TD1 d'informatique industrielle
Geii1 – II2
Algorithme et langage C

Ces exercices ont pour objectif de vous familiariser avec la syntaxe du langage C et les techniques de transposition d'un algorithme.
Pour mener à bien ces activités, il vous faut consulter :

- **le cours (partie introduction au C).**

Exercice 1 [while et for]

Donnez le code de calcul de la factorielle de n (notée n!).
On rappelle le principe de la factorielle (dans le cas de factorielle 5) :

5!=5*4*3*2*1=120

Le résultat est obtenu par la répétition de plusieurs opérations.
Ici, on va, par exemple, utiliser une variable temporaire nommée `tempo` et effectuer une multiplication/accumulation par tous les chiffres jusqu'à 5 :

Au départ : tempo←1

Calcul à faire	Valeur de tempo
tempo ← tempo *2	2
tempo ← tempo *3	6
tempo ← tempo *4	24
tempo ← tempo *5	120

Voici l'algorithme pour une boucle FOR :

```
tempo← 1
Pour i variant de  1 à n faire
  tempo←tempo*i
Fin Tant que
Factorielle←tempo
```

Utilisez une boucle **while** puis une boucle **for** Pour décrire ce traitement.
Le résultat final doit être placé dans une variable nommée `factorielle`.

Exercice 2 [if]

Résoudre l'équation $ax^2 + bx + c = 0$.
Vous utiliserez l'instruction if.
Rappel : il faut calculer le déterminant $\Delta=(b^2-4a.c)$
Si Δ est positif ou nul, alors 2 racines réelles $(-b\pm\sqrt{\Delta})/2.a$
Si Δ est négatif, alors pas de solution réelle (ici, on ne calculera pas les racines complexes).
Remarque: dans ce traitement, on suppose que les variables a,b,c existent déjà et contiennent un nombre. Dans votre traitement, vous déclarerez les variables a,b,c et vous leur affecterez une valeur arbitraire.

Exercice 3 [for et if]

Construire un programme qui permet de saisir une suite de caractères, puis de compter et d'afficher le nombre de lettres 'e' et d'espaces.
Remarque: le nombre maximum de caractères dans la chaîne est de 20.
Le nombre de lettres 'e' sera placé dans la variable `compteurLettreE`.
Le nombre d'espaces sera placé dans la variable `compteurEsp`.

Exercice 4 [utilisation d'une librairie]

On suppose que :

- on dispose d'une fonction clavier() qui retourne 0 si aucune touche n'est active ou sinon la valeur de la touche.
- on dispose d'une fonction delay_ms(valeur_ms) qui permet de créer une temporisation en milli-secondes

Ecrivez un programme qui affiche le carré des entiers 1, 2, 3, .. toutes les 500 ms tant qu'aucun caractère n'a été frappé au clavier.

Exercice 5 [pointeurs]

padr1 et padr2 sont des pointeurs pointant sur des réels. Le contenu de adr1 vaut -45,78; le contenu de adr2 vaut 678,89. Ecrivez un programme qui affiche les valeurs de padr1, padr2 et de leur contenu.

Remarque: dans ce traitement, il faut déclarer les deux variables et leur affecter une valeur; il faut aussi déclarer les pointeurs.

Exercice 6 [pointeurs et tableaux]

Un programme contient la déclaration suivante:

Unsigned int tab[10] = {4,12,53,19,11,60,24,12,89,19};

Complétez ce programme de façon à pouvoir afficher les adresses des éléments du tableau.

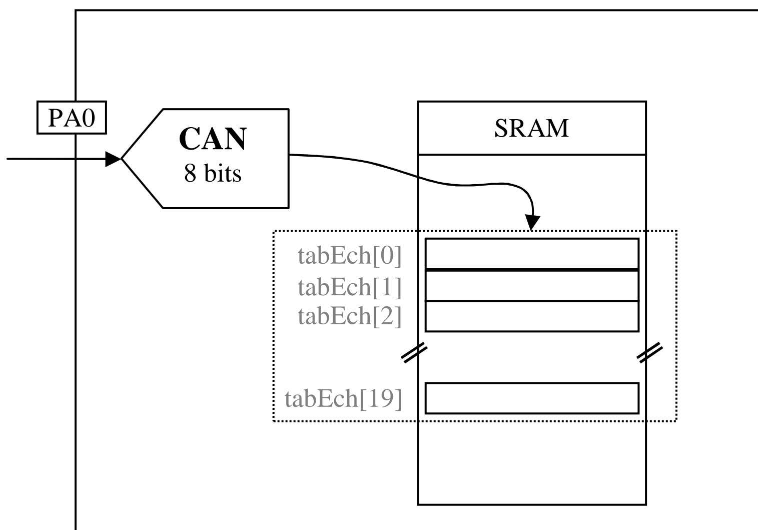
Exercice 7 [fonctions]

Dans le cadre d'un système d'acquisition de données, 20 échantillons de tension ont été numérisés à l'aide d'un CAN 8 bits (1) ; les échantillons sont stockés en mémoire SRAM dans le tableau nommé *tabEch*.

Voici un morceau de code :

```
printf("valeur moyenne des échantillons %3.2f",valMoy());
printf("valeur max des échantillons %d",valMax());
printf("valeur min des échantillons %d",valMin());
```

Donnez l'algorithme et le code C qui permet d'obtenir et d'afficher la valeur moyenne (fonction `valMoy`()), la valeur max (fonction `valMax`()) et la valeur min (`valMin`()) calculée à partir de ces 20 échantillons.



(1) un CAN 8 bits est un système qui transforme une valeur analogique en une grandeur numérique équivalente. Dans notre cas, la grandeur numérique est codée sur 8 bits donc entre 0 et 255

IUT Angoulême	**TD2 d'informatique industrielle** **Geii1 – II2** **Contrôle des PORTS et temporisation**

Exercice 1 : contrôle des ENTREES/SORTIES

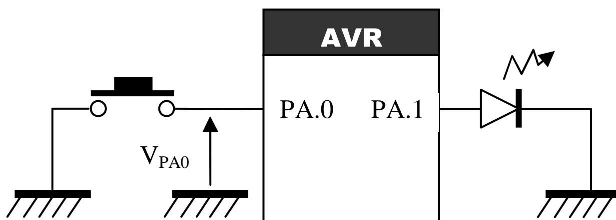
Rappelez le nom des deux registres qui permettent:
- de programmer la direction d'un PORT,
- de placer une valeur sur un PORT.

Donnez les lignes de code en C qui permettent d'obtenir la configuration suivante :

AVR	
PORTA	PORTA en sortie avec toutes les lignes à l'état logique '1'
PORTB	PORTB en entrée
PORTC3-0	PORTC (poids faibles) en sortie (avec toutes les lignes initialement à l'état logique '0')
PORTC7-4	PORTC (poids forts) en entrée

Exercice 2 : étude des résistances de pull-up

Voici un schéma simple dans lequel un bouton poussoir est câblé sur la broche 0 du PORTA.
Une led est reliée à la broche 1 du PORTA.
Le bouton poussoir (BP) est relâché au repos.
OBJECTIF : la led doit s'allumer quand le bouton est appuyé.



