SERIE N°5 (TS)
TP1

PRESENTATION GENERALE

Partie abordée ou système support:

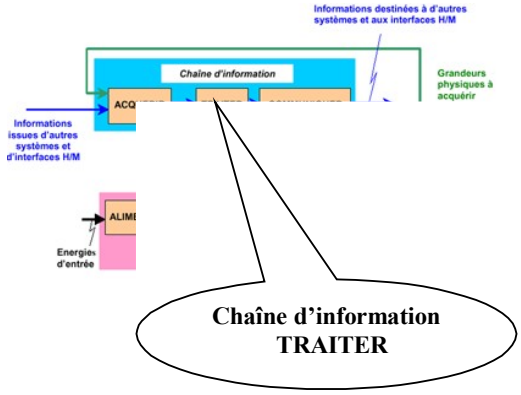
STRUCTURE LOGICIELLE DE LA FONCTION "TRAITER"

Intitulé du TP

COMMANDER LA POMPE DE L'ECHANTILLONNEUR

Durée du TP **2h**

Axe(s) mis en œuvre par le TP :



DONNEES PEDAGOGIQUES

Centre d'intérêt :	**CI.10 : TRAITEMENT DE L'INFORMATION (thème I11).** **CI.11 : SYSTEMES LOGIQUES ET NUMERIQUES (thème I10).**
Compétences attendues :	**Configurer le produit et le faire fonctionner.** **Générer automatiquement le programme et l'implanter dans le système cible.** **Modifier la spécification comportementale à l'aide d'un éditeur (atelier logiciel, interface de développement rapide).** **Tester le fonctionnement.**
Savoirs et Savoir-faire associés :	**B4 : Traiter l'information.** **C24 : Comportement des systèmes numériques.**
Pré-requis :	**L'élève a déjà programmé l'échantillonneur et connaît les différentes phases de fonctionnement de la pompe.** **L'élève a des notions d'algorithme et de langage C.**

DONNEES TECHNIQUES

Environnement matériel et logiciel nécessaire :	**1 alimentation stabilisée12V** **4 maquettes servant à simuler le fonctionnement de l'échantillonneur d'eau (la carte «distributeur » dont le moteur et le codeur incrémental 24 encoches sera pour ce TP censé être le moteur de la pompe et sera connecté à la carte présence d'eau; la carte présence d'eau; la carte « afficheur LCD»; la carte « kit ATMEL»).**
Documents à utiliser :	**Documentation technique de l'échantillonneur (sur le poste).** **Plan de câblage des cartes.** **Annexes , contenus des bibliothèques INOUT , delay et LCD_V3.** **Algorigramme, algorithme, langage C.**

TRAVAIL DEMANDE

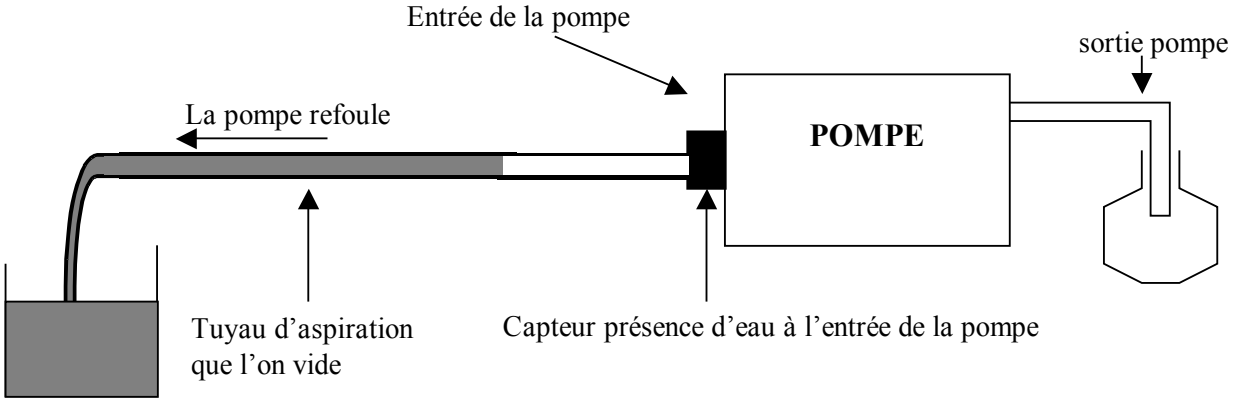
Partie I : préparation à la maison (page 2 , 3, 4) ½ h de travail

