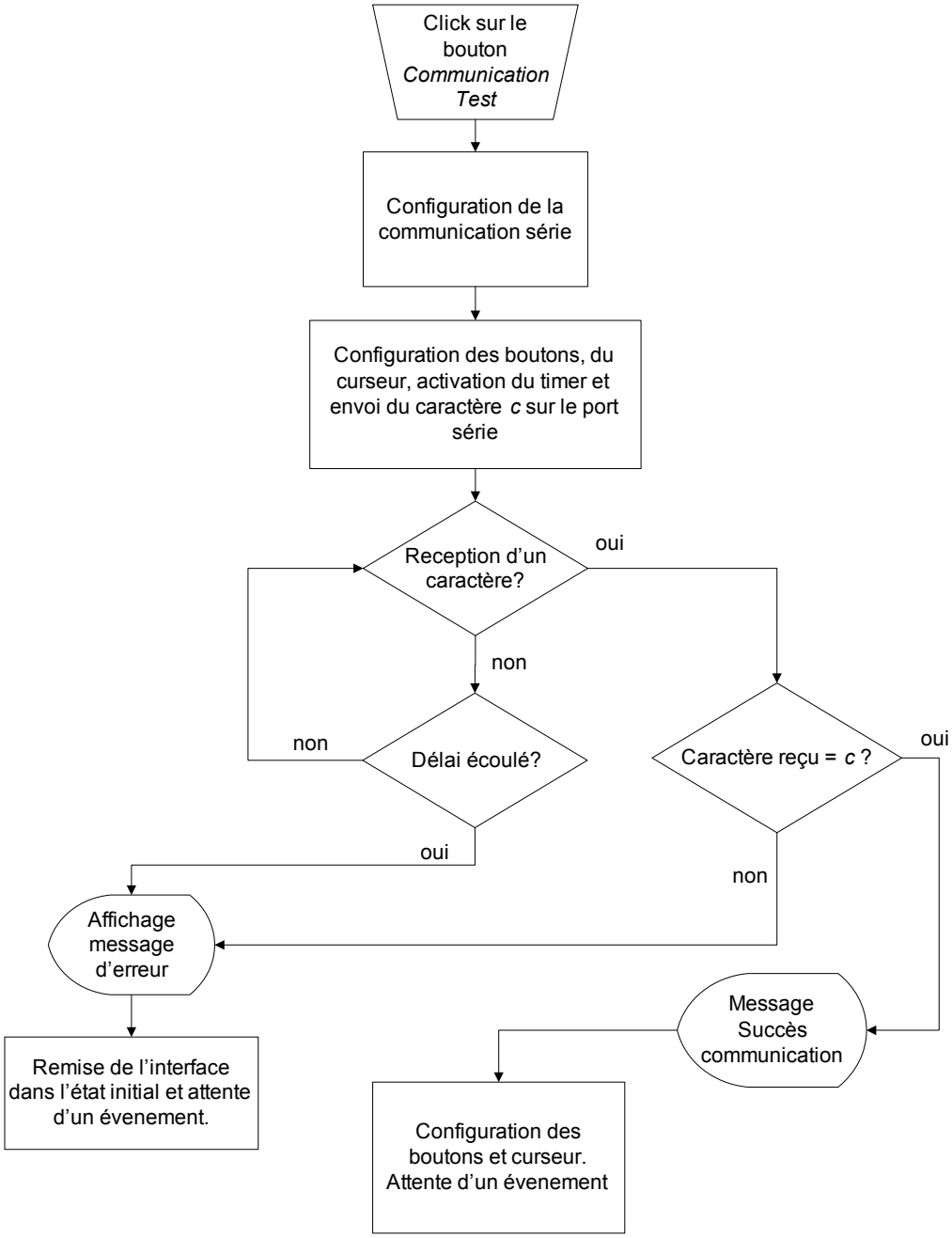
Avant de l'envoyer ou de recevoir une donnée, il faut ouvrir le port de communication. Cette action se fait à l'aide de :

MSComm1.PortOpen = True

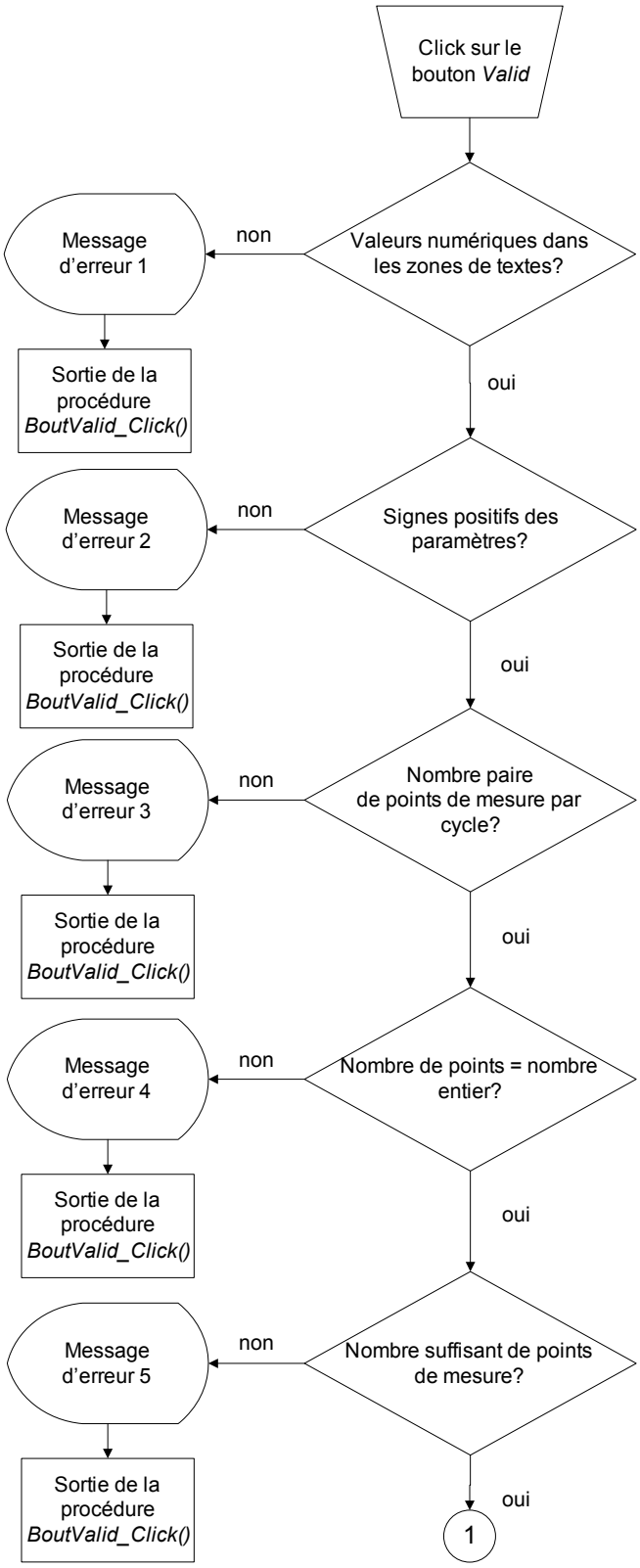
Lorsque l'on veut fermer le port de communication après utilisation, on met cette propriété à *false*.