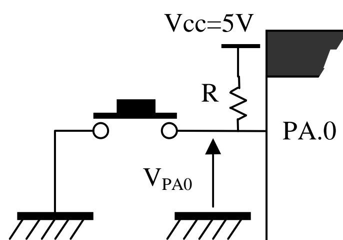
Q1 : indiquez dans quel sens (ENTREE/SORTIE) doit être configurée la broche PA0. Donnez la valeur de DDRA.0
Même travail avec PA1.

Q2 : quand le bouton poussoir est enclenché, quelle est la tension que l'on peut mesurer sur l'entrée PA0 → VA0=
Quel est l'état logique équivalent que l'on peut lire sur PA0?

Q3 : quel état logique faut-il alors envoyer sur PA1?

Q4 : quand le bouton poussoir est relâché, quelle est la tension mesurée sur PA0 :
- 0V
- 5V
- tension 'flottante' impossible à déterminer (on dit haute impédance)

On est alors obligé de câbler une résistance R entre le Vcc est l'entrée PA0.



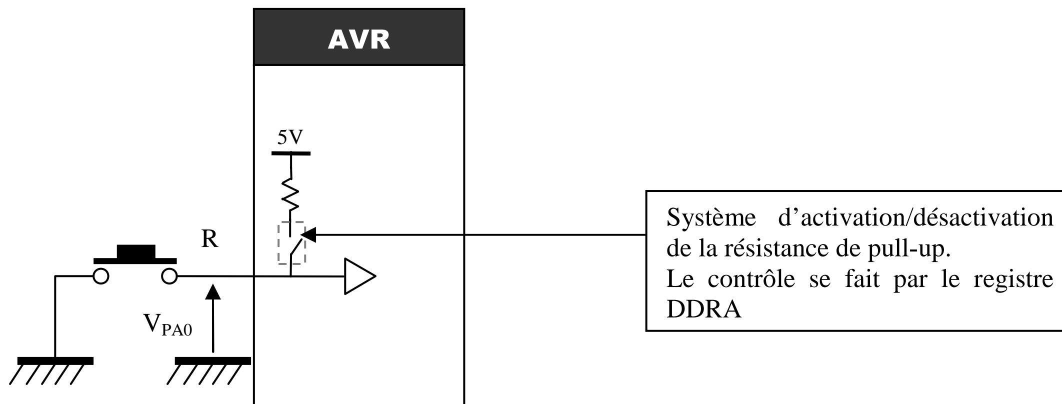
Q5 : tracer le chemin du courant qui passe dans la résistance quand le bouton poussoir est enfoncé.
Remarque : les broches du microcontrôleur, quand elles sont configurées en entrée, présentent une impédance très élevée (idéalement infinie).
On veut limiter le courant dans R à 100µA.
Donnez la valeur de R.

Q6 : tracez le chemin du courant qui circule dans la résistance quand le bouton poussoir est relâché.
Quelle est la tension aux bornes de R VR=
Quelle est la tension VPA0=
Quel est le niveau logique équivalent que l'on peut lire sur PA0?

Q7 : la résistance R est appelée résistance de pull-up (notée R_{PU}). Justifiez ce terme et proposez une traduction en français.

Q8 : dans un système à microcontrôleur embarqué, on cherche à réduire le nombre de composants autour du microcontrôleur.
C'est la raison pour laquelle, les microcontrôleurs possèdent déjà une résistance de pull-up interne pour chaque broche des PORTS.

Mais celle-ci est configurable logiciellement au sens ou l'on peut choisir de d'activer ou pas la résistance.



Indiquez la valeur qu'il faut donner à DDRA.0

Q9 : en consultant les spécifications électriques de l'Atmega16, indiquez quelle est la plage de valeurs possibles pour Rpu.

DC Characteristics

T_A = -40°C to 85°C, V_{CC} = 2.7V to 5.5V (Unless Otherwise Noted)

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{IL}	Input Low Voltage	Except XTAL1 pin	-0.5		0.2 V_{CC}(1)	V
V_{IL1}	Input Low Voltage	XTAL1 pin, External Clock Selected	-0.5		0.1 V_{CC}(1)	V
V_{IH}	Input High Voltage	Except XTAL1 and $\overline{\text{RESET}}$ pins	0.6 V_{CC}(2)		V_{CC} +0.5	V
V_{IH1}	Input High Voltage	XTAL1 pin, External Clock Selected	0.7 V_{CC}(2)		V_{CC} +0.5	V
V_{IH2}	Input High Voltage	$\overline{\text{RESET}}$ pin	0.9 V_{CC}(2)		V_{CC} +0.5	V
V_{OL}	Output Low Voltage(3) (Ports A,B,C,D)	I_{OL} = 20 mA, V_{CC} = 5V I_{OL} = 10 mA, V_{CC} = 3V			0.7 0.5	V V
V_{OH}	Output High Voltage(4) (Ports A,B,C,D)	I_{OH} = -20 mA, V_{CC} = 5V I_{OH} = -10 mA, V_{CC} = 3V	4.2 2.2			V V
I_{IL}	Input Leakage Current I/O Pin	Vcc = 5.5V, pin low (absolute value)			1	µA
I_{IH}	Input Leakage Current I/O Pin	Vcc = 5.5V, pin high (absolute value)			1	µA
R_{RST}	Reset Pull-up Resistor		30		60	kΩ
R_{pu}	I/O Pin Pull-up Resistor		20		50	kΩ
V_{ACIO}	Analog Comparator Input Offset Voltage	V_{CC} = 5V V_{in} = V_{CC}/2			40	mV
I_{ACLK}	Analog Comparator Input Leakage Current	V_{CC} = 5V V_{in} = V_{CC}/2	-50		50	nA
t_{ACID}	Analog Comparator Propagation Delay	V_{CC} = 2.7V V_{CC} = 4.0V		750 500		ns

Quel est alors le courant consommé quand l'interrupteur est activé?

Q10 : on adoptera toujours le PRINCIPE suivant :
Quand une broche n'est pas utilisée, alors on la configure

en entrée	C'est une précaution de sécurité. En effet, si la broche est placée en sortie à l'état '1' et qu'un périphérique soit câblé (par inadvertance) sur cette broche, alors le périphérique serait actionné ce qui pourrait être préjudiciable pour la sécurité du système
avec résistance de pull-up	pour fixer un état logique haut

Complétez alors :

DDRA

7	6	5	4	3	2	1	0

PORTA

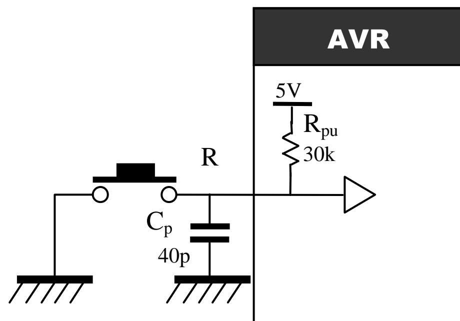
7	6	5	4	3	2	1	0

Remarque : les résistances de pull-up n'ont aucun intérêt quand une broche est configurée en sortie

Q11 : SYNTHESE complétez le code C la partie initialisations pour obtenir le fonctionnement attendu

