Polycopiés (suite)

Cours Informatique embarquée

GEII Toulouse 2020-2021

A. Nketsa

CHAP 6 Les entrées-sorties numériques d'un système à base de microprocesseur

1- Approche intuitive

Un microcontrôleur permet de :

- contrôler un environnement

  Cela signifie surveiller ⇒ lire les capteurs qui renseignent le µC sur l'environnement qu'il doit surveiller.

- commander à un environnement

  Cela signifie donner des ordres à des dispositifs

Exemple

On dispose de 2 interrupteurs (inter0 et inter1) pour commander une lampe de la façon suivante :

- inter0 = 1 permet d'allumer la lampe
- inter1 = 1 permet d'éteindre la lampe si inter0 est à 0

Une analyse simple de cet exemple montre que :

- lorsque inter0 =1 alors la lampe s'allume
- la lampe reste allumée même lorsque inter0 retombe à 0 et inter1 = 0

  Conclusion :

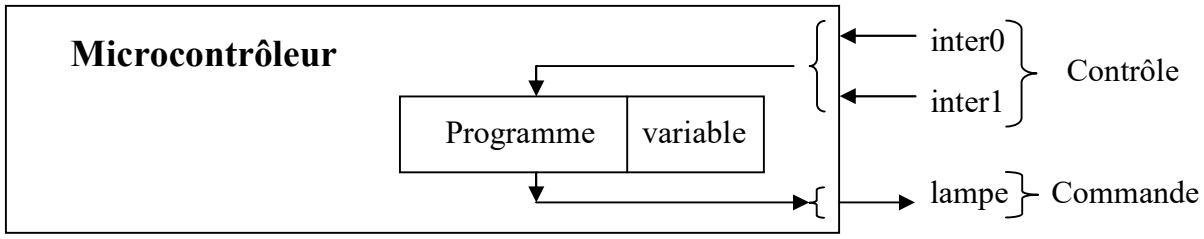
  ☞ il y a une mémoire qui conserve l'état de la lampe après l'action de inter0

  ☞ Comme nous sommes en électronique numérique : cette mémoire est une bascule D
- lorsque inter1 passe à 1 et inter0 est à 1 alors la lampe reste allumée
- lorsque inter1 passe à 1 et inter0 est à 0 alors la lampe s'éteint

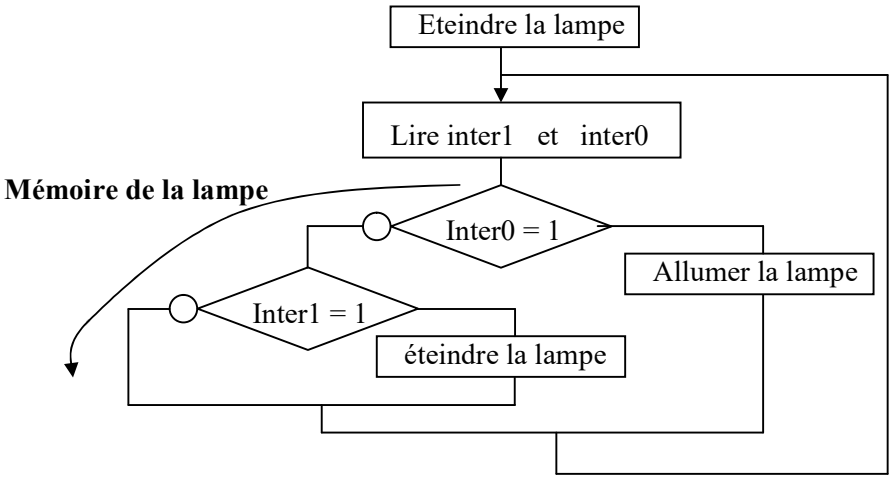
Bilan de cet exemple

☞ Inter0 et inter1 sont des **entrées** pour le microcontrôleur (renseignent le µC) **: contrôle**

☞ la lampe est une **sortie** (par laquelle le pC (donne des ordres) **: commande**



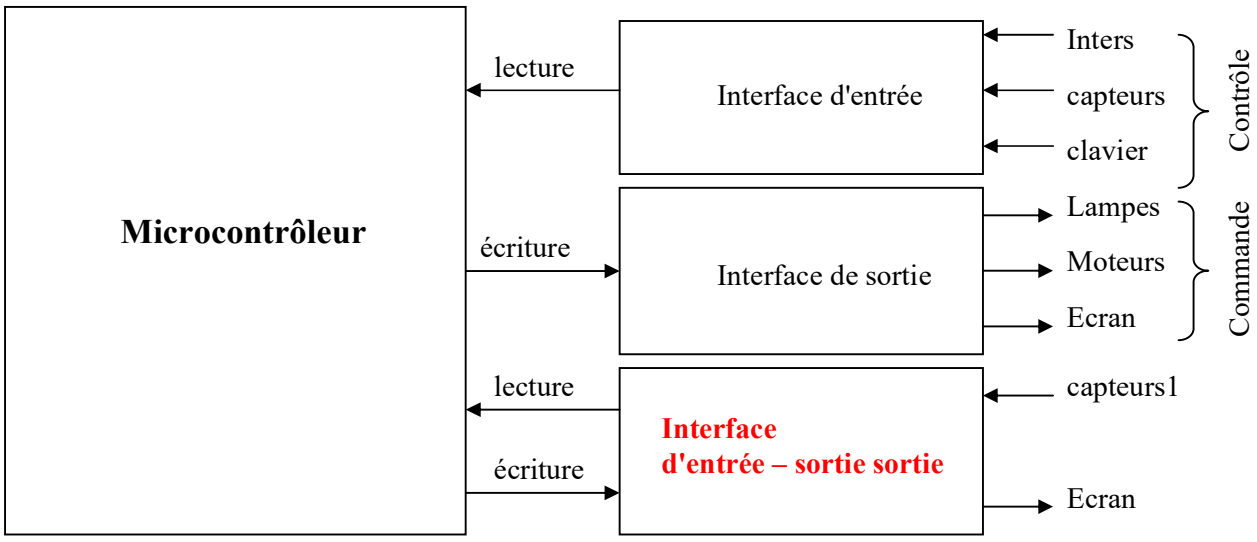
L'organigramme du programme pourrait être :