1. Rappel sur le fonctionnement de la pompe (3 tâches à effectuer)

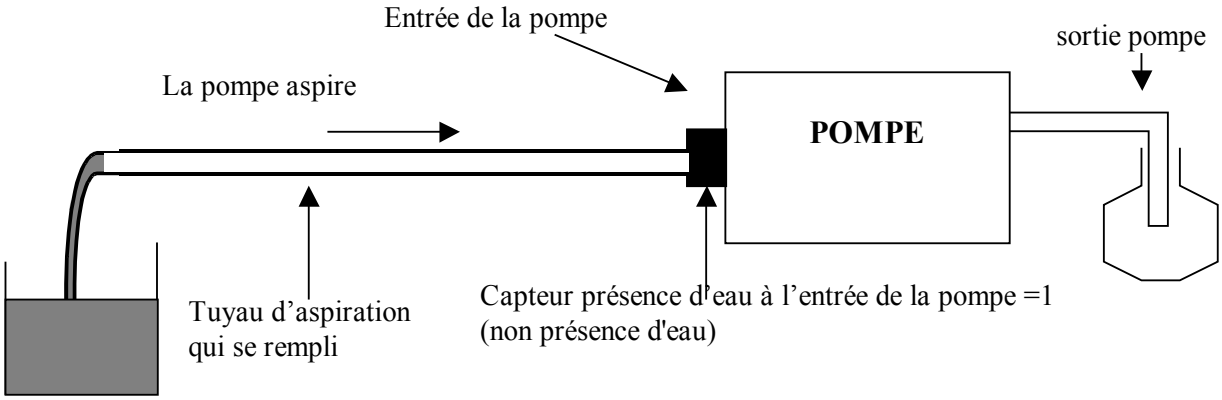
La pompe utilisée dans le système calypso est réversible (elle peut aspirer ou refouler), le volume d'eau pompée est fonction du nombre de tours du rotor de la pompe. Sur ce rotor est fixé un codeur incrémental percé de 24 encoches dont une grande (top au tour) qui nous permettra de contrôler le volume d'eau pompée.

Tâche n°1 « vidanger le tuyau d'aspiration » :

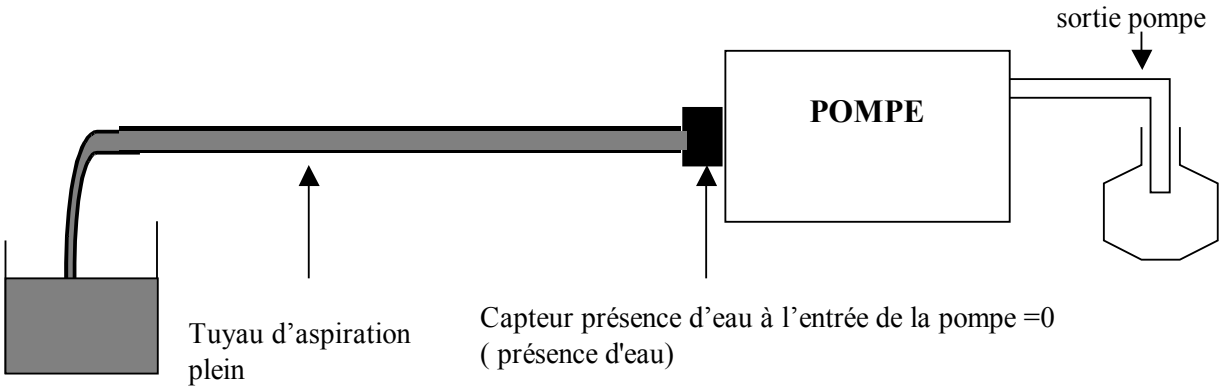
Avant le prélèvement d'un échantillon, il faut vider le tuyau d'aspiration de façon à ne pas mélanger l'eau de deux prélèvements différents.



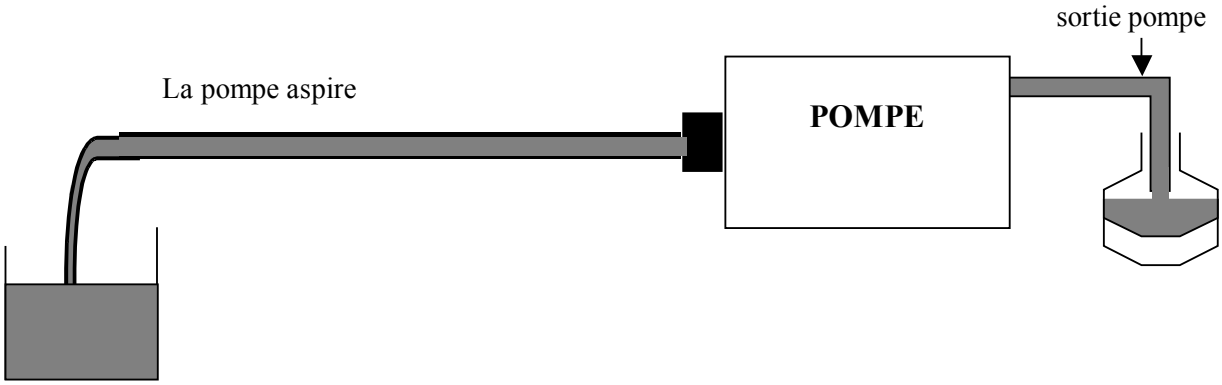
Tâche n°2 « aspirer jusqu'à présence d'eau à l'entrée de la pompe » :