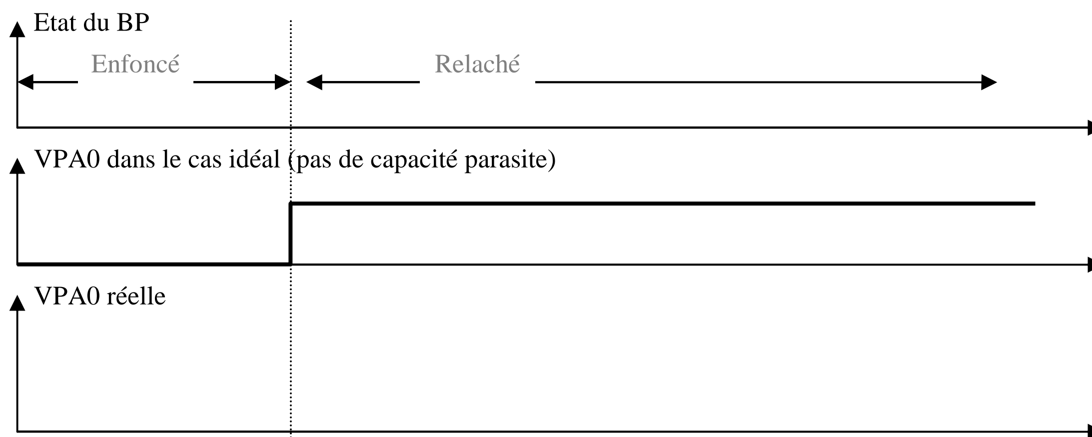
```
void main(void)
{
        // initialisations
        ..........................
        // superboucle du cycle µc
        while (1)
        {
          PORTA.1=!PINA.0;
        }
}
```

Dans certains cas, la présence d'une résistance peut avoir des effets (indésirables?) qu'il faut prendre en compte.
Reprenons le montage et supposons qu'une capacité parasite de 40pF est présente.



Le BP est maintenu enfoncé entre l'instant 0 et t1.
A t1, le BP est relâché.

Q12 : complétez le chronogramme



Remarque : vous ferez bien apparaître la constante de temps $\tau = R_{pu} \times Cp$

On considère que la charge est complète après 3τ.

Q13 : le microcontrôleur voit un état logique '1' sur ses entrées quand la tension présente sur son entrée dépasse 2.5V. Dans ce système, quel est le retard introduit par la présence de la capacité parasite ?

Exercice 3 : contrôle des bits d'un PORT

Pour forcer à '1' ou à '0' le bit d'un PORT de sortie sans modifier la valeur des autres bits du PORT, il faut utiliser des masques logiques :

ILLUSTRATION:

Contexte : les 8 lignes du PORTA sont reliées sur les ANODES de 8 leds (LED0 à LED7).

On suppose qu'initialement, le PORTA est dans l'état suivant :

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
0	0	1	1	0	0	1	0

et l'on veut placer le bit PA2 à '1' (pour allumer la LED2) et le bit PA1 à '0' (pour éteindre la LED1) sans modifier l'état des autres lignes du PORTA.

Donnez le résultat obtenu lorsque l'on effectue l'opération suivante (OU logique avec un masque):

	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
	0	0	1	1	0	0	1	0
OU	0	0	0	0	0	1	0	0
=								

Donnez le résultat obtenu lorsque l'on effectue l'opération suivante (ET logique avec un masque):

	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
	0	0	1	1	0	0	1	0
ET	1	1	1	1	1	1	0	1
=								

En Résumé

Pour **isoler** une ou plusieurs informations binaires présentes sur un port d'E/S, il est nécessaire d'effectuer une opération de **masquage** entre ce port et une **constante** appelée **masque**. La **mémorisation** du résultat de l'opération de masquage est réalisée en l'affectant à une **variable**.

<nom_variable> ← <Lecture_Contenu du Port> ET <Masque>

soit, par exemple, en C AVR :

<nom_variable> = PINx & <Masque>

La librairie `io.h` permet de travailler simplement avec les registres de configuration.

Remarque : pour utiliser cette librairie il faut écrire `#include <io.h>`

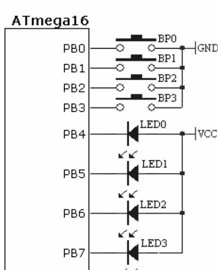
- Pour écrire une valeur dans un registre, on écrira :
  PORTA=0x33;
- Pour lire une valeur stockée dans un registre, on écrira :
  v=PINA; // v est une variable en C qui reçoit la valeur de l'octet
- Pour forcer un bit à 1, on écrira :
  `sbi(PORTB, 1); // PB1←1   ou encore en CODEVISION PORTB.1=1;`
- Pour forcer un bit à 0, on écrira :
  `cbi(PORTB, 1); // PB1←0   ou encore en CODEVISION PORTB.1=0;`

En utilisant les fonctions de la librairie `io.h`, donnez le code qui permet de réaliser la même opération que celle effectuée avec les masques.

Exercice 4 (entrainement personnel)

Rappelez le nom du registre qui permet de lire l'état logique d'un PORT qui a été au préalable configuré en ENTREE.

Dans le schéma ci-dessous, précisez :



- la configuration à donner au PORTB,
- l'état logique à transmettre sur les lignes PB4-PB7 pour d'allumer les LEDs,
- l'état logique lu sur les lignes PB0-PB3 quand un bouton poussoir est actif.

ILLUSTRATION:
On souhaite que l'état des 4 LEDs soit à l'image de l'état des 4 boutons poussoirs (bouton activé → led allumée !!!!).
L'algorithme de traitement est :

```
DEBUT
  Initialiser les PORTS
  Répéter toujours
    Recopier sur les sorties PB4-PB7 l'état des entrées PB0-PB3
  Fin répéter
FIN
```

Codez l'algorithme en C.

Pour bien comprendre la méthode, nous allons raisonner sur un exemple :
On suppose que le bouton poussoir BP1 est activé.

Quel est l'état de PINB ?

Quelle valeur faut-il envoyer sur PB4-PB7 ?

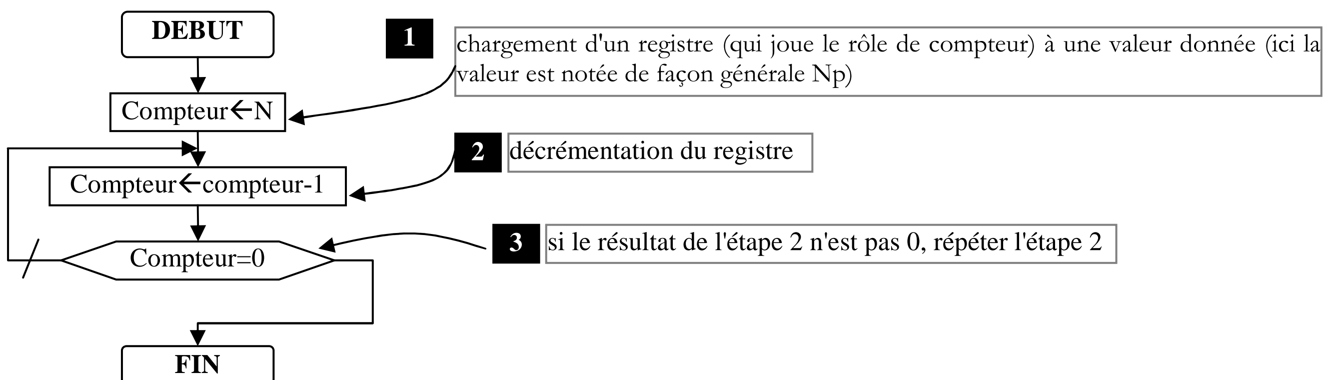
Quelle valeur faut-il imposer sur PB0-PB3 ?