Conclusion

Le µC dispose

- des interfaces d'entrée pour convertir les informations d'entrée à mettre à sa disposition quand il en a besoin
  - comme on se met à la place du µC, on dit que le µC lit les interfaces d'entrée
- des interfaces de sortie pour convertir les informations binaires envoyées par le µC vers les éléments à commander.
  - comme on se met à la place du µC, on dit que le µC écrit sur les interfaces de sortie
- **plus généralement des interfaces d'entrée-sortie qui intègrent les 2 types d'interfaces. Dans ce cas,**
  - **on doit choisir la direction de l'interface :**
    - **soit en entrée**
    - **soit en sortie**
  - **on peut relire la dernière valeur écrite sur le port de sortie si celui-ci a été programmé en sortie.**
  - **il est conseillé que les bits non utilisés du port soient en entrée**



Remarque :

En TP, nous utiliserons les interfaces d'entrée-sortie

Définitions et vocabulaire

☞ Interface d'entrée

Définition :

Une interface d'entrée est un dispositif qui met à la disposition du microprocesseur quand il en fait la demande des données présentes sur les entrées de l'interface au moment de la demande.

A retenir :

L'interface d'entrée ne mémorise pas d'information.
Elle ne peut être que lue.

Elle fournit l'information au microcontrôleur uniquement au moment où celui en fait la lecture

Vocabulaire :

L'interface d'entrée est aussi appelée port d'entrée ou simplement entrée

☞ Interface de sortie

Définition

Une interface de sortie est un dispositif qui met à la disposition des périphériques lents la donnée que le microprocesseur présente de façon brève sur le bus de donnée.

A retenir :

L'interface de sortie mémorise la dernière donnée écrite.
Elle est constituée de bascules D
Elle ne peut pas être lue.

La dernière information écrite sur le port de sortie est mémorisée

Vocabulaire :

L'interface de sortie est aussi appelée port de sortie ou simplement sortie

☞ **Interface d'entrée-sortie**

Définition :

Une interface d'entrée-sortie est un dispositif qui contient à la fois une interface d'entrée et une interface de sortie qui se partage les mêmes broches d'entrée-sortie.

Dans ce type d'interface, on ajoute une bascule pour choisir la direction de la broche.

A retenir :

1- On choisit la direction d'une broche en écrivant dans le registre de direction.

2- Lorsque l'interface d'entrée-sortie est configurée en entrée,

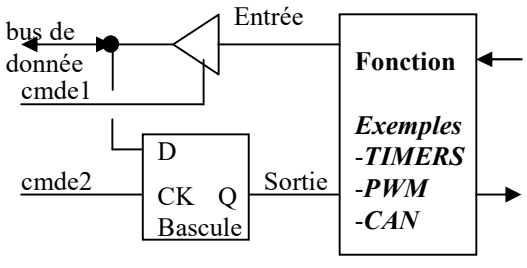
- il n'y a pas de mémorisation de l'information d'entrée
- le µC lit l'information présente sur le port d'entrée au moment de la lecture
- une écriture sur un bit configuré en entrée n'a pas d'effet.

1- Lorsque l'interface d'entrée-sortie est configurée en sortie,

- la dernière information écrite est mémorisée,
- la dernière information écrite peut être lue.

4- Une interface d'entrée-sortie non utilisée doit être configurée en entrée par souci de protection des sorties

Interface fonctionnelle

Une interface fonctionnelle est constituée de 2 parties : - une partie est en entrée-sortie - une partie est fonctionnelle Nous reviendrons sur cette structure un peu plus tard	

Application au microcontrôleur C167

Le microcontrôleur C167 possède plusieurs ports d'entrée-sortie intégrés. Mais nous n'utiliserons que les suivants :

- le port **P2 de 16bits**
- le port **P3 de 13bits** (ce port est multifonction)
- le port **P7 de 8bits** (ce port est multifonction)
- le port **P8 de 8bits**

Chaque port possède :

- un **registre de donnée** (en entrée ou en sortie selon la direction) **nommé Pi** comme le port
- un **registre de direction** qui permet de programmer la direction du registre de donnée **nommé DPi**

Chaque bit de chaque port **est indépendant**.
Nous noterons dans un premier temps **Px.y** le bit y du registre de donnée du port **Px**..
DPx.y le bit y du registre de direction du port **DPx.**
Chaque bit d'un port peut être programmé en entrée ou en sortie en fixant DPx.y

DPx = 0 **port en entrée** c'est l'état à l'initialisation du processeur
DPx = 1 **port en sortie**

On peut programmer plusieurs bits d'un port avec des directions différentes.
Pour cela, il suffit d'utiliser les masques.