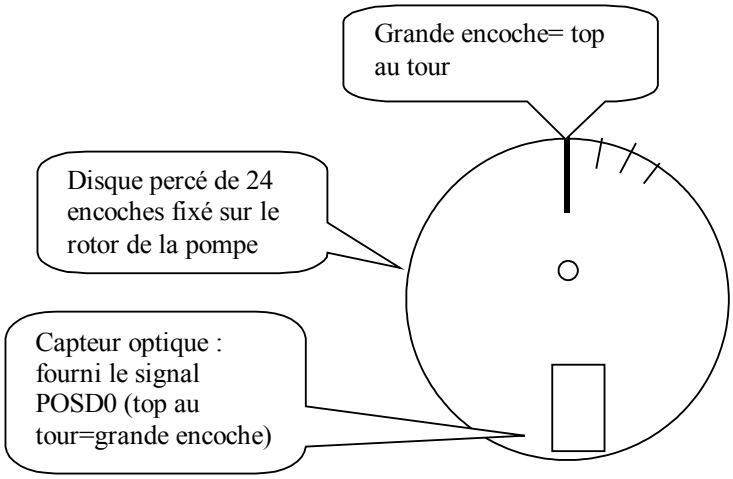
Fin de la tâche 2



Tâche n°3 « aspirer jusqu'à ce que flacon soit rempli avec le volume adéquat (fonction du nombre de tours du rotor de la pompe, programmé avec le microswitch 1 » :



ORGANIGRAMME GLOBAL DU FONCTIONNEMENT DE LA POMPE

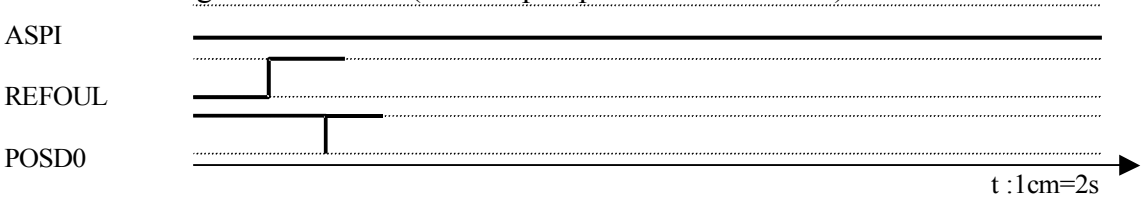


ETUDE DE LA TÂCHE «vidanger le tuyau d'aspiration »

Lors de cette tâche la pompe doit refouler, le rotor de la pompe doit au moins faire 3 tours. Pour cela nous allons utiliser un des signaux du codeur incrémental fixé en sortie du motoréducteur : POSD0.

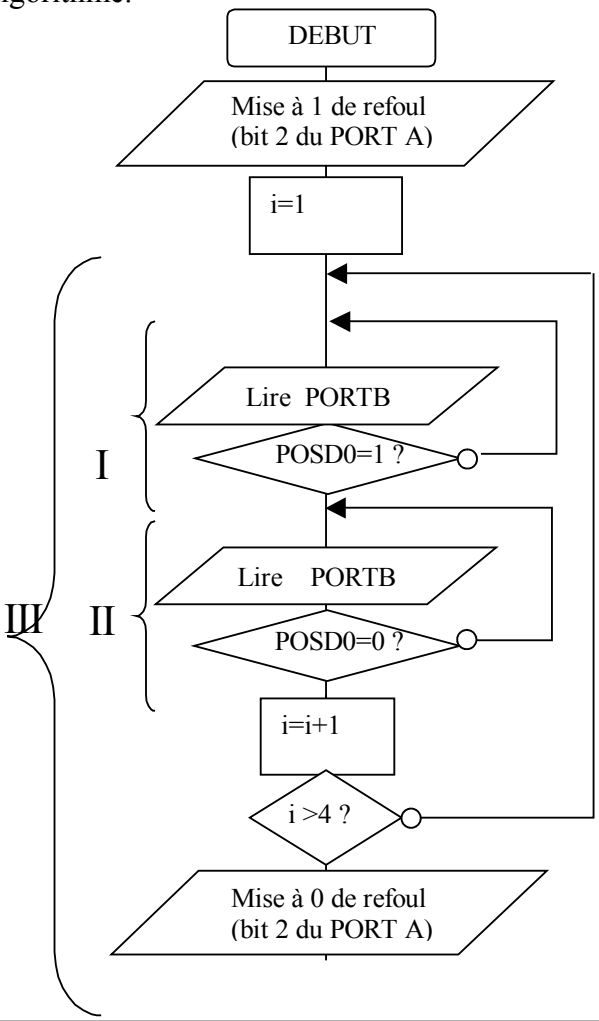
POSD0 est le signal « top au tour », il est associé à la grande encoche du disque. POSD0 passe au niveau logique bas lorsque la grande encoche passe dans la fourche du capteur optique. Si nous comptons 4 fronts descendant de POSD0, nous avons la garantie que le rotor de la pompe a bien fait au moins 3 tours.