Exercice 5

Un microcontrôleur est très souvent utilisé pour générer des intervalles de temps précis.
Pour créer des temporisations, vous pouvez :

- utiliser les compteurs programmables (timers) qui sont intégrés dans le µc,
- utiliser la notion de boucles dans un programme en prenant en compte la durée de chaque instruction

Un programme simple de temporisation se déroule comme suit :



La durée totale dépend du temps d'exécution de chaque instruction et de la valeur mise dans le registre.
La durée maximale dépend de la taille du registre (avec un compteur 8 bits, on a N=255 max). Toutefois, il est possible d'imbriquer plusieurs boucles les unes dans les autres pour obtenir des temporisations plus longues.
En C, il est impossible d'évaluer simplement la durée des instructions. Par conséquent, les fonctions de temporisation de base sont écrites en ASSEMBLEUR.

			NOMBRE DE CYCLES
tempo:			
	ldi	**r16,N**	
boucle:	**dec**	**r16**	
	brne	**boucle**	

Recherchez dans la documentation du constructeur sur le jeu d'instructions assembleur la colonne qui indique le nombre de cycles de chaque instruction.

4.1 Identifiez l'analogie entre le code assembleur et l'algorigramme.
4.2 Comptez le nombre de cycle de chaque instruction colonne (NOMBRE DE CYCLES).
4.3 Quel est le nombre de cycles nécessaires pour parcourir 1 fois la boucle?
4.4 Indiquez le nombre de cycles quand on exécute la séquence complète (TOT_CYCLE)?
4.5 Quelle est la temporisation la plus longue que l'on puisse obtenir si Fosc=16MHz (TMAX)?
4.6 Calculer la valeur de Np (format OCTET) pour obtenir une temporisation de 35µs.
4.7 Comment faire pour obtenir une temporisation plus longue ?

En langage C, il existe deux routines pour gérer les temporisations :

delay_us(valeur) pour une temporisation exprimée en µs (valeur est un entier entre 0 et 65 535)
delay_ms(valeur) pour une temporisation exprimée en ms

Ces deux routines sont définies dans la librairie `#include <delay.h>`.

Exemple :

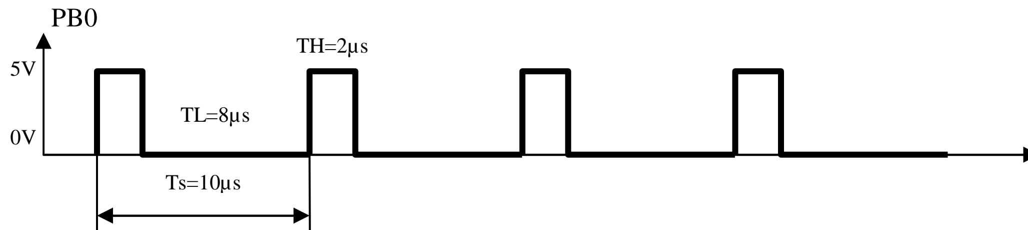
Pour obtenir une temporisation de 100µs, on écrira

```
delay_us(100);
```

Comment faire pour obtenir une temporisation de 10 minutes en C.

Exercice 6 (entrainement personnel)

On veut obtenir sur le bit 0 du PORTB un signal carré périodique d'allure ci-dessous.



Donner l'algorigramme et le code.

Vous ferez en sorte que les durées des signaux respectent précisément les critères présentés sur le chronogramme.

Il faudra alors vraisemblablement introduire du code assembleur à l'intérieur du code en C.

Voici comment procéder :

```
<ici le code en C>
#asm
      ldi r16,10
      <ici la suite du code en asembleur>
#endasm
```

Exercice 7

Ecrivez un programme C qui permette d'obtenir un chenillard sur le PORTB. Les bits seront décalés avec une période de 100ms. Pour cela, vous disposez d'une fonction déjà écrite nommée *`delay_ms(valeur)`* qui est enregistrée dans la bibliothèque *`delay.h`* fournie par le constructeur `(<delay.h>).`

Les LEDs du chenillard sont câblées en cathode commune à la masse. Dessinez le schéma du système.

IUT Angoulême

TD3 d'informatique industrielle
Geii1 – II2
Analyse de la structure d'un microcontrôleur

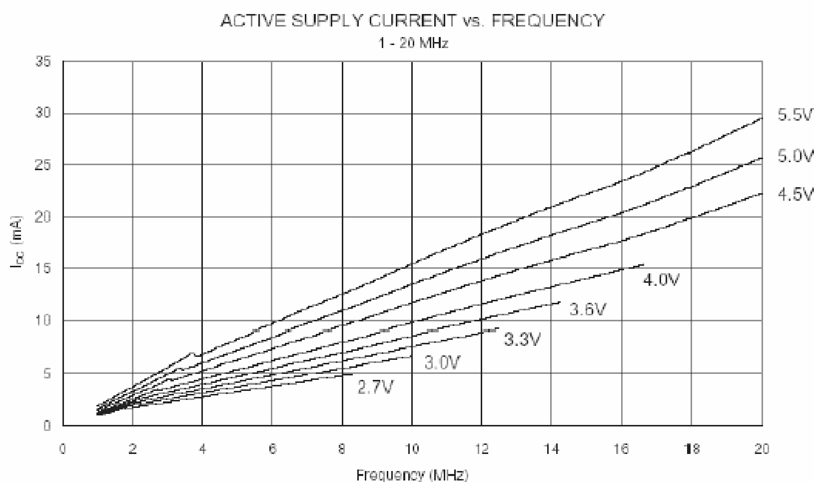
A partir de la documentation du microcontrôleur Atmega32, répondez aux questions suivantes :

Question 1

Citez les périphériques **internes** associés au microprocesseur. Donner leur rôle.

Question 2

Donnez les caractéristiques électriques de l'Atmega32 (tension max/min, fréquence max/min) [page 4]
Voici la caractéristique de consommation du composant (extraite de la doc. ATMEL) :

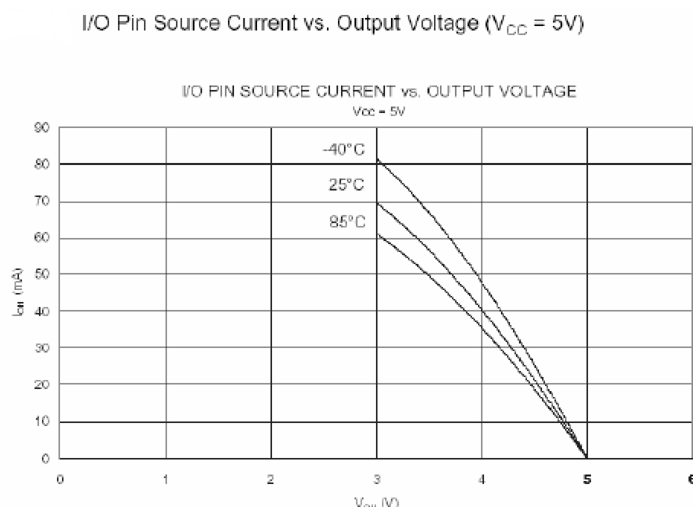


2.1 Que concluez-vous concernant la loi de variation consommation/fréquence?
Les microcontrôleurs sont la plupart du temps intégrés dans des systèmes embarqués. Ils sont donc alimentés par piles ou par accumulateur.
Classiquement, on trouve les caractéristiques suivantes pour une pile alcalines LR6 1.5V : capacité 1500mAh.