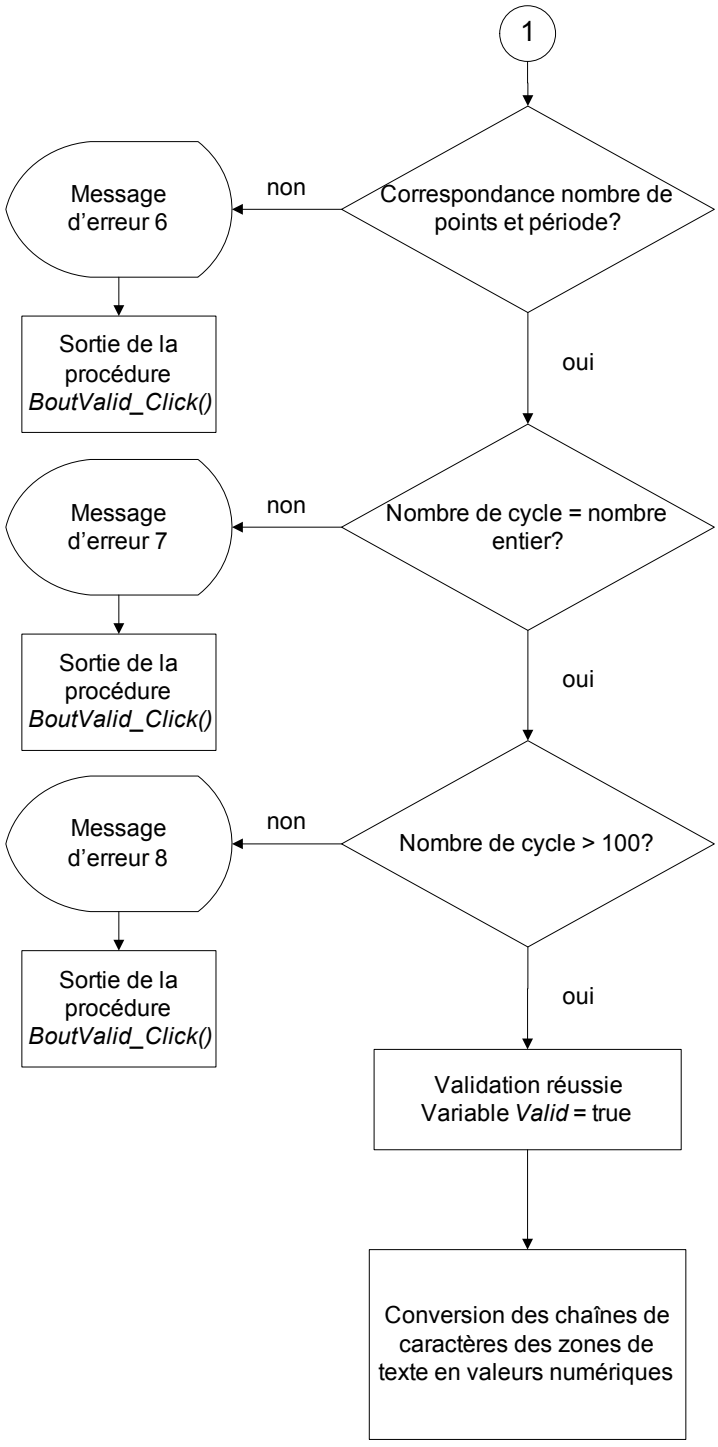
V.3. Programme :

Le programme complet et commenté est disponible en Annexes page 96.
Lors du chargement de l'application, l'utilisateur doit, en premier lieu, tester si tout est prêt pour commencer le test. Il le fera au travers du bouton *Communication Test*.
Lorsque l'utilisateur clique sur ce bouton, Visual Basic configure la communication série suivant le port choisi (1 ou 2). Il envoie ensuite un caractère (*c*) au microcontrôleur.
Si celui-ci est prêt il renverra à l'interface le même caractère. Dans le cas contraire, il ne renverra rien ou un caractère différent de celui envoyé.
Afin de ne pas attendre indéfiniment, on utilise un Timer qui va permettre à Visual Basic d'attendre au maximum cinq secondes pour que le microcontrôleur renvoie quelque chose.La procédure liée au click de ce bouton est intitulée : *BoutComm_Click()*.
On peut illustrer cette procédure à l'aide d'un organigramme.