Compléter le chronogramme suivant (le rotor pompe fait un tour en 5s)



Algorigramme de la tâche « refouler »

Dans cet algorigramme nous repérons deux structures « faire ……jusqu'à » (I et II) qui permettent de détecter un front descendant de POSD0 et une structure « pour i de 1 à 4 faire » (III), **compléter** l'algorithme.



Début

Mise à 1 de REFOUL

Pour i de 1 à 4 **faire**

faire

POSD0=bit 2 du PORT B ;

Jusqu'à POSD0=1

Faire

……

……

Fin pour

……

Comparer la fonction en langage C correspondant à l'algo, et la **compléter**.

```
void vidanger()
{       unsigned char i, POSD0;
        mise_a_1_PORTA_bit(2);  //mise à 1 de refoul
        message2();
        for (i=1;i<=4;i=i+1)
        {       do
                        {POSD0=test_PORTB_bit(2);}
                while (……);
                delay_ms(100);
                do
                        {POSD0=test_PORTB_bit(2);}
                while (……);
                delay_ms(100);
        }
    mise_a_0_PORTA_bit(2);  //mise à 0 de refoul
}
```

- Composant logiciels réutilisables de la bibliothèque INOUT :

```
mise_a_1_PORTA_bit(2);
mise_a_0_PORTA_bit(2);

test_PORTB_bit(2);
```

- Composant logiciels réutilisables de la bibliothèque LCD_V3 :

```
delay_ms(100); tempo 100ms
```

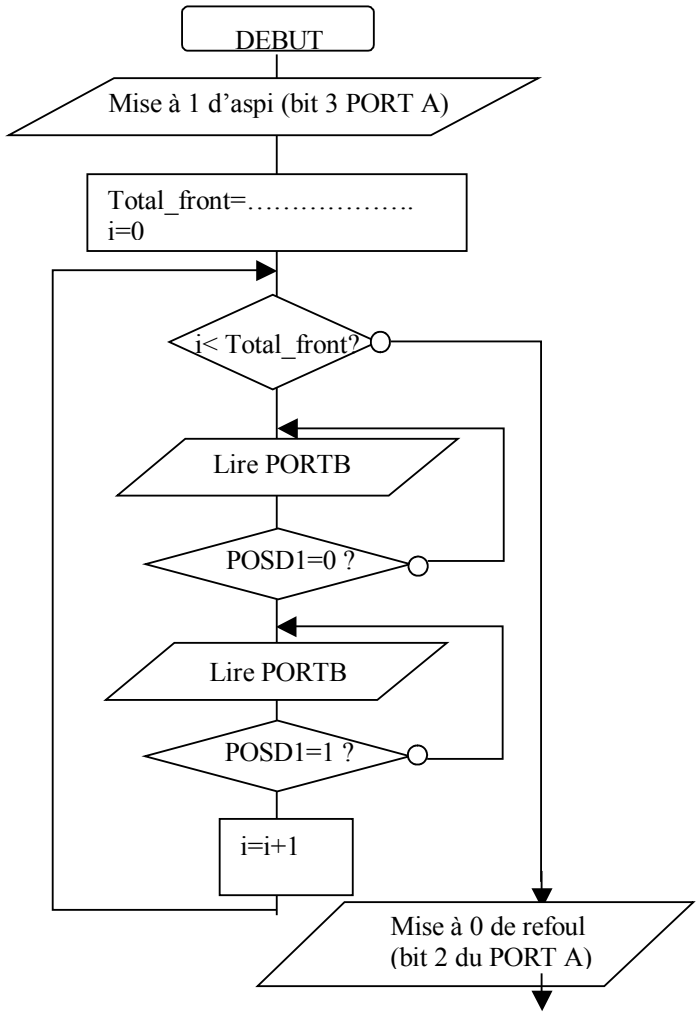
Mise au point de la fonction « tache3_remplissage »

Au départ de la tâche 3 l'eau est présente à l'entrée de la pompe. Le volume pompé est fonction du nombre de tours que va effectuer le rotor de la pompe, et ce nombre est fixé par le microswitch1 relié au PORT D. (variable nbre_de_tours).
Lors de cette tâche la pompe doit aspirer, le rotor de la pompe doit faire exactement « **nbre_de_tours** » tours. Pour cela nous allons utiliser un des signaux du codeur incrémental : POSD1. **POSD1** est le signal associé à la grande encoche du disque et aux 23 petites. POSD1 passe au niveau logique bas lorsque la grande ou une des 23 petites encoches passent dans la fourche du capteur optique.
Si nous comptons 24 fronts descendants de POSD1, nous avons la garantie que le rotor de la pompe a fait 1 tour. (+/- la résolution du codeur : 360/24=15° pour le 1°tour).

Combien de fronts descendants de POSD1 faudra t'il compter pour garantir que le bon volume a été pompé. ?
total_fronts= …… ?

Nous allons donc utiliser une structure algorithmique de type : TANT QUE……FAIRE

Algorigramme de la tâche 3 : **compléter** l'algorithme en vous inspirant de la tâche 1.