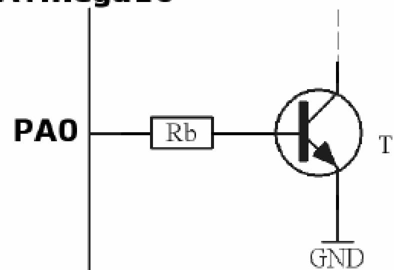
2.2_Quelle est l'autonomie (la durée de fonctionnement) du montage si le microcontrôleur est alimenté en 4.5V/5mA de consommation?
2.3_Que devient cette autonomie en mode Idle mode, sous 3V à 1MHz?

Question 3

Quel est le courant maximal "NORMAL" que peut débiter/absorber une broche du composant?
APPLICATION:
Le microcontrôleur doit fournir sur la broche PA0 un courant de 40mA pour alimenter une charge.



Ib=40mA
Vbe=1.7V



Calculer la valeur de Rb en tenant compte des caractéristiques de sortie en courant présentée ci dessous.

Question 4

Quelle est la quantité de mémoire embarquée ? Détaillez le rôle de chaque mémoire.

Question 5

A quoi sert une interruption ?
De combien de sources d'interruptions dispose-t-on dans ce microcontrôleur ?
Ces différentes sources ont-elles un ordre de priorité ?

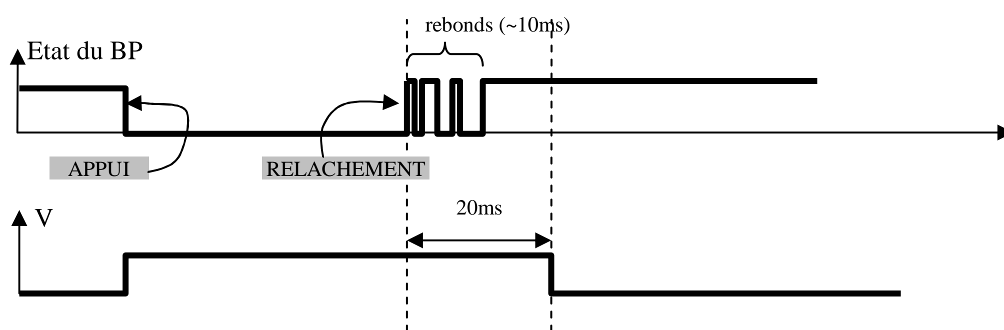
En général, les systèmes à concevoir (donc les programmes) sont complexes et exigent une méthode de spécification adéquate et claire.
La **machine à états** est un des outils efficace permettant de décrire le fonctionnement des systèmes.

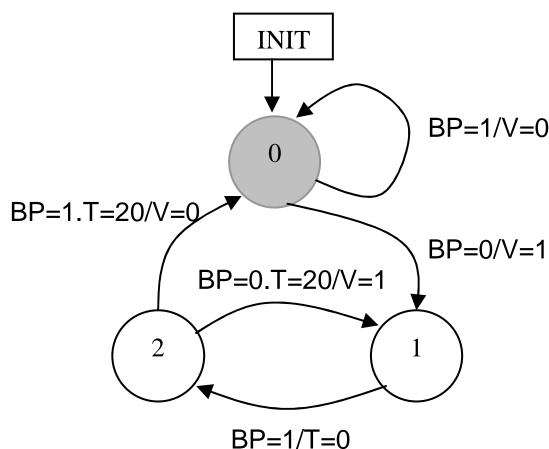
Illustration

Pour l'exemple, nous allons gérer les rebonds d'un bouton poussoir (BP).
La variable V doit prendre la valeur 1 quand le BP est activé, 0 sinon.
Il faut savoir qu'un phénomène de rebonds apparaît lors du relâchement du BP (les lames du BP vibrent mécaniquement). Pour traiter ces rebonds nuisibles, on peut mettre en œuvre le principe suivant :

au relâchement, c'est-à-dire quand un front montant est détecté, on attend 20ms (la valeur de 20ms est empirique) puis l'on vient relire l'état du BP:

- s'il est encore à l'état 1, on en conclue que le BP a été effectivement relâché
- s'il est à 0, le front montant était un simple parasite, on attend le front montant suivant.





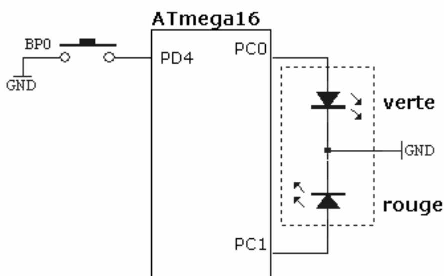
Méthodologie

Voici les étapes que je vous recommande de suivre pour construire précisément un diagramme d'états.
1. Identifiez toutes les entrées. Pour l'exemple, il s'agit de BP.
*Remarque : ici, la variable T est une **variable interne**; elle s'incrémente automatiquement toutes les 1ms (on verra plus tard comment) et elle est manipulée à la fois comme entrée ou comme sortie.*
2. Énumérez toutes les sorties. V dans notre cas.
3. Énumérez tous les états possibles. Nous avons trois états (0, 1, 2).
4. Créez une variable assez grosse pour contenir tous les états possibles. Encoder les états sur le nombre nécessaire de bits. Pour notre exemple on a E0 et E1 avec encodage {00, 01, 10, 11}.
5. Assignez un état initial au système. Le graphe montre que cet état est 0.
Ce qui est mentionné précédemment (points 1, 2, 3,4 et 5) est vrai tant pour les systèmes matériels que logiciels.
Cependant, pour l'aspect logiciel, nous devons ajouter quelques remarques supplémentaires pour tenir compte de certaines réalités:
6. La variable contenant l'état du système peut être un peu plus grosse que nécessaire pour une simple question de commodité. Dans les cas que nous traiterons, un type uint8 est généralement suffisant (255 états !!).
7. Il faut souvent créer des variables (au moins temporaires) pour représenter les entrées et les sorties.
8. Codez les instructions qui permettent le changement d'un état à un autre en fonction de l'état présent et des entrées. Généralement, cette partie est écrite avec une instruction "switchcase".
9. Programmez le comportement de chaque état de façon claire et bien identifiée.