Rappel:

1- L'interface d'entrée-sortie mémorise la dernière donnée écrite.

2- On choisit la direction d'une broche en écrivant dans le registre de direction.

3- L'interface peut être lue et écrite si elle est configurée en sortie

- la dernière information écrite est mémorisée,
- la dernière information écrite peut être lue.

4- Si l'interface est en entrée, l'écriture sur l'interface ne change rien.

5- Une interface d'entrée-sortie non utilisée doit être programmée en entrée par souci de protection des sorties

Programmation en langage C

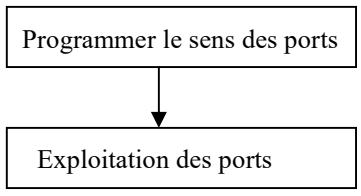
Le compilateur du langage C du microcontrôleur C167 connaît tous les ports intégrés, P2, P3, P7 et P8 et leurs registres de direction DP2, DP3, DP7 et DP8.

Les registres de donnée et de direction sont bit-adressables. C'est-à-dire que chaque bit de ces registres est directement accessibles et manipulables. Nous n'utiliserons pas cette possibilité pour les ports d'entrée-sortie. Nous préférons les masques

Séquence de programmation

Pour programmer un système avec des ports d'entrée-sortie, on doit respecter la séquence ci-contre :

- **le sens des ports est programmé une seule fois**
- **L'exploitation fait partie de la boucle du programme principal.**



Langage C (suite 4)
(Utilisation des masques en informatique embarquée)

- Principe général
- Gestion des capteurs
- Actualisalisation des sorties

En informatique embarquée, les capteurs et les acyionneurs binaires sont souvent regoupés pour formés des mots (octet ou mot de 16bits)
Lors de l'utilisation, on a souvent besoin de ne tester ou manipuler qu'un ensemble de ces bits.
Pour cela : il faut les isoler pour les manipuler
Nous avons vu lors des instructions logiques que l'on peut isoler un ou plusieurs bits dans un mot, de même en utilisation les instructions booléennes on peut tester ces bits

Rappel
Pour isoler les bits ,

- on construit un masque en ET
- et on effectue l'opération logique ET (&)

Pour tester les bits isolés,

- on construit un masque avec la valeur attendue des bits à tester
- on effectue une comparaison entre le résultat des bits isolés et le masque des valeurs attendues

Pour mettre des bits à 0

- on construit un masque en ET des bits à mettre à 0
- on effectue un ET logique (&) entre le mot et le complément à 1 (~) du masque en ET

Pour mettre des bits à 1

- on construit un masque en ET des bits à mettre à 1
- on effectue un OU logique (|) entre le mot et le masque en ET

Pour compémenter des bits à 1

- on construit un masque en ET des bits à complémenter
- on effectue un OUexl (^) entre le mot et le masque en ET

Gestion des capteurs

Considérons que nous avons un octet qui regroupe : les capteurs, les interruopteurs et les poussoirs d'un système occupant dans l'octet les positions suivantes :

Variable : octet_entree

poussoirs		interrupteurs		capteurs			
B7							B0
Pouss1	Pouss0	Inter1	Inter0	C3	C2	C1	C0

On admet que les capteurs sont actifs à 0

Gestion des actionneurs

Considérons que nous avons un octet qui regroupe : les actionneurs d'un système occupant dans l'octet les positions suivantes :

Variable : octet_sortie

poussoirs		interrupteurs		capteurs			
B7							B0
buzzer	sirène	moteur1	moteur0	Led3	Led2	Led1	Led0

On admet que les actionneurs sont actifs à 1

Exercice

si c2=0 et inter0=1 et pouss0 = 0
alors allumer la Led2
sinon eteindre la led2 et complémenter la led3