Début

Mise à 1 d'ASPI
total_front=
i=

TANT QUE (……..

……

……

……

……

……

……

……

……

……

……

Programme C correspondant : le **compléter** et le **faire vérifier**.

```
void tache3_remplissage(unsigned char nbre_de_tours)
{       unsigned char i,total_fronts, POSD1 ;
        message_tache3();

        mise_a_1_PORTA_bit(……); //mise à 1 d'aspi

        total_fronts=…….*……;

        i=0;
        while (……)
        {          // test front descendant de POSD1
                do
                        {POSD1=test_PORTB_bit(……);}
                 while (……);
                   delay_ms(100);
                do
                        {POSD1=test_PORTB_bit(……);}
                while (……);
                delay_ms(100);
                i=i+1;
        }


        mise_a_0_PORTA_bit(3); //mise à 0 d'aspi
}
```

Mise en œuvre du micro-contrôleur ATMEL et de ses périphériques.

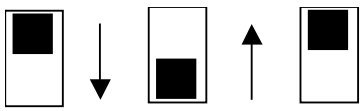
Matériel : Un PC équipé d'un logiciel de programmation micro-contrôleur en langage C.
Un kit ATMEL connecté au PC.
Un afficheur LCD.
Un carte comportant un moto-réducteur associé à un capteur de position.
La carte présence d'eau associée à une sonde.

A partir du schéma proposé en annexe 1, **câbler** les ports du kit Atmel à ses périphériques. **Alimenter** la carte distributeur et la carte présence d'eau (Vbat=12V).

La carte LCD affiche les messages d'information du fonctionnement.

N'ayant pas de poussoir à disposition c'est l'interrupteur SW0 de la carte distributeur qui commandera le départ cycle (dcy), pour un fonctionnement normal, cet interrupteur doit être remis en position repos.

! Cycle d'appui sur dcy = SW0 ⇒



- **Lancer** le logiciel de programmation ATMEL, et **charger** le projet POMPE1.
- **Compléter** la fonction tâche3.
- **Compiler** et **télécharger** le programme dans la mémoire flash du microcontrôleur.
- **Effectuer** les tests en programmant différents volumes d'eau à pomper avec le microswitch (3, 4, et 5 tours).
- **Appeler votre professeur** et **effectuer** un test devant lui.

Modification :

Nous remarquons que dans la tâche 1 nous détectons un front descendant de POSD0 (bit 2 de PORT B) et que dans la tâche 3 nous détectons un front descendant de POSD1 (bit 3 de PORT B).

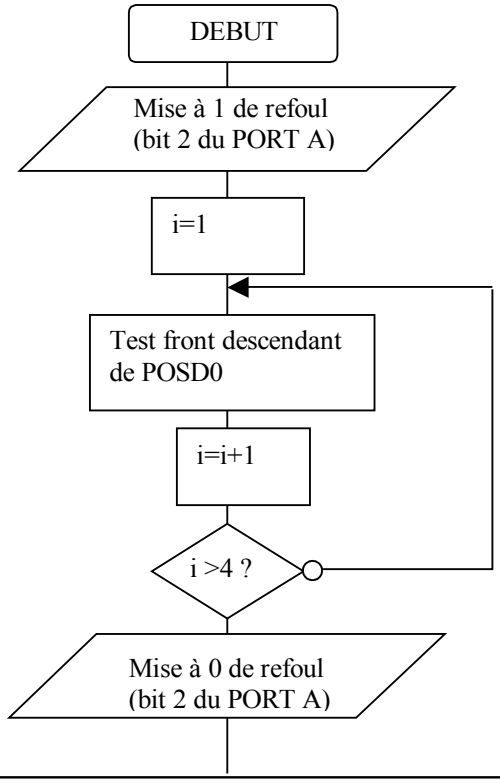
Il serait donc judicieux de créer un composant logiciel réutilisable permettant de détecter un front descendant sur n'importe quel bit du PORT B.

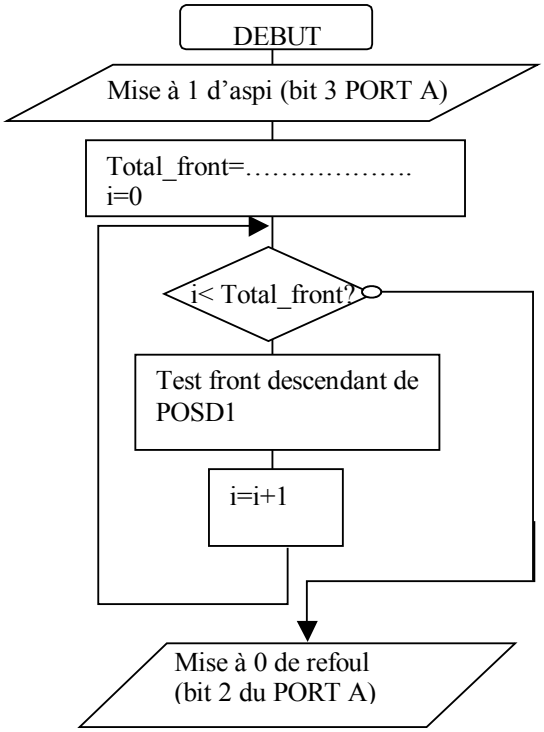
Front_desc_PORTB_bit(unsigned char i) ; avec i de 0 à 7, cette fonction ne se terminerait que lorsqu'elle aurait détecté un front descendant sur le bit i du PORT B.

```
Void        Front_desc_PORTB_bit(unsigned char i)
{
            do { }
            while (test_PORTB_bit(i)==0);
            delay_ms(100);
            do { }
            while ( test_PORTB_bit(i)==1)
            delay_ms(100);

}
```