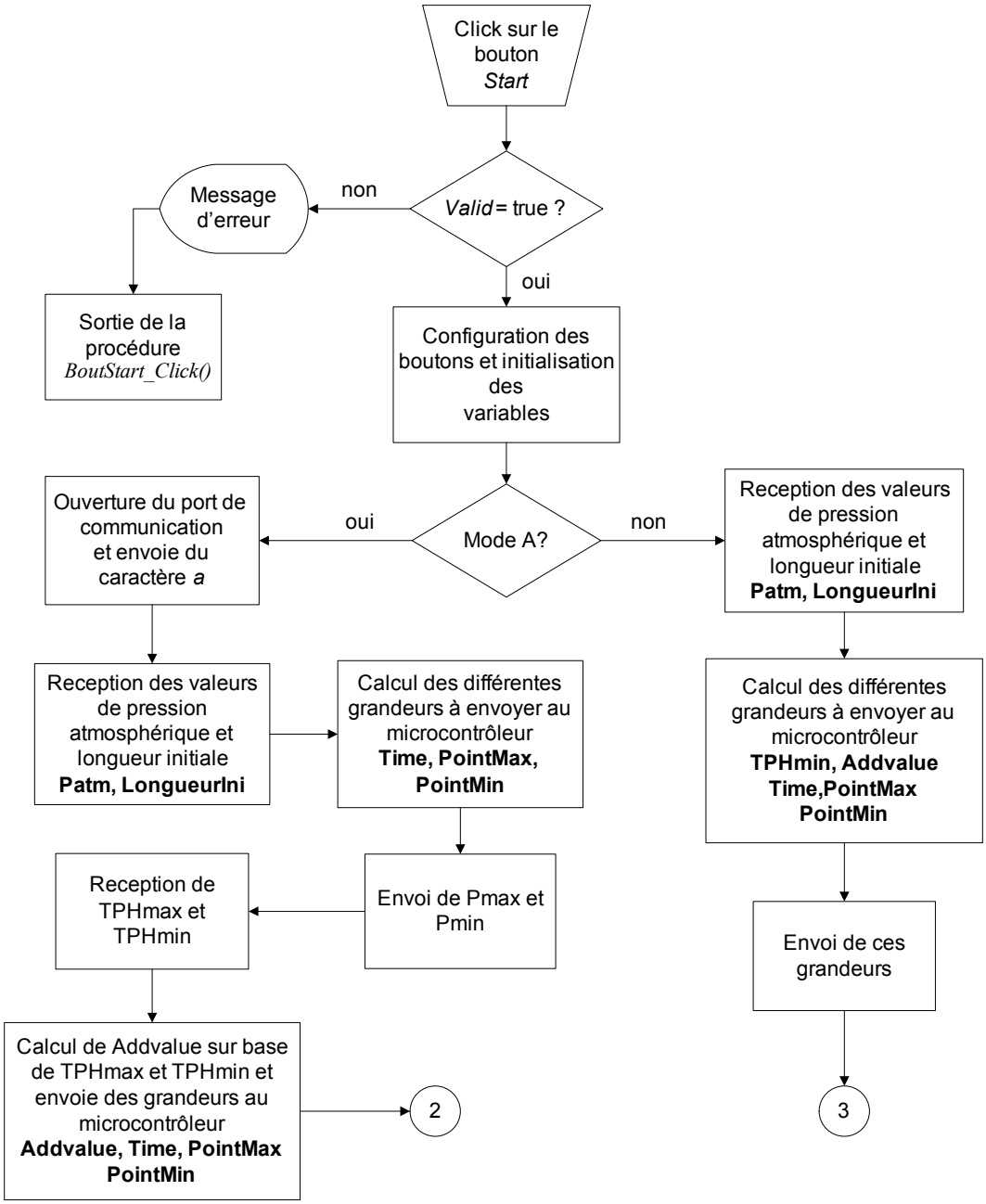


Maintenant qu'on s'est assuré que tout était prêt pour entamer le test, on a accès au bouton *Start*. Avant de lancer le test en cliquant sur ce bouton, on doit valider les paramètres que l'on a inscrits dans les zones de texte. Pour valider ces paramètres, on clique sur le bouton *Valid*.
Voici le déroulement de la procédure associée (*BoutValid_Click()*) :