Donc il faut isoler les bits C2, inter0 et poussà dans octet_entree

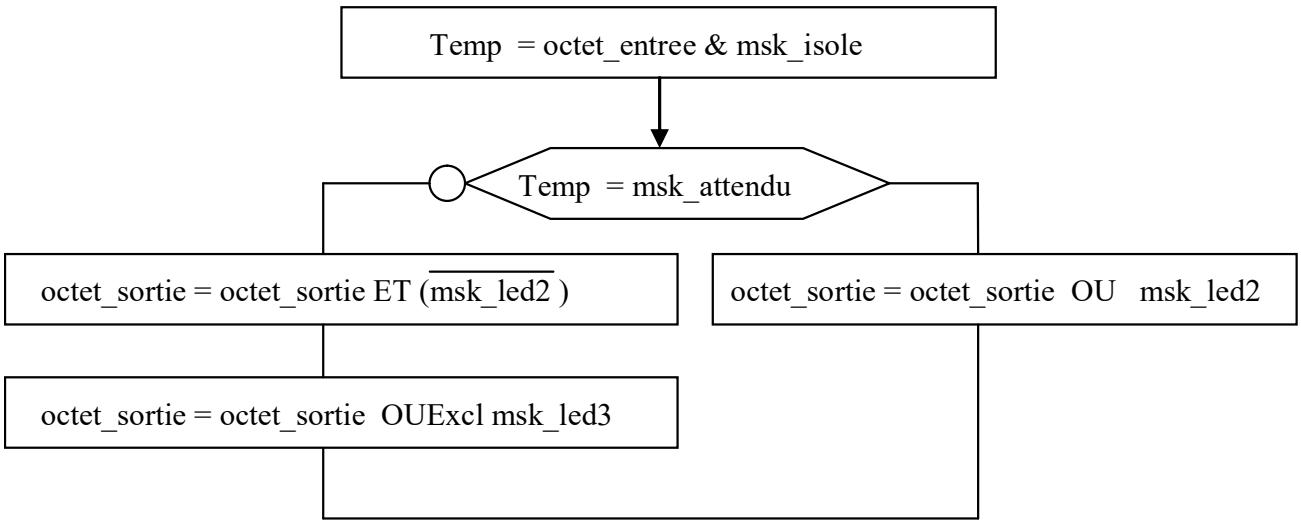
	b7							b0
	Pouss1	Pouss0	Inter1	Inter0	C3	C2	C1	C0
Masque en ET (isolement) Msk_isole	0	1	0	1	0	1	0	0
Masque valeur attendue Msk_attendu	0	0	0	1	0	0	0	0

- masque en ET : 01010100b ≡ 0x54
- masque de test : 00010000b ≡ 0x10

Pour allumer la led2 sans modifier les autres, il faut construire le masque correspondant

	b7							b0
	buzzer	sirène	moteur1	moteur0	Led3	Led2	Led1	Led0
Masque en ET pour mise à 1 led2 (msk_led2)	0	0	0	0	0	1	0	0
Masque pour complémenter led3 (msk_led3)	0	0	0	0	1	0	0	0

D'où le programme



Traduction en langage C

```
Temp = octet_entree & msk_isole;
if  (Temp == msk_attendu)
  {
     octet_sortie  =  octet_sortie | msk_led2;
  }
Else
   {
        octet_sortie  =  octet_sortie & (~msk_led2);
        octet_sortie  =  octet_sortie ^  msk_led3;
    }
```

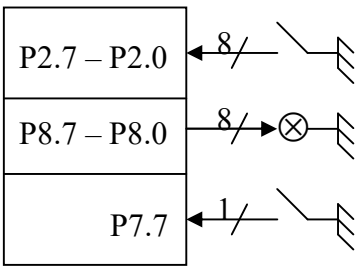
Exemples d'application

Exemple 1

On veut écrire un programme qui recopie l'état de 8 interrupteurs connectés sur les bits 7 à 0 du port P2 sur les 8leds branchés sur le port P8 jusqu'à ce que P7.7 soit égal à 0

a) Donner l'organigramme basé composant de ce programme
b) Traduire cet organigramme en langage C.

Dessin du problème

Les interrupteurs doivent être lus par le µC. Donc ils sont connectés à des entrées Les leds doivent être commandées pour les allumer ou les éteindre. Elles sont connectées sur les sorties	

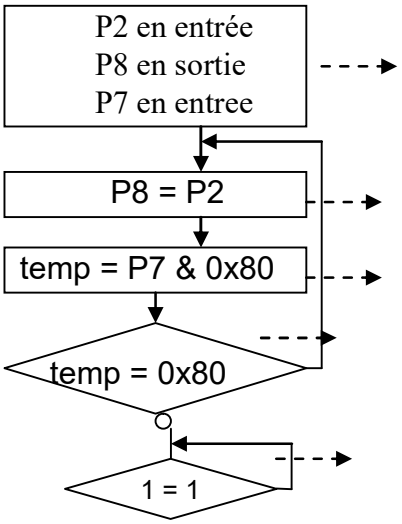
Analyse

Bilan :

P2.7 à P2.0 doivent être programmés en entrée
P8.7 à P8.0 doivent être programmés en sortie
P7.7 doit être programmé en entrée

1- lire l'état des interrupteurs => lecture du port P2 (bit 7 à 0)
2- écrire le résultat sur les leds pour le visualiser => écriture sur le port P8 (bit 7 à 0)
3- recommencer en 1- si P7.7 =0
4- Boucle infinie pour contrôler le µC

Organigramme

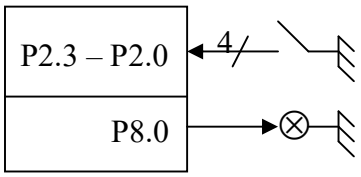
Compléter l'organigramme ci_contre	a) Organigramme
	 
		b) Traduction en langage C
		unsigned char temp;
		
		DP2 = P2 & 0xFF00;
		DP8 = 0xFF;
		DP7 = 0x00; // DP7 = DP7 & 0x7F;
		do
		{ P8 = P2;
		
		Temp = P7 & 0x80;
		}
		while (temp == 0x80);
		
		while (1):

Exemple 2

On veut écrire un programme qui réalise la fonction logique $F = I0.I1 + \overline{I2}.I3 + \overline{I0}.I2$.
F sera visualisée sur la led branchée sur P8.0
I0, I1, I2 et I3 sont interrupteurs connectés respectivement sur les bits P2.0, P2.1, P2.2 et P2.3
Le programme fera ce calcul tout le temps.