Cette fonction se trouve dans la bibliothèque INOUT2 (avec celles de la fonction INOUT), elle nous permettra de nous simplifier la vie (c'est le but des composants logiciels réutilisables)

Nouveaux algorigrammes et nouvelles fonctions tâche 1 et tâche 3





FONCTIONS

```
void tache1_vidanger()
{       unsigned char i;
            message_tache1();
            mise_a_1_PORTA_bit(2);  //mise à 1 de refoul


            // la pompe fait trois tours pour vidanger
            for (i=1;i<=4;i=i+1)
            {
               Front_desc_PORTB_bit(2);
            }

        mise_a_0_PORTA_bit(2);  //mise à 0 de refoul
}
```

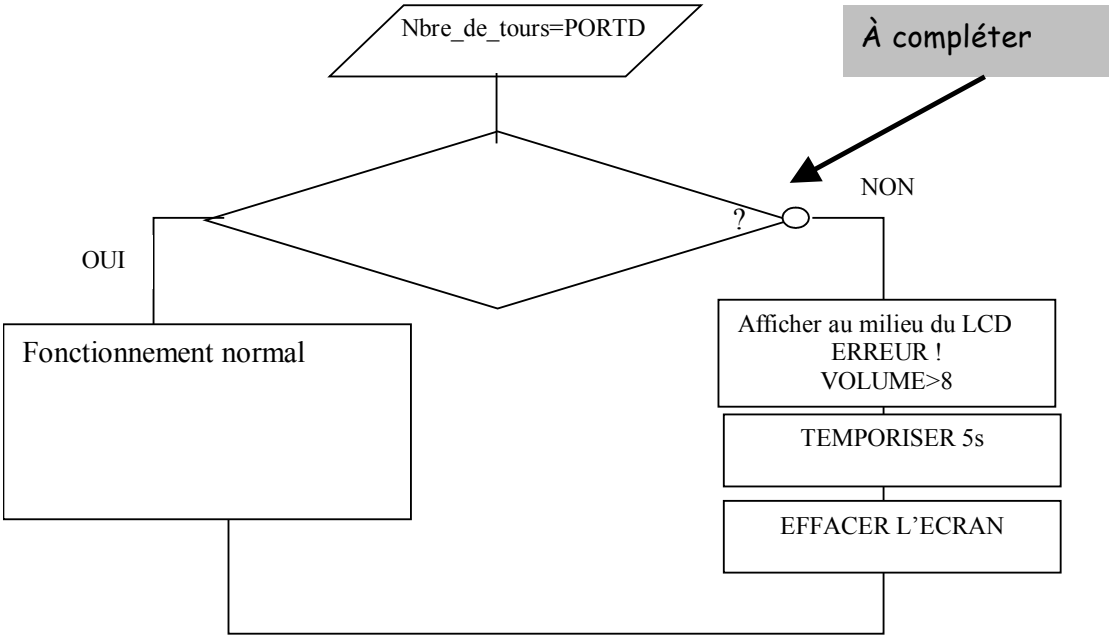
```
void tache3_remplissage(unsigned char nbre_de_tours)
{           unsigned char i,total_fronts;
            message_tache3();

            mise_a_1_PORTA_bit(3); //mise à 1 d'aspi

            total_fronts=24*nbre_de_tours;

            i=0;
            while (i<total_fronts)
            {
               Front_desc_PORTB_bit(3);
                    i=i+1;
            }


            mise_a_0_PORTA_bit(3); //mise à 0 d'aspi
}
```

Modification du programme

- **Modifier** les tâches 1 et 2 ; (attention rajouter au début du programme #include<INOUT2> et **supprimer** #include<INOUT> , **effectuer** le test pour voir si tout se passe bien.

MODIFICATION :

Les flacons ne peuvent contenir une quantité infinie d'eau, il faut donc limiter le volume d'eau à pomper.
Nous désirons modifier le programme de façon à ce qu'il affiche un message d'erreur lorsque le nombre programmé avec le microswitch1 dépasse 8.
Pour cela il nous faut modifier l'algorigramme , l'algorithme et le programme en C.



- **A quelle** structure algorithmique cette modification correspond t'elle (voir annexe) ……?