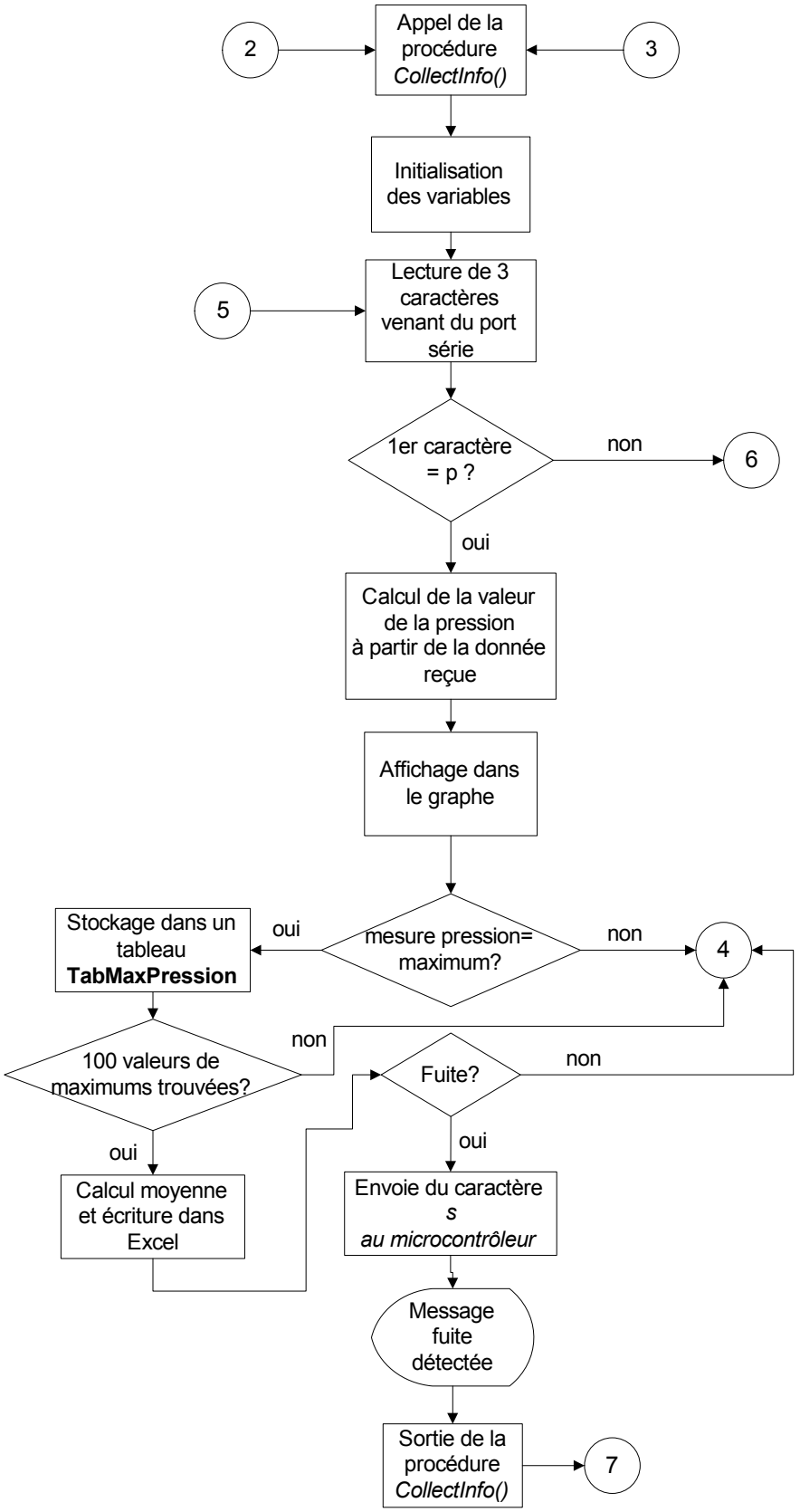


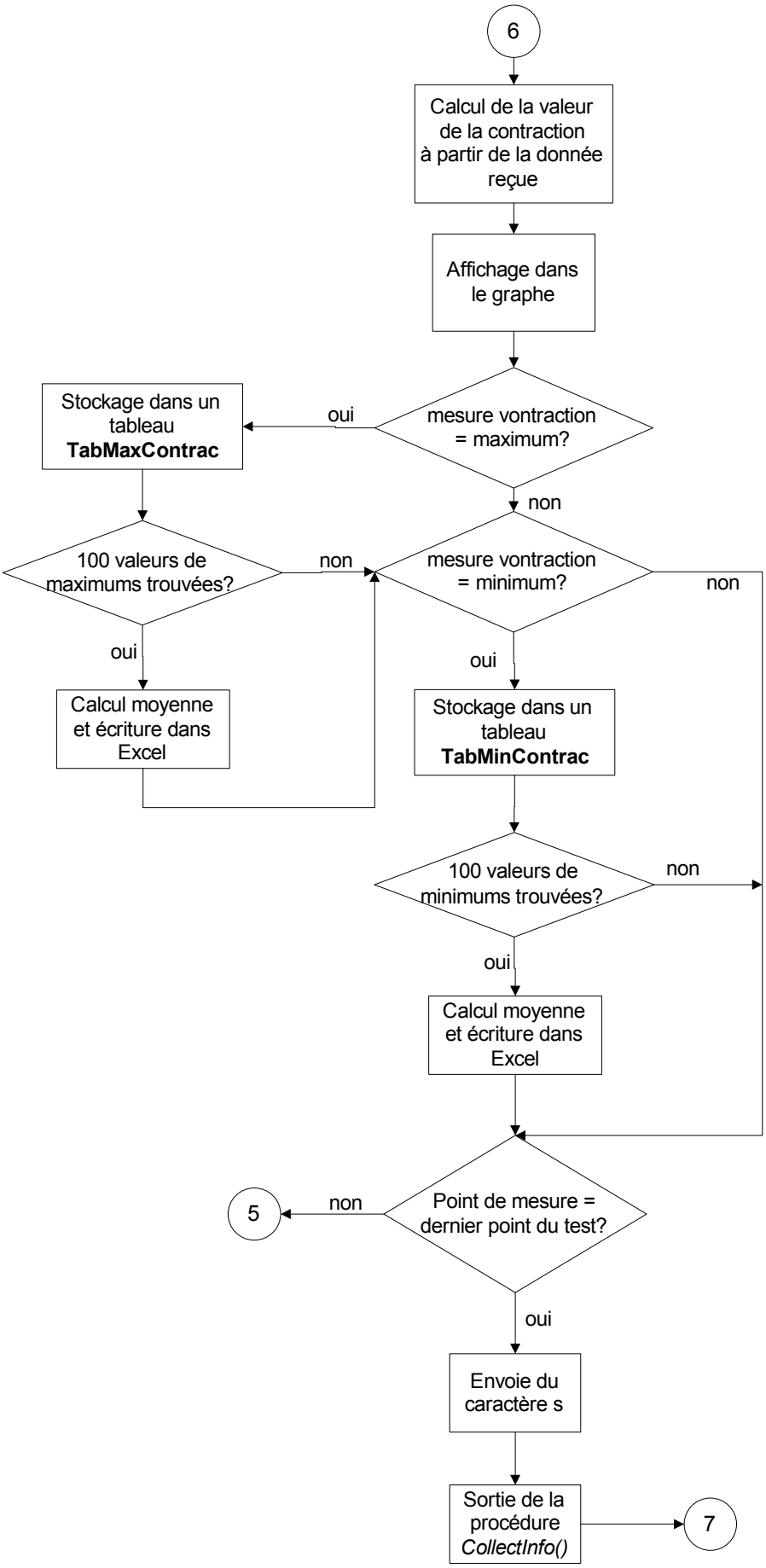

1
Correspondance nombre de points et période?
non
Message d'erreur 6
Sortie de la procédure *BoutValid_Click()*
oui
Nombre de cycle = nombre entier?
non
Message d'erreur 7
Sortie de la procédure *BoutValid_Click()*
oui
Nombre de cycle > 100?
non
Message d'erreur 8
Sortie de la procédure *BoutValid_Click()*
oui
Validation réussie
Variable *Valid* = true
Conversion des chaînes de caractères des zones de texte en valeurs numériques

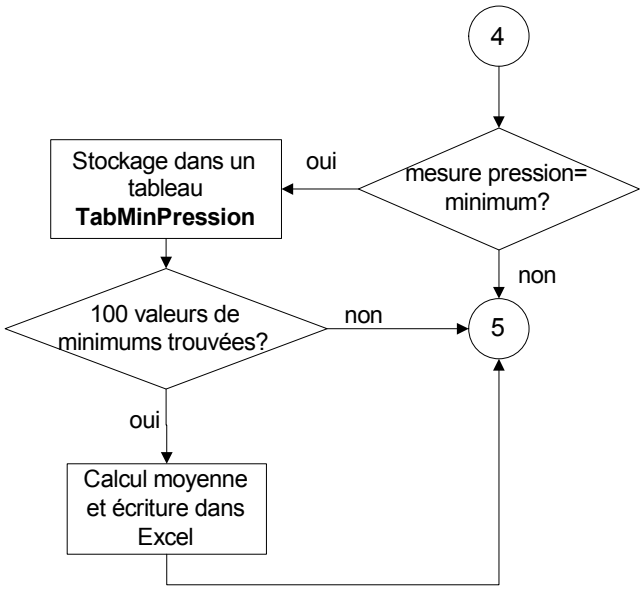
La validation des paramètres est maintenant faite, on peut entamer le test en cliquant sur *Start*.
La procédure associée s'appelle *BoutStart_Click()*.

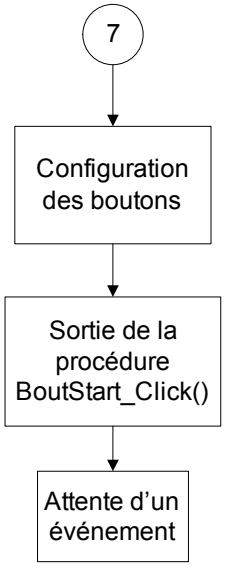


A cet instant, l'interface va recevoir les divers points de mesures du microcontrôleur.
Les mesures de pression sont précédées du caractère *p*. Les mesures de contractions sont précédées du caractère *r*.