Donner l'organigramme basé composant de ce programme
Traduire cet organigramme en langage C.

Dessin du problème

Les interrupteurs doivent être lus par le µC. Donc ils sont connectés à des entrées Les leds doivent être commandées pour les allumer ou les éteindre. Elles sont connectées sur les sorties	

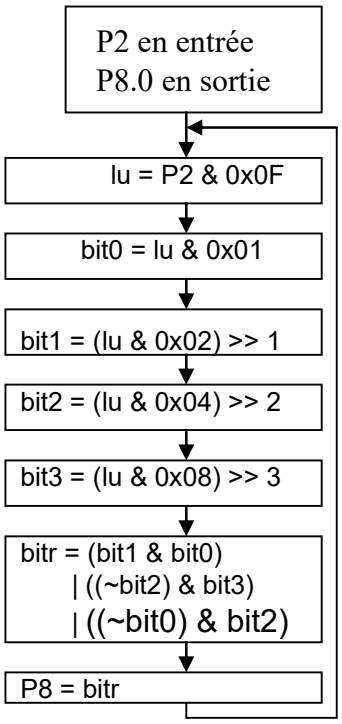
Analyse

Bilan : P2.3 à P2.0 doivent être programmés en entrée
P8.0 doit être programmé en sortie

La fonction à réaliser est une fonction logique combinatoire. On doit donc utiliser les instructions logiques.

Remarque importante :
Le calcul logique se fait bit à bit sans report => il faut amener tous les bits de calcul à la même position binaire par des opérations de décalage. Pour cela, nous allons utiliser des variables pour garder chaque bit à la position 0

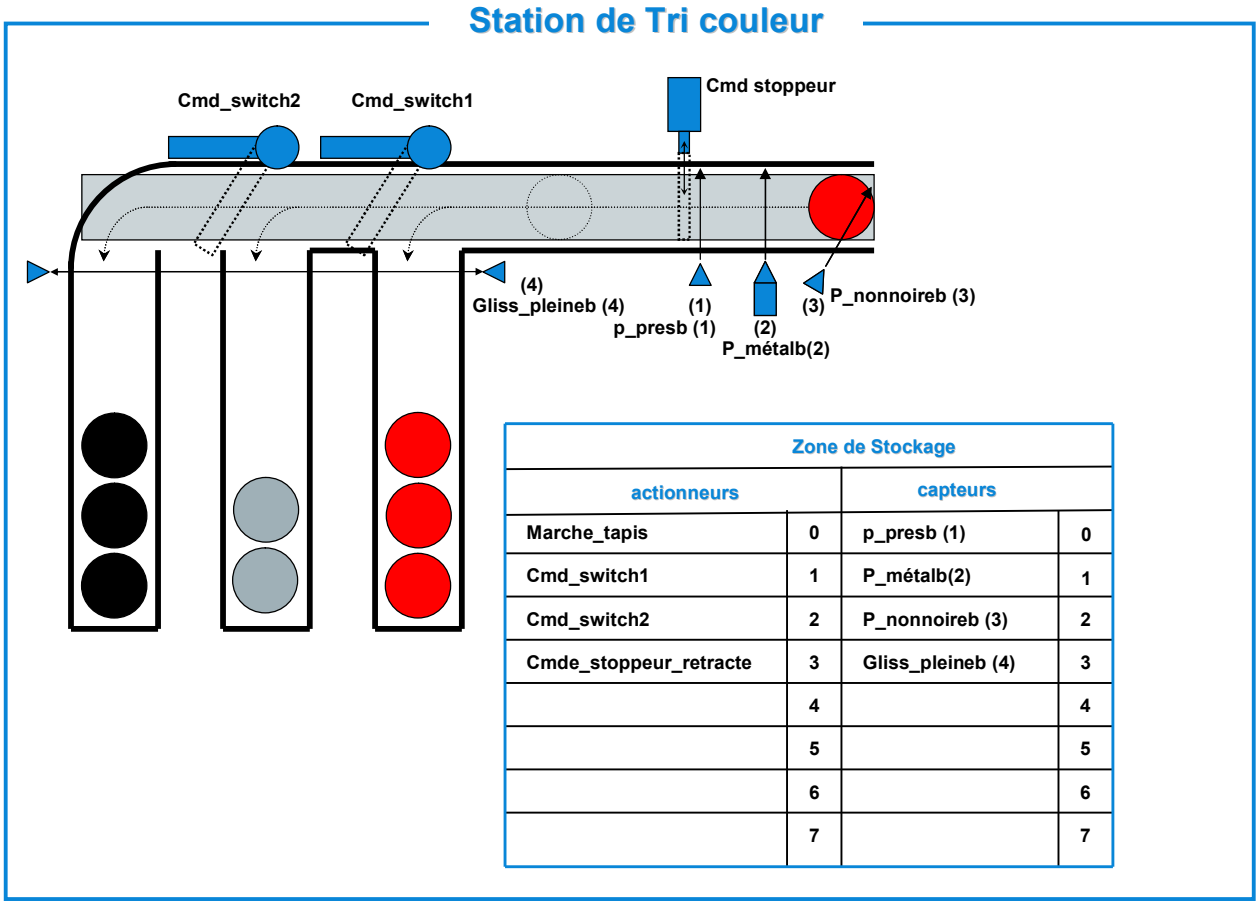
1- lire le port P2 et ne conserver que les bits 3, 2, 1, 0	lu = 0 0 0 0 I3 I2 I1 I0
2- bit0 = garder le bit 0	bit0 = 0 0 0 0 0 0 0 I0
2- bit1 = amener le bit1 à la position 0	bit1 = 0 0 0 0 0 0 0 I1
3- bit2 = amener le bit2 à la position 0	bit2 = 0 0 0 0 0 0 0 I2
4- bit3 = amener le bit3 à la position 0	bit3 = 0 0 0 0 0 0 0 I3
5- Calcul de la fonction logique bitr = (bit1 & bit0) | ((~bit2) & bit3) | ((~bit0) & bit2)	