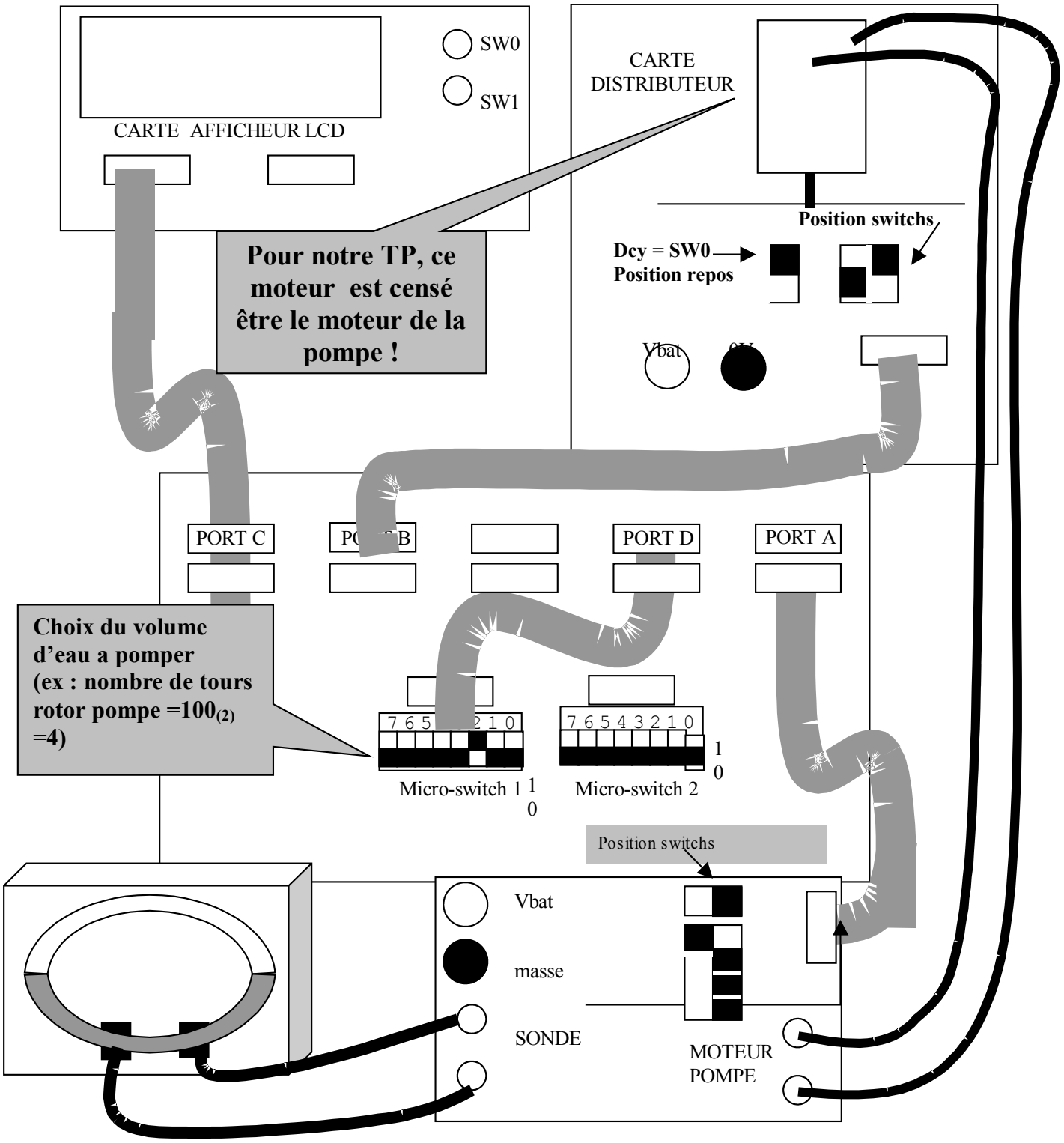
ALGORITHME	LANGAGE C

……
(……)
……
afficher « ERREUR VOLUME>7»
temporisation de 5s
……
« fonctionnement normal »
fin_si
	…… (……..)
	{
	……//eff
	acer l'écran
	……// placer curseur
	en 5;0
	……// afficher
	« ERREUR »
	……// placer curseur
	en 3 ;1
	……//afficher
	VOLUME>7
	……//t
	emporiser 5s
	……//eff
	acer l'écran
	}
	else
	{ fonctionnement_normal()
	}

- Après vérification, **charger** le projet POMPE3, **compléter** le programme et **tester** (pour des raisons de taille limite de programme, l'affichage des tâches est supprimé). Si vous avez le temps **ajouter** une boucle POUR… FAIRE… de façon à faire clignoter le message 5 fois 1 seconde.

ANNEXE 1

Câblage du kit ATMEL et de ses périphériques (le moteur est connecté à la carte relais).



SONDE

Maquette présence d'eau et relais

CONFIGURATION DES PORTS

PORTA= a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0

Présence eau
ASPI
REFOUL

PORTB= b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0

POSD1
POSD0
dcy = SW0

PORTC= c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0

commande de l'afficheur LCD

PORTD= d_7 d_6 d_5 d_4 d_3 d_2 d_1 d_0

relié à un micro-switch

Composants logiciels réutilisables de la bibliothèque LCD_V3

- **void lcd_clear()**

Efface l'écran et est prêt à écrire en position ligne 0 et colonne 0.