2
Appel de la procédure *CollectInfo()*
3
Initialisation des variables
5
Lecture de 3 caractères venant du port série
1er caractère = p ?
non
6
oui
Calcul de la valeur de la pression à partir de la donnée reçue
Affichage dans le graphe
Stockage dans un tableau **TabMaxPression**
oui
mesure pression= maximum?
non
4
non
100 valeurs de maximums trouvées?
Fuite?
non
oui
oui
Calcul moyenne et écriture dans Excel
Envoie du caractère *s* *au microcontrôleur*
Message fuite détectée
Sortie de la procédure *CollectInfo()*
7


6
Calcul de la valeur de la contraction à partir de la donnée reçue
Affichage dans le graphe
mesure vontraction = maximum?
oui
Stockage dans un tableau **TabMaxContrac**
non
100 valeurs de maximums trouvées?
non
oui
Calcul moyenne et écriture dans Excel
mesure vontraction = minimum?
non
oui
Stockage dans un tableau **TabMinContrac**
100 valeurs de minimums trouvées?
non
oui
Calcul moyenne et écriture dans Excel
Point de mesure = dernier point du test?
non
5
oui
Envoie du caractère s
Sortie de la procédure *CollectInfo()*
7





Il faut également remarquer qu'une fois dans la procédure BoutStart_Click(), l'utilisateur peut à tout moment interrompre le test. En effet, le fait de cliquer sur le bouton *Stop* nous fera sortir de cette procédure et donnera l'ordre d'arrêt du test. On se retrouvera alors en situation de départ avec seulement les boutons *Communication Test* et *Close Application* disponibles.

VI. CONCLUSION

En premier lieu, rappelons les objectifs fixés de ce projet : nous étions amenés à réaliser un banc d'essai afin d'étudier le comportement des muscles artificiels pneumatiques dans le temps.
Ce banc d'essai inclut donc :

- une structure mécanique conçue à cet effet.

- les divers capteurs afin de pouvoir effectuer des mesures correctes des grandeurs qui nous intéressent, une vanne pneumatique pour pouvoir régler la pression à l'intérieur de l'actionneur.

- un microcontrôleur qui supervisera la procédure de test de l'actionneur. Il assure donc le contrôle de la vanne de même que la collecte des données. Il est en constante communication avec l'interface.

- l'interface pc : simple d'utilisation, elle va permettre à l'utilisateur d'encoder tous les paramètres relatifs au test. On va pouvoir également observer les différentes grandeurs (pression et contraction) dans un graphique rafraîchi périodiquement.

La réalisation de cette installation mécatronique m'a donc permis de découvrir les différentes facettes du travail de l'ingénieur.

En effet, le dimensionnement et la réalisation de la structure mécanique m'ont permis de mettre en application mes connaissances de la résistance des matériaux et d'en acquérir bien d'autre dans ce domaine.

Ensuite, la métrologie et ses capteurs ont constitué l'étape suivante dans la réalisation du projet. Il a donc fallu faire un choix des différents capteurs pour réaliser des mesures adéquates.

Le traitement des signaux des capteurs a fait partie de mon travail ; l'électronique a alors pris une place importante dans le projet. L'utilisation du microcontrôleur y a largement contribué. La programmation de ce dernier a pris une place importante dans ce travail.

C'est dans un domaine orienté informatique que s'est conclu mon travail. J'ai effectivement réalisé l'interface en Visual Basic afin de pouvoir utiliser aisément le banc d'essai.

La réalisation de toutes ces étapes ne s'est pas effectuée sans acquérir certaines notions et documentations sur les sujets abordés.
Effectivement, pour la programmation du microcontrôleur, il a fallu en premier lieu comprendre et maîtriser le fonctionnement et les particularités de ce dernier. J'ai ensuite dû me familiariser avec le jeu d'instructions compatibles avec le MOTOROLA 68HC11.

Il en était de même avec l'interface Visual Basic. Il a fallu pour ma part découvrir ce langage de programmation et m'adapter à son fonctionnement jusqu'ici inconnu.

Le travail réalisé répond aux spécifications du cahier de charges. Il est alors possible de tester la durée de vie des muscles artificiels pneumatiques de façon automatique, à l'aide de l'interface sur ordinateur.

Dans le futur, on pourrait néanmoins apporter des améliorations au banc d'essai. La première modification serait bien entendu le remplacement de la vanne pneumatique par une autre comprenant une régulation précise de la pression. Dans ce cas, l'interface nous permet d'éviter la phase d'initialisation indispensable actuellement.

Ensuite, au niveau du capteur de pression, l'utilisation d'un amplificateur d'instrumentation intégré dans une puce donnerait de meilleurs résultats.

Une modification au niveau de la mesure de contraction peut également être apportée : comme celle-ci est actuellement de type potentiométrique, il y a donc frottement au niveau des éléments en contact, lors du déplacement du pointeau. L'utilisation répétée de ce type de capteur entraînera une certaine usure. Il serait donc intéressant de favoriser les capteurs sans contact tels que les capteurs inductifs, capacitifs ou encore incrémentaux.
Ce choix ne s'est pas fait car ceux-ci ne répondaient pas à notre critère économique du moment.

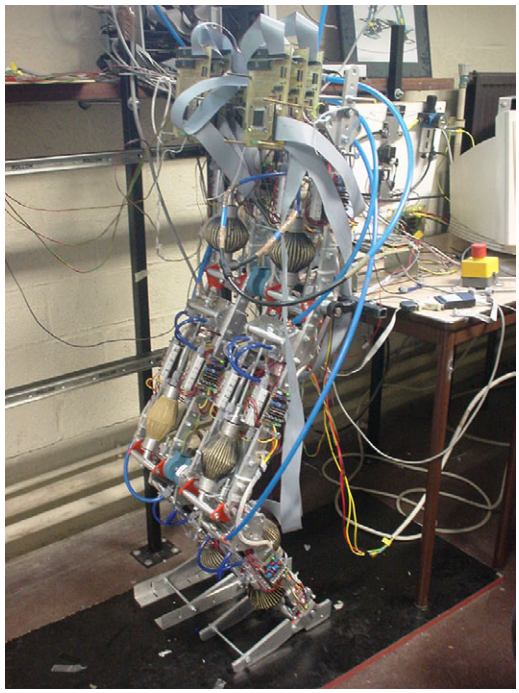
En guise de conclusion, j'aimerais mettre en avant le fait que la réalisation d'un tel projet dans les délais imposés ne peut se faire sans une gestion adéquate du travail. Ceci à donc constitué un autre aspect très instructifs dans l'élaboration du mémoire.