Organigramme	Traduction en langage C
	

Exemple 3 Gestion de tri de pièces par couleur sur un tapis roulant

Remarque:

Cet exercice sera traité en TD et en TP.



Zone de Stockage			
actionneurs		capteurs	
Marche_tapis	0	p_presb (1)	0
Cmd_switch1	1	P_métalb(2)	1
Cmd_switch2	2	P_nonnoireb (3)	2
Cmde_stoppeur_retracte	3	Gliss_pleineb (4)	3
	4		4
	5		5
	6		6
	7		7

Les capteurs sont connectés sur le port P2. **(Tous les capteurs sont actifs à 0)**

Les actionneurs sont connectés sur le port P8. **(Tous les actionneurs sont actifs à 1)**

Nous allons mettre en œuvre un programme qui range les pièces de couleur dans des glissières suivant une politique donnée.

Fonctionnement des capteurs

p_nonnoireb permet de détecter uniquement les pièces rouges ou métalliques. Ce capteur ne voit pas les pièces noires.

p_metalb ne détecte que les pièces métalliques

p_presb détecte toutes les pièces (rouge, métallique et noire)

Dans cet exercice, on veut stocker les pièces de même couleur dans la même glissière, par exemple les pièces rouges sont stockés dans la première glissière, les métalliques dans la deuxième et les moires dans la troisième glissière.

Pour stocker la pièce rouge dans la première glissière, il suffit de mettre le tapis en marche, de commander le switch1 pour que la pièce soit dirigée vers la première glissière. On détecte que la pièce est bien entrée dans la glissière en testant le capteur gliss_pleineb à 0.

On procède de même pour la pièce métallique en commandant le switch2.

On procède de même pour la pièce noire mais on n'a pas de switch à commander

a) Donner l'organigramme structuré de la fonction, rangement_dans_glissiere qui range une pièce dans une glissière si on considère qu'en entrée type_piece est 0 pour la pièce rouge, 1 pour la pièce métallique et 2 pour la pièce noire.	→ uc type_piece rangement_dans_glissière

Analyse

Traitement de chaque pièce :

Rouge (type_piece = 0) :

tapis en marche, stoppeur_retracté, switch1 activé => port_sortie = 0b00001011= 0x0B

Attendre que la pièce tombe dans la glissière => masque_isole = 0x08
masque_attendu = 0x00

métallique (type_piece = 0):

tapis en marche, stoppeur_retracté, switch2 activé => port_sortie = 0b00001101= 0x0D