- **void lcd_gotoxy(unsigned char x, unsigned char y)**

positionne le caractère à afficher à une position colonne X et ligne Y. Par défaut X = Y = 0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0																
1																

- **void lcd_putchar(char c)**

Affiche le caractère correspondant au code ASCII c à la position courante (prédéfini).

- **lcd_putsf**(«chaîne de caractères »);

  affiche une chaîne de caractères à la position courante du curseur

Composants logiciels réutilisables de la bibliothèque INOUT

- bibliothèque INOUT : contient des fonctions, permettant de tester individuellement les bits des ports configurés en entrée, ainsi que de forcer les bits des ports configurés en sortie.

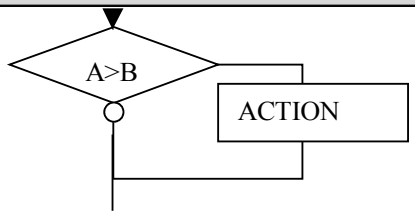
- unsigned char test_PORTA_bit(unsigned char i) : teste le bit i du port A et renvoie 0 ou1
- unsigned char test_PORTB_bit(unsigned char i) : teste le bit i du port B et renvoie 0 ou1
- void mise_a_1_PORTA_bit(unsigned char i) : force à 1 le bit i du port A (sans changer les autres)
- void mise_a_0_PORTA_bit(unsigned char i) : force à O le bit i du port A (sans changer les autres)
  :

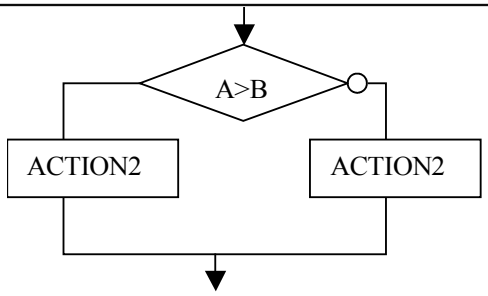
Composants logiciels réutilisables de la bibliothèque delay

- bibliothèque delay : contient deux fonctions de temporisation

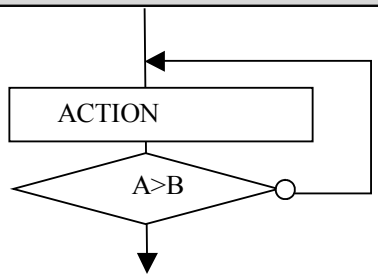
  delay_ms(unsigned int i) :temporisation de i ms (0<i<65535)
  delay_us(unsigned int i) :temporisation de i µs (0<i<65535)
- delay_ms(100) : temporisation de 100ms
- delay_us(1000) : temporisation de 1000µs

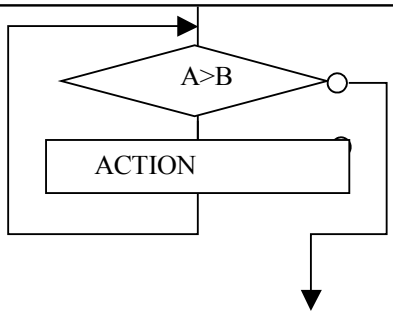
ALGORIGRAMME	ALGORITHME	LANGAGE C
STRUCTURES CONDITIONNELLES		

	**Si** A>B
	Alors
	Action
	Fin si
		If (A>B)
		{
		Action ;
		}

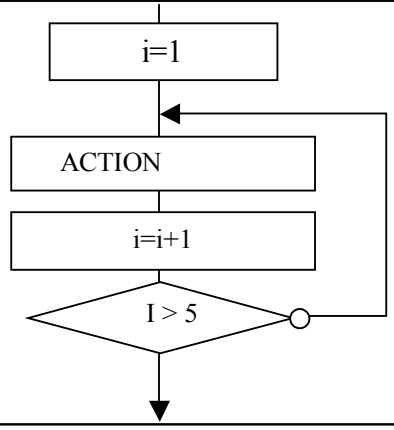
	**Si** A>B
	Alors Action 1
	Sinon Action 2
	Fin si
		If (A>B)
		{
		Action 1 ;
		}
		else
		{
		Action 2 ;
		}

STRUCTURE RÉPÉTITIVE		

	**Répéter**
	Action
	Jusqu'à A>B
		Répéter….. jusqu'à n'existe pas en langage C, on utilise **répéter…. tant que**, avec condition inversée.
		do {
		Action ;
		}
		while (A<=B)

	**Tant que** A>B
	Action
	Fin Tant que
		while (A>B)
		{
		Action;
		}

STRUCTURE BOUCLE		

	**Pour** i de 0 à 5
	Action
	Fin pour
		For (i=1 ; i<=5 ; i=i+1)
		{
		Action;
		}