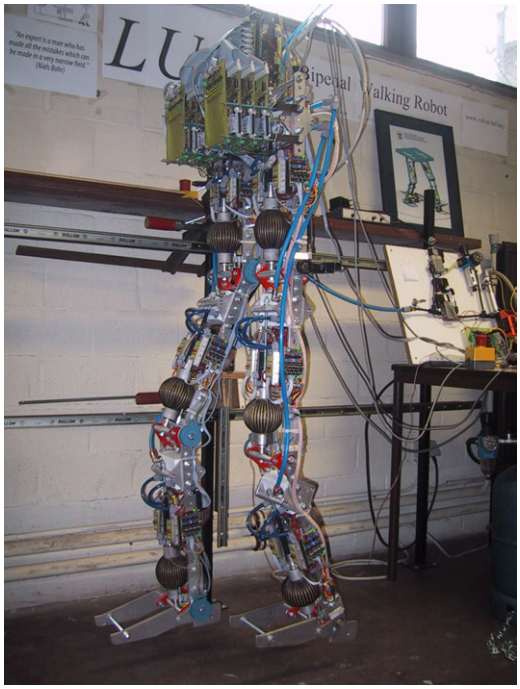
VII. ANNEXES

VII.1. Informations complémentaires :

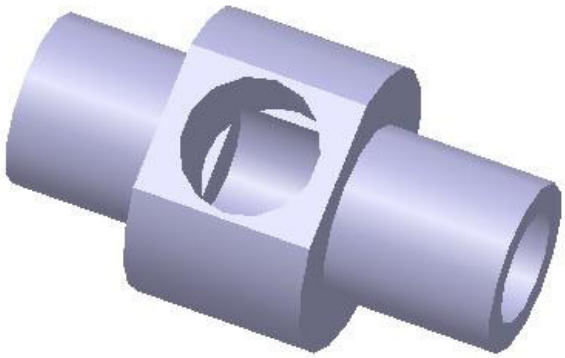
Robot Lucy

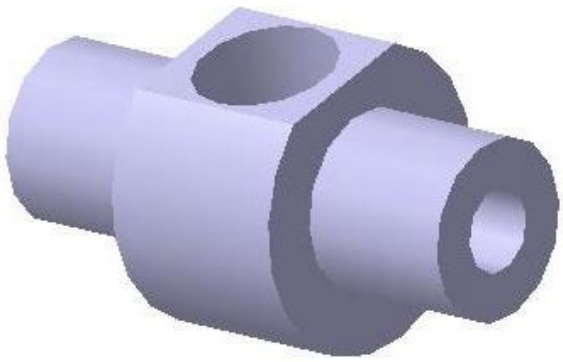
Ce robot bipède est actionné par des muscles artificiels pneumatiques :

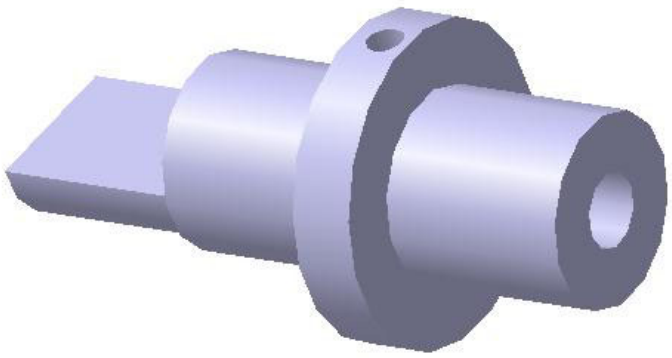


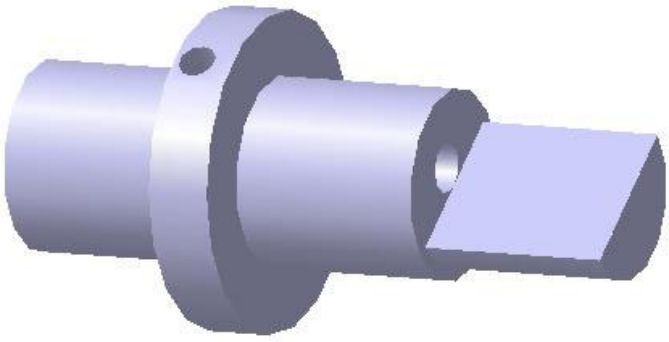


Pièce de fixation du muscle artificiel









Calibration du capteur de pression

Pression atm (bar)	Pression relative (bar)	Pression absolue (bar)	Données du convertisseur A/D
0,983	0	0,983	175
	0,66	1,643	719
	1,48	2,463	1408
	2,2	3,183	2016
	3,1	4,083	2752
	4,2	5,183	3680
	4,48	5,463	3876

Superposition de la droite des moindres carrés et des mesures expérimentales :
Capteur de pression

Calibration du capteur de contraction

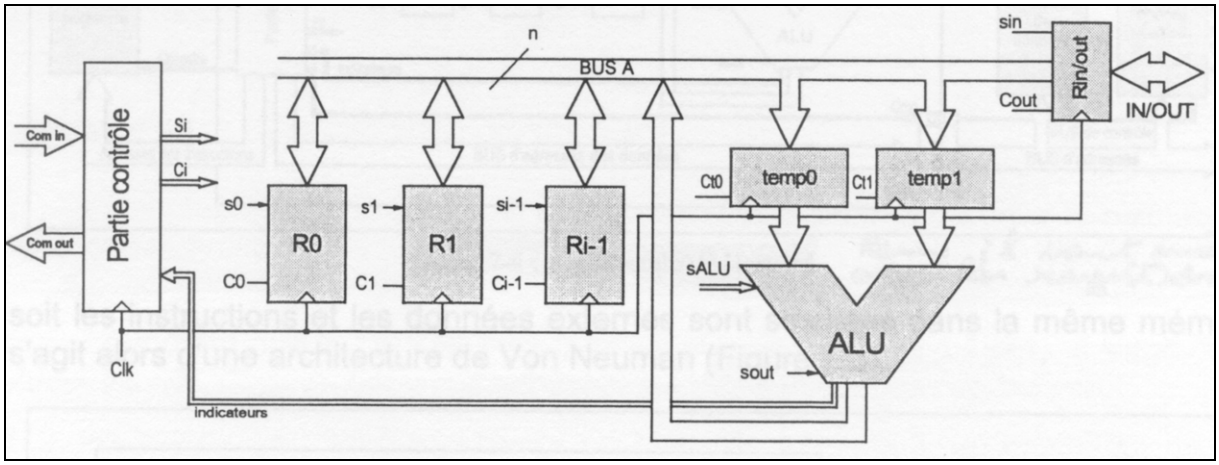
Donnée numérique	position (mm)
0	0
1279	30
2160	50
3072	70

Superposition de la droite des moindres carrés et des mesures expérimentales :
Capteur de contraction

Description générale des microcontrôleurs

Contrairement à la programmation événementielle, les microcontrôleurs fonctionnent de manière séquentielle ; en effet dans une application Windows, par exemple, un événement tel un clic sur un bouton ou un autre type de contrôle va déclencher l'exécution d'une série d'instructions. Or notre microcontrôleur n'attend pas d'événement bien défini pour exécuter les instructions établies dans son programme mais va exécuter celles-ci les unes après les autres.
Un microcontrôleur est donc un système séquentiel qui peut être décomposé en deux parties :
la partie contrôle et la partie chemin de données.
La partie chemin de données est constituée principalement d'une Unité Arithmétique et Logique (ALU) ainsi que d'un ensemble de connections appelée bus sur lesquels vont transiter les données. C'est donc la partie chemin de données qui réalise des traitements sur les données ; ces traitements sont effectués par l'ALU qui est capable de réaliser les opérations arithmétiques et logiques élémentaires tel que :
ET, OU, addition, soustraction, décalages…
La partie contrôle reçoit des renseignements (indicateurs ou flags) de la partie chemin de données ; elle va effectuer des sélections et des commandes sur cette dernière en fonctions des indications reçues et des demandes extérieures.