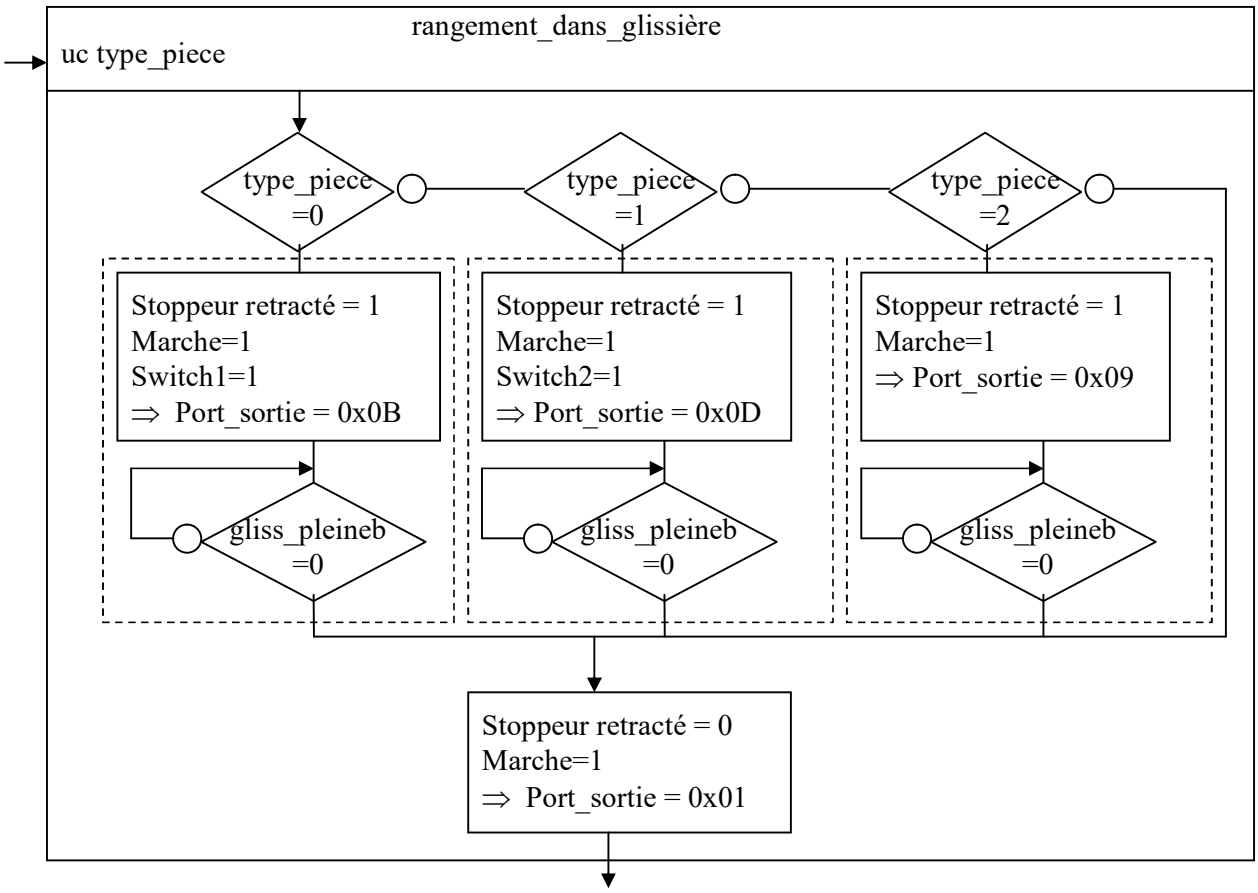
Attendre que la pièce tombe dans la glissière => masque_isole = 0x08
masque_attendu = 0x00

noire (type_piece = 0):

tapis en marche, stoppeur_retracté => port_sortie = 0b00001001= 0x09

Attendre que la pièce tombe dans la glissière => masque_isole = 0x08
masque_attendu = 0x00

désactiver stoppeur_retracté et switch1 => port_sortie = 0b00000001= 0x01



Organigramme de la fonction

Traduction en langage C

début traduction en langage C

```
void rangement_dans_glissiere
                    (unsigned char type_piece)
{
  switch (type_piece & 0x03)
    {  case  0x00 : //rouge
                    P8 = 0x0B;
                    // attente passage pièce dans
                    // la glissière
                    whille ( (P2 & 0x08)== 0x08);
                     break;
       case  0x01 : //métallique
                    P8 = 0x0D;
                    // attente passage pièce dans
                    // la glissière
                    whille ( (P2 & 0x08)== 0x08);
                     break;
       case  0x02 : //noire
                    P8 = 0x0D;
                    // attente passage pièce dans
                    // la glissière
                    whille ( (P2 & 0x08)== 0x08);
                     break;
    }
    P8 = 0x01;
 }
```

Suite traduction en langage C

b) Donner l'organigramme de la fonction qui identifie la couleur de la pièce placée sur le tapis

Analyse
Pour faire cette détection, analysons comment chaque couleur est détectée.

Rappel :

Fonctionnement des capteurs

p_nonnoireb permet de détecter uniquement les pièces rouges ou métalliques. Ce capteur ne voit pas les pièces noires.

p_metalb ne détecte que les pièces métalliques

p_presb détecte toutes les pièces (rouge, métallique et noire)

Position des capteurs

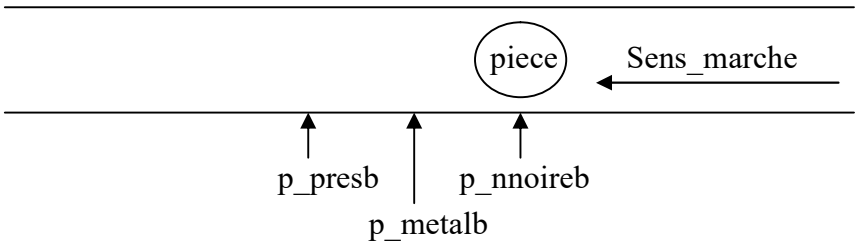


Tableau de détection des pièces

	Capteur p_nnoireb	Capteur p_metalb	Capteur p_presb
Pièce rouge	déctectée		déctectée
Pièce métallique	déctectée	déctectée	
Pièce noire			déctectée