Voici le code en C de cette machine :

```
#include <io.h>
#include <iut/types.h> // pour les types de variables
#include <delay.h>
// construire les états possibles
typedef enum {ETAT0,ETAT1,ETAT2} tetats;
// équivalences
#define V PORTC.0
#define BOUT_POUS PIND.4
void initAVR(void) {
  DDRD.4=0;PORTD.4=1; // bouton poussoir en entrée avec pull-up
  DDRC.0=1;PORTC.0=0; // led eteinte
}
/* ______ TRAITEMENT PRINCIPAL -----------------*/
void main(void) {
tetats etat=ETAT0;
uint8 T=0;
bit BP;
  // initialiser les périphériques
  initAVR();
  // moteur ... de la machine !!!
  while (1) {
    // lire les entrées
     BP=BOUT_POUS;
    // faire évoluer la machine en fonction des états et des entrées
    switch (etat) {
      case ETAT0 :
        if (BP==0) {etat=ETAT1; V=1;} else V=0;
      break;
      case ETAT1 :
        if (BP==1) { T=0;etat=ETAT2;}
      break;
      case ETAT2 :
        if (BP==0 && T==20) { T=0;etat=ETAT1;}
        if (BP==1 && T==20) { etat=ETAT0;}
      break;
    }
    // temporiser entre deux tops d'horloge
   delay_ms(1);
   T++; // incrémenter la variable interne
  }
}
```

Application 1



Le microcontrôleur doit contrôler une DEL bicolore.
Quand le système démarre, la DEL doit s'allumer en rouge. Si le bouton-poussoir BP0 est activé, la DEL affiche la couleur ambre.
Quand le bouton-poussoir est relâché, la DEL devient verte. Si le bouton est encore activé, la DEL prend la couleur ambre. Quand il est relâché, la DEL s'éteint. Si le bouton est encore activé, la DEL affiche la couleur ambre. Quand il est relâché, la DEL tourne au rouge ce qui fait que le système est de retour à son état initial et tout peut recommencer.

Application 2

On utilise toujours la même LED bicolore, mais cette fois ci, le défi est de concevoir un jeu de réflexe.
Quand le microcontrôleur démarre, il attend 10 secondes pendant lesquelles la led clignote en rouge à la cadence d'1/10 de seconde. Quand la lumière est éteinte, le joueur doit activer le bouton poussoir aussitôt que possible. Si le joueur active le bouton à l'intérieur d'une seconde, la DEL devient verte. Si le joueur active sur le bouton passé une seconde, la lumière prend la couleur rouge.
Il faut appuyer à nouveau sur le bouton poussoir pour revenir au départ.

Nous allons étudier les temporisateurs compteurs (timers-counters), en s'intéressant à ceux contenus dans le microcontrôleur Atmega16 d'Atmel.
Nous avons vu qu'il est possible de réaliser des temporisations en décrémentant des registres imbriqués les uns dans les autres. Mais il s'avérait difficile de calculer exactement la temporisation réalisée, et pour de longues temporisations, la construction était fastidieuse. De plus, l'énorme inconvénient de cette méthode réside dans le fait que le microprocesseur est occupé pendant ce calcul de temporisation.
Utiliser un timer permet par contre de :

- réaliser facilement des temporisations précises et importantes
- libérer le microprocesseur pour d'autres tâches pendant l'écoulement de la temporisation.

Informations techniques

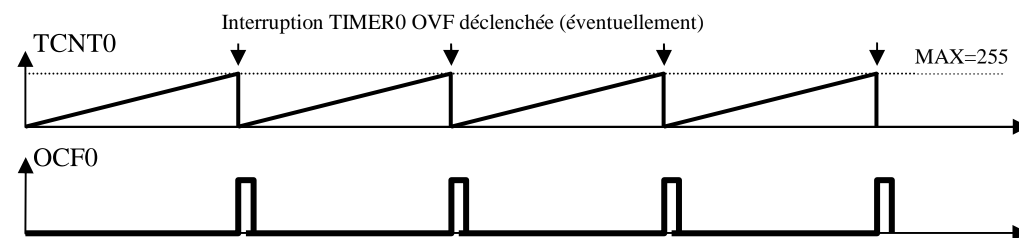
L'AVR Atmega16 contient 3 timers :

- 2 timers 8 bits (TIMER0 et TIMER2) qui comptent de 0_255 (2^8-1)
- 1 timer 16 bits (TIMER1) qui compte de 0 à 65535 (2^{16}-1)

Chaque timer possède son horloge notée Htimer. Cette horloge peut être construite à partir de l'horloge du microcontrôleur sur laquelle on applique un facteur de pré division (/1, /8, /64, /256,/1024).
Les timers peuvent fonctionner de plusieurs façons, cela dépend de la manière dont vous allez les configurer. Nous allons nous intéresser plus particulièrement au timer/counter 0.

TIMER0 : mode normal

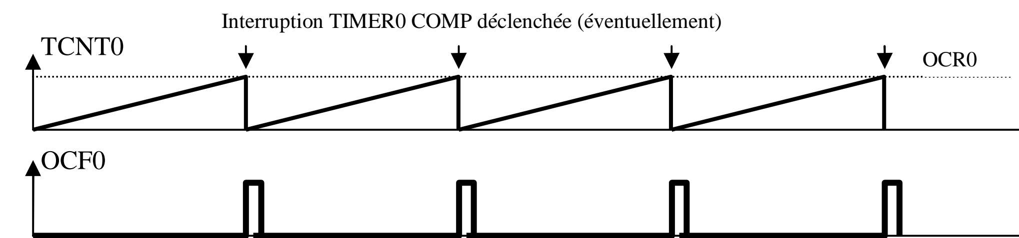
C'est le mode le plus simple (**WGM01:0** à 0). Le compteur compte à partir de 0. Quand il atteint la valeur maximale 255 (**$FF)** il déborde, le drapeau **TOV0** est mis à 1, puis le compteur repart de 0.
Si on le souhaite, une interruption de débordement peut être déclenchée (mettre un 1 dans le bit 0 du registre TIFR).
Le drapeau **OCF0** est automatiquement remis à 0 lors de l'exécution de l'interruption.
ATTENTION, si l'interruption n'est pas programmée, le drapeau doit être remise à 0 manuellement (en écrivant un 1



dedans !!!!!).

TIMER0 : mode remise à 0 sur Comparaison (CTC)

Dans le mode **CTC** (**WGM01:0** à 2), le registre **OCR0** est utilisé pour définir la résolution du compteur.
Le compteur est incrémenté et quand sa valeur courante est égale à **OCR0**, le compteur est remis à 0 et le drapeau **OCF0** est mis à 1 pour déclencher une éventuelle interruption (dans ce cas, mettre un 1 dans le bit 1 du registre TIFR).



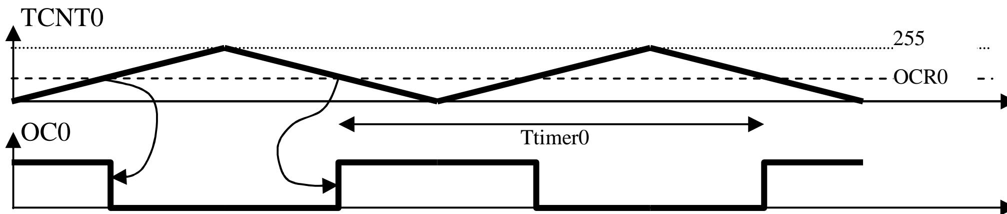
La période du TIMER0 est donnée par :

Ttimer0=Tµc x N x (OCR0+1)

Avec **N** le rapport du pré diviseur (1, 8, 64, 256 ou 1024) et **et Tµc** la période d'horloge du quartz.