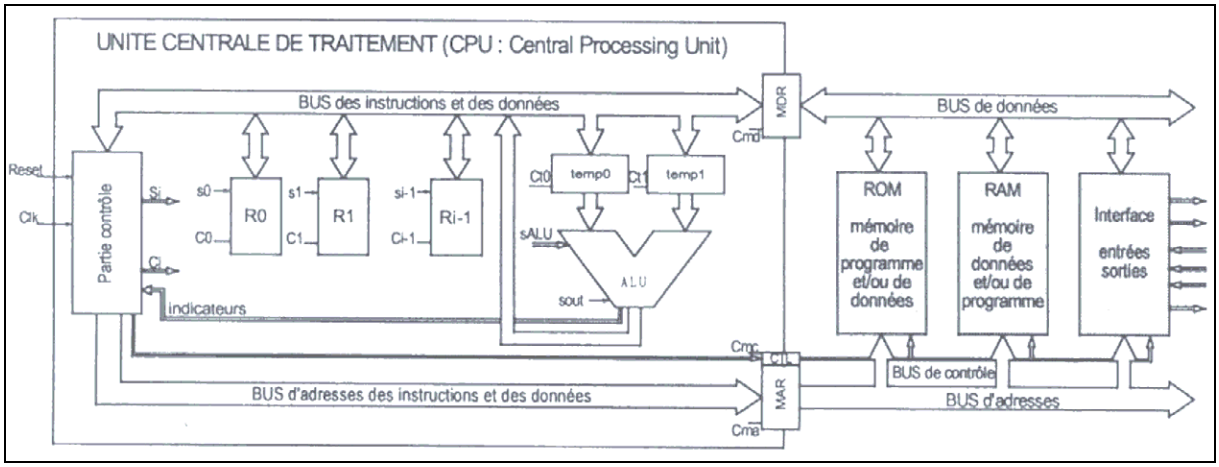
La philosophie de résolution de calculs est la suivante : à partir d'opérations élémentaires simples, on va pouvoir réaliser n'importe quel traitement ou calcul, aussi compliqué soit il.
Par exemple, diviser un nombre par deux revient à exécuter une translation de ce nombre vers la droite : 10 en décimal correspond à 1010 en binaire, pour diviser 10 par deux on décale son expression en binaire vers la droite, ce qui donne donc 101 qui correspond à 5 en décimal.
Un schéma élémentaire représenté à la figure 3.1 illustre la structure interne d'un microprocesseur :



Structure de base

On remarque la présence des signaux c_i et s_i qui correspondent respectivement aux signaux de commande et aux signaux de sélection. Les signaux de sélection permettent de choisir les registres à connecter à l'ALU par l'intermédiaire du bus de données ; ils déterminent également le type d'opération que l'ALU doit réaliser.
Les signaux de commandes désignent quant à eux dans quel registre le résultat de l'opération doit être sauvé.
On observe également dans le schéma de la figure 3.1 la présence de registres temporaires appelés *temp0* et *temp1*. Ceux–ci sont nécessaires afin de réduire le nombre de connections sur lesquelles circulent les données à un bus unique ; ils servent ainsi de registres tampons.
Etant donné que la structure interne a évolué vers un bus de données unique, celui-ci doit permettre la circulation d'information dans les deux sens, il s'agit donc d'un bus bidirectionnel.

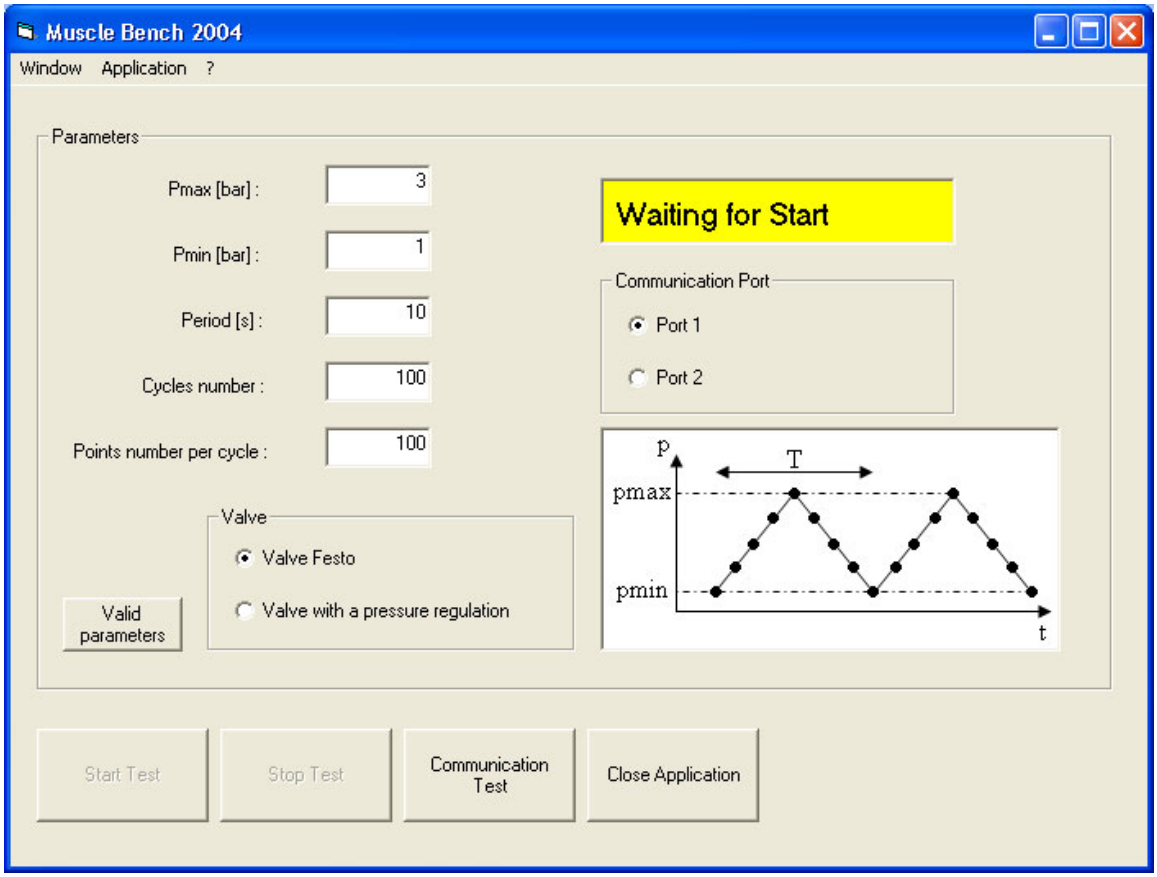
Afin de compléter la description de notre microprocesseur, il faut encore rajouter les mémoires à la structure décrite précédemment. En effet, le programme, constitué d'une suite d'instructions, doit être stocké dans une mémoire et être accessible à tout moment ; on décrit alors l'architecture de notre microprocesseur. Il existe deux types d'architecture : l'architecture dite de HARVARD et l'architecture de VON NEUMAN.
Lorsque les instructions du programme sont stockées dans une mémoire et les données externes dans une autre, on parle d'architecture de HARVARD.
Si par contre, les instructions et les données externes sont stockées dans la même mémoire, il s'agit alors d'une architecture de VON NEUMAN.
Le MOTOROLA 68HC11 appartient au second type d'architecture. On peut alors illustrer l'ensemble de la structure interne du microprocesseur avec un schéma plus complet :