Première conclusion

a) pour détecter la pièce rouge, il faut p_nonnoireb (actif) puis p_presb (actif)

b) pour détecter la pièce métallique, il faut p_nonnoireb (actif) puis p_metalb (actif)

c) pour détecter la pièce rouge, il faut uniquement p_presb (actif)

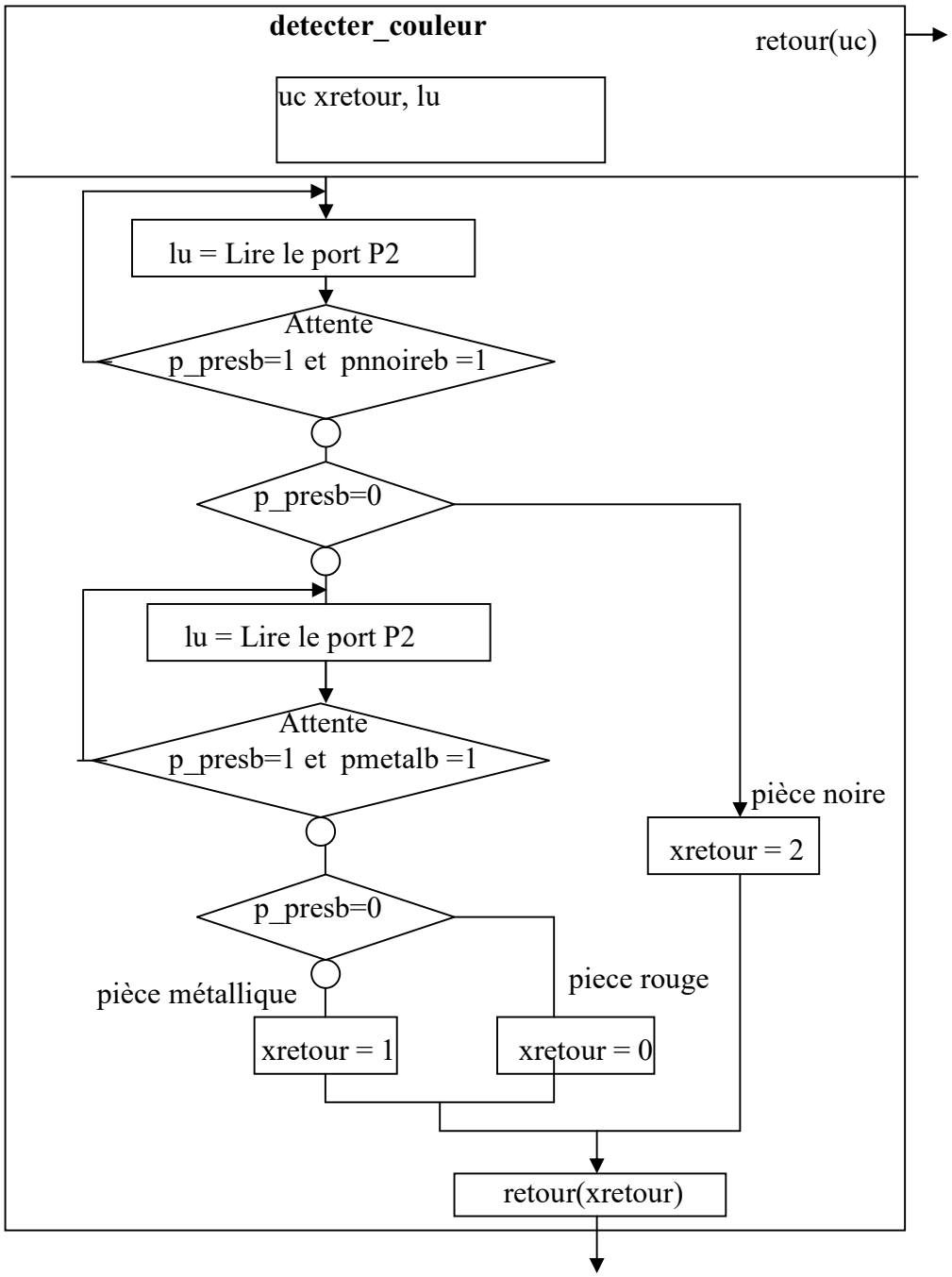
Deuxième conclusion

Lorsqu'une pièce arrive en début de tapis, elle est soit nonnoire soit noire
Donc il faut surveiller les deux capteurs (p_nnoireb et p_presb)

on attend jusqu'à ce qu'un de ces capteurs soit actif
⇒ attendre tant qu'aucune des deux couleurs n'est détectée
si c'est **p_presb** qui est actif
alors c'est une pièce noire (piece = 2)
sinon c'est une pièce rouge ou métallique
on surveille maintenant p_metalb et p_presb
on attend jusqu'à ce que l'un de ces capteurs soit actif
⇒ attendre tant qu'aucune des deux couleurs n'est détectée
si c'est **p_presb** alors la pièce est rouge (piece =0)
sinon la pièce est métallique (piece = 1)
fsi

fsi

Organigramme