TIMER0 : mode PWM correct (PWM C)

Ce mode fait travailler le compteur dans les deux sens (comptage et décomptage).
Il permet de produire un signal modulé en largeur d'impulsion sur la broche OC0 (PB3) du microcontrôleur (la broche doit être programmée en sortie).
TCNT0 passe de 0 à 255 et ensuite de 255 à 0. Dans le mode montant, la sortie **OC0** est remise à 0 sur comparaison entre **TCNT0** et **OCR0** et dans le mode descendant, la sortie sur **OC0** est mise à 1 sur comparaison entre **TCNT0** et **OCR0.**
La sortie OCO doit être configurée en mode CLEAR ou SET.



La période du TIMER0 est donnée par :

Ttimer0=Tµc x N x (255*2)

La durée de l'ETAT HAUT est : TH= Tµc x N x 2 x OCR0
La durée de l'ETAT BAS est : TL= Tµc x N x 2 x (255-OCR0)

Question 1

Citez les registres de configuration du timer 0.

Exercice 1 : analyse de la période d'un signal

Soit une horloge de **8MHz**. TCNT0 compte de 0 à 255 (mode NORMAL). OCR0 contient la valeur 100.
Quelle est la fréquence du signal observé sur **OC0** si le TIMER0 est configuré avec TCCR0 = 0001 0011

Exercice 2 : analyse d'une temporisation simple

Voici une fonction en C réalisant une temporisation:

```
void tempo(void) {
  TCCR0=(0<<CS02)|(1<<CS01)|(0<<CS00);
  TCNT0=0;
  TIFR|=(1<<OCF0);
  OCR0=200;
  while ((TIFR & (1<<OCF0))==0);
}
```

1. Analysez très en détail la routine tempo; bien identifier les registres utilisés, ainsi que le mécanisme de scrutation et de remise à zéro de "l'interrupt flag" OCF0.
2. Quelle est la durée de la temporisation réalisée?
3. Quels sont les deux moyens pour modifier cette temporisation?
4. Modifiez le programme pour obtenir une temporisation de 1ms.
5. Quelle est la temporisation maximale que l'on peut programmer?
6. Cette fonction est-telle BLOQUANTE?
Une fonction est dite BLOQUANTE quand le système est arrêté dans la fonction (comme dans une boucle).

Exercice 3 : timer en interruption

Les timers sont couramment utilisés pour lancer un traitement à intervalle de temps régulier.
On parle alors de traitement en tache de fond.
Le traitement principal est décrit dans la super boucle `while(1)`.
Ici, on veut faire un chenillard (périodicité 2s) sur PORTD dont le sens est modifiable par un bouton poussoir BP0 (câblé sur PC0).
Le chenillard sera décrit dans la boucle principale du `main()`.
Le timer0 sera programmé en interruption toutes les 10ms pour venir évaluer l'état du bouton poussoir et redéfinir (éventuellement) le sens de déplacement.

Complétez le code :

```
// ___ Librairies  utiles
#include <io.h>
#include <iut/TPboard.h>
#include <delay.h>

#define   DROITE  0
#define   GAUCHE  1

u08 sens=GAUCHE;

// Timer0 Compare A : interruption déclenchée toutes les 10ms
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
   ………… < à compléter >
}
```

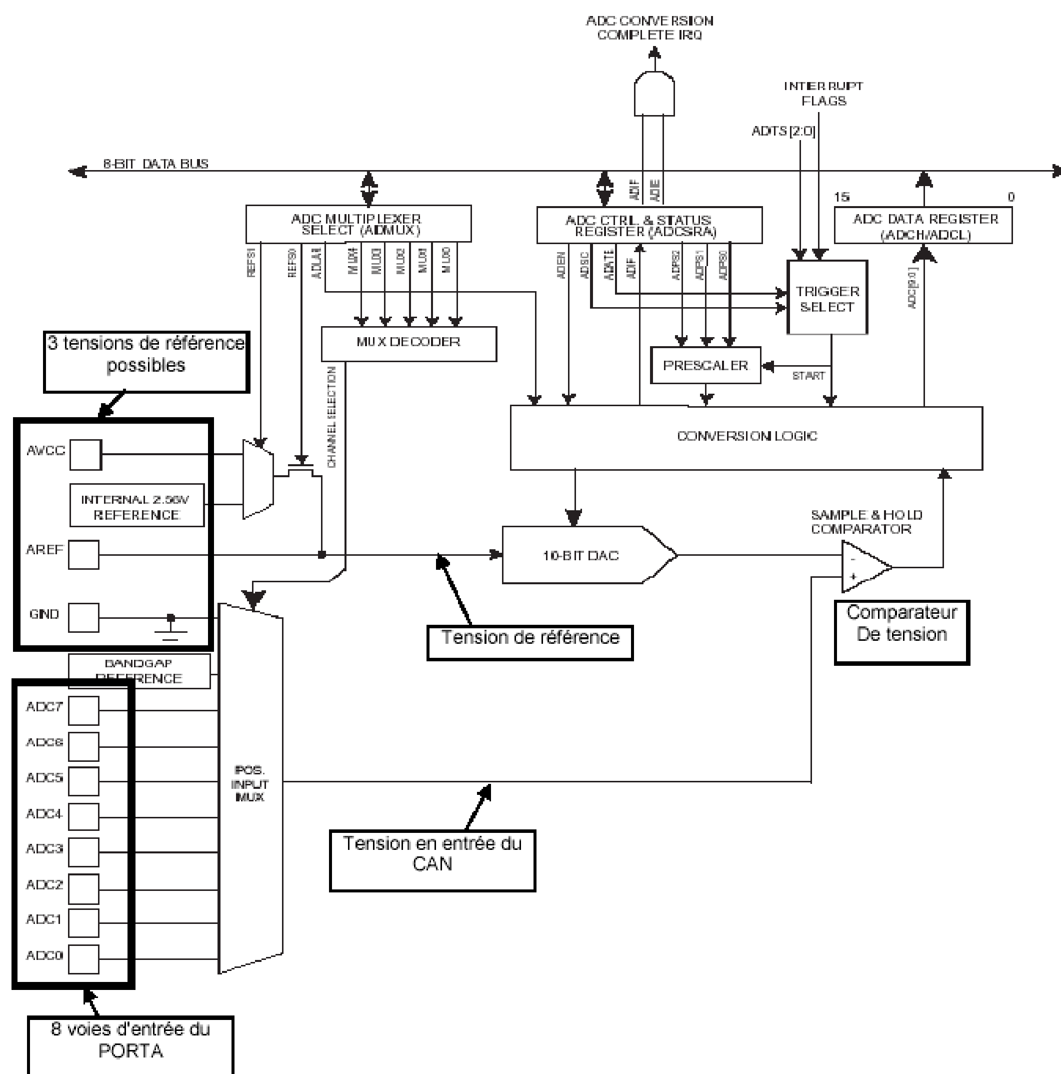
```
/**
 * Initialisations des périphériques du µc.
 */
void init(void)
{
  BARGRAPH_DDR=0xFF;BARGRAPH=1;
  BP_ROUGE_DDR=0;BP_ROUGE_PORT=1;
  // timer0
  TCCR0=……………………………………………………………………………
  TCNT0=…………………………………………………………………………………
  OCR0=……………………………………………………………………………………
  TIMSK = ………………………………………………………………………………
  sei(); // pour autoriser les interruptions !!!!

// ____ traitement PRINCIPAL ____
int main(void)
{
  init();
  while(1)
  {
    if (sens==GAUCHE)
      BARGRAPH<<=1;
    else
      BARGRAPH>>=1;
    if (BARGRAPH==0) BARGRAPH=1;
    delay_ms(1000);
  }
  return 0;
}
```