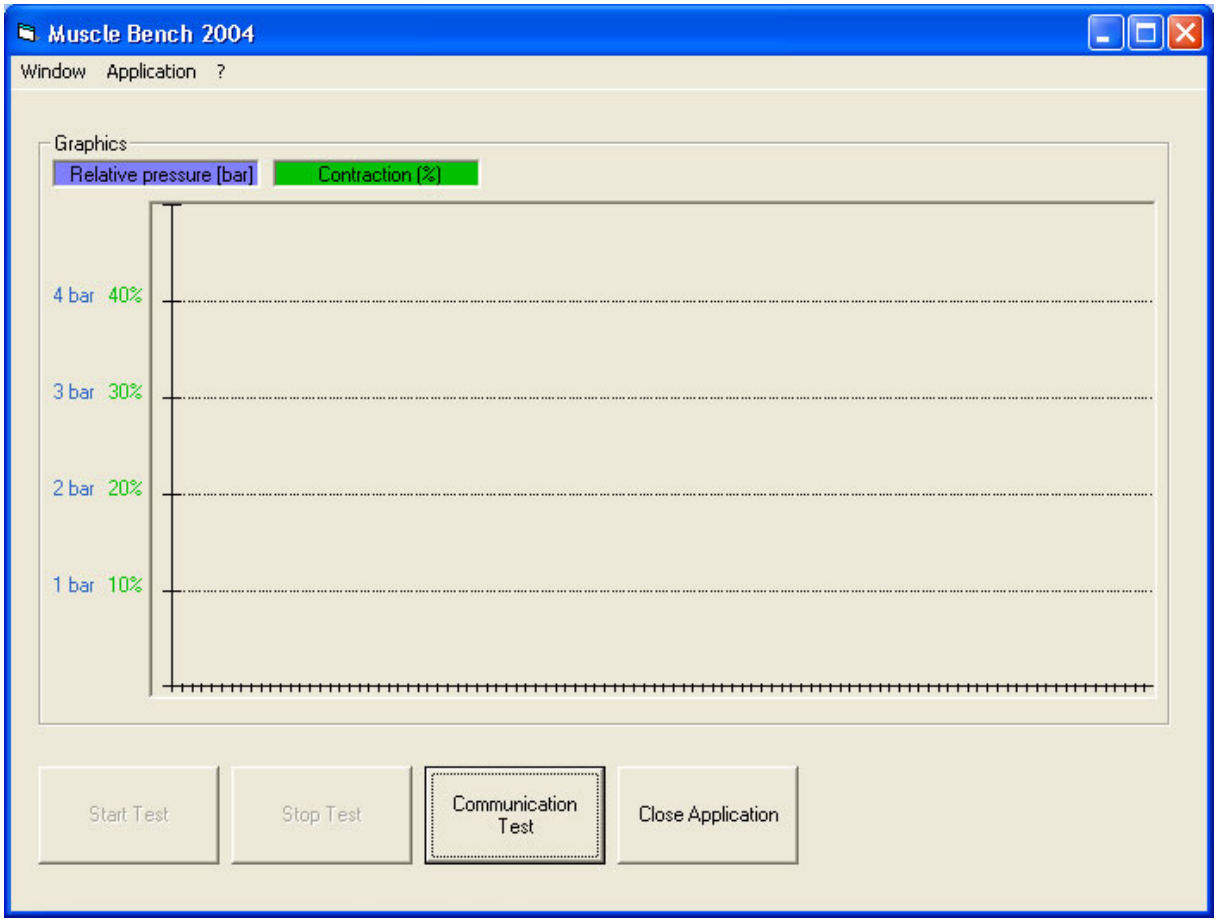
Structure interne

On remarque alors la présence d'un bus supplémentaire appelé bus d'adresse qui spécifie où se trouve l'instruction ou la donnée en mémoire, nécessaire pour le déroulement du programme.

Interface Visual Basic



Fenêtre Parameters



Fenêtre Graphics

VII.2. Programmes :

Programme Assembleur

Interface Visual Basic

VII.3. Données techniques :

- *Capteur CPC100*
- *Vanne MPP-3-1/8*
- *IC LM324*
- *A/D LTC1286*
- *Capteur PN2024*
- *Capteur PLS100*

VIII. Bibliographie

- [GOULET] : *Résistance des matériaux* – J. Goulet et J-P. Boutin – édition Dunod
- [DROUET] : *Calcul des constructions métalliques* – édition Drouet
- [XIONG] : *Formulaire de résistance des matériaux* – Y. Xiong – édition Eyrolles
- *Amplificateur d'instrumentation* – O. Français
- [CHOU] : *Measurement and modelling of McKibben Pneumatic Artificial Muscles* - C. Chou et B. Hannaford
- [DAERDEN] : *Pleated Pneumatic Artificial Muscles : Compliant Robotic Actuators* - F. Daerden – VUB
- [NOEL] : *Electronique industrielle* – J. Noël – ECAM
- *Microcontroller Technology The 68HC11* – P. Spasov – edition Prentrice Hall
- *Initiation au microcontrôleur 68HC11* – M. Bairanzadé – ETSF
- *Acquisition de données, du capteur à l'ordinateur* – G. Asch & co – édition Dunod
- *Le programmeur : Visual Basic 6* – G. Perry – édition CampusPress