RESSOURCE : document sur le microcontrôleur ATMEL ATMEGA (pages 24 à 30).
L'Atmega16 contient un convertisseur Analogique Numérique avec une résolution de 10 bits.
On peut le configurer pour ramener la résolution sur 8 bits.
Le CAN est de type à approximations successives (il lui faudra plusieurs cycles de son horloge CAN pour faire une conversion).
Quand la fonction convertisseur est activée dans le microcontrôleur, les 8 lignes du **PORTA** ne sont pas considérées comme des entrées-sorties numériques mais comme les **8 voies d'entrée du convertisseur**.
Il faut alors programmer le registre ADMUX pour indiquer le numéro de voie à convertir.
Il faut bien comprendre que le convertisseur possède 8 voies d'entrée, mais, à un instant donné, il ne peut faire qu'une seule conversion, on parle de voies d'entrées multiplexées.
Le circuit utilise une référence de tension. On peut choisir parmi trois tensions possibles : une tension externe que l'on doit appliquer sur la broche AREF, la tension d'alimentation du microcontrôleur (+5V) ou une source de tension interne de 2.56V.

REMARQUES :
- la 1ère conversion qui suit la validation de la fonction CAN est plus longue, en effet le CAN procède à certains ajustements internes. Cette 1ère conversion nécessite 25 cycles d'horloge CAN.
- si on se limite à une résolution de 8 bits (En ignorant les 2 bits de pds faible) la fréquence de l'horloge de conversion peut être choisie jusqu'à 1MHz. (Voire 2MHz si résolution utile 6 bits)

Le schéma fonctionnel du convertisseur est le suivant :



la programmation s'opère à l'aide de trois registres

- **ADMUX (ADC Multiplexer Select Register)** pour sélectionner la voie d'entrée
- **ADCSR (ACD Control and Status Register)** pour paramétrer le mode de fonctionnement
- **ADCW=(ADCL, ADCH)** qui contient le résultat de la conversion sur 10 bits

Il est possible de déclencher une interruption (numéro 15 – ADC conversion complete) quand le processus de conversion est terminé.
Le convertisseur reconnaît 2 modes de fonctionnement :

- en mode simple conversion (**single conversion**)
  il faut donner l'ordre pour lancer une nouvelle conversion et le CAN n'effectue qu'une seule conversion.
- en mode conversion automatique (**free run**)
  il faut donner l'ordre pour lancer une première conversion; quand celle-ci est terminée, le CAN relance automatiquement une nouvelle conversion.

La valeur numérique en sortie du CAN est donnée par (Ve est la tension à mesurer) :

CAN=1024*Ve/VREF si le CAN travaille en mode 10 bits (2^{10}=1024)

CAN=256*Ve/VREF si le CAN travaille en mode 8 bits (2^8=256)

Consulter la page 26,27 de la documentation AVR pour répondre aux questions 1,2.

QUESTION 1

Quelle est la fréquence de fonctionnement MAXIMALE quand le CAN travaille en résolution maximale de 10 bits?

QUESTION 2

Quel est le temps de conversion en mode normal ?
On place la valeur **xxxx x110** dans le registre ADCSRA.
Quelle est la valeur de la fréquence d'horloge du CAN? Que pouvez-vous dire?
Quel est le temps de conversion?

QUESTION 3

Quelle est la valeur numérique que donne le CAN si VREF=2.56 et Ve=0.8V en mode 8 bits, 10 bits ?
Avec VREF=5V, quelle est la plus petite tension mesurable en mode 8 bits?
Si le CAN donne une valeur de 79, quelle est la tension d'entrée (VREF=5V).

Application 1

On veut écrire un programme simple de lecture de la tension de PA2 convertie au format 10 bits avec une tension de référence à VCC.
La tension est lue une seule fois.
Pour cela, on utilise une procédure scrutation du bit 4 ADIF du registre ADCSRA [(1)]
Déterminez **l'algorithme** des éléments de programme suivant:

1. Sous-programme `init` : définition du registre ADMUX et programmation des directions (PORTA).
2. Sous-programme `acquisition` : attente ADIF et lecture de la valeur convertie.

Le programme principal `main` est construit comme suit :

```
MAIN
DEBUT
valeur : entier

  init
  valeur←acquisition
FIN
```

[(1)] le bit ADIF du registre ADCSRA est un bit d'information (drapeau). Initialement (en début de conversion), il doit être forcé à 0 (en écrivant un 1 dedans); il passe automatiquement à 1 en fin de conversion.

Application 2

On se propose maintenant d'écrire un programme de lecture de la donnée en interruption afin de ne plus attendre continûment la montée de ADIF. Le processeur pourra réaliser ainsi, en même temps, une autre application.

1. Algorithme du sous-programme `init`: initialisation de l'interruption.
2. Algorithme du sous-programme `interruption`.

Le programme principal `main` est construit comme suit :

```
MAIN
DEBUT
valeur : entier

  init
FIN
```

Application 3

On veut réaliser maintenant des périodes d'échantillonnage constantes et calibrées (Réf. de tension: AVCC).
Pour cela, utiliser le TIMER0 de l'AVR.
On veut faire 100 mesures d'un signal (appliqué sur PA0) dont la fréquence maximale est de 10kHz.
On choisit donc une fréquence d'échantillonnage de 20kHz.

Justifiez cette valeur d'échantillonnage !!!

Les 100 mesures (au format 8 bits) seront placées dans le tableau tech[100];

Complétez le code du programme principal et des programmes d'interruptions pour résoudre le problème.

```
#include <avr/io.h>
#include <avr/interrupt.h>

// ___ Prototypes pour les fonctions
void init(void);
u08 acquisition(void);

// ___ Variables globales
u08 tech[100],Nb=0;

// ____ traitement PRINCIPAL ____
int main(void)
{
  init();
  while(1);
  return 0;
}

/**
 * Initialisations des périphériques du µc.
 */
void init(void)
{

  // timer0
  < à compléter … >

  // can
  < à compléter … >

  // it autorisées
  sei();
}

/**
 * Interruption TIMER0 de comparaison (50µs). Fech=20kHz
 */
ISR(TIMER0_COMP_vect)
{
  < à compléter … >
}


/**
 * acquisition d'une grandeur analogique.
 */
u08 acquisition(void) {

  < à compléter … >

}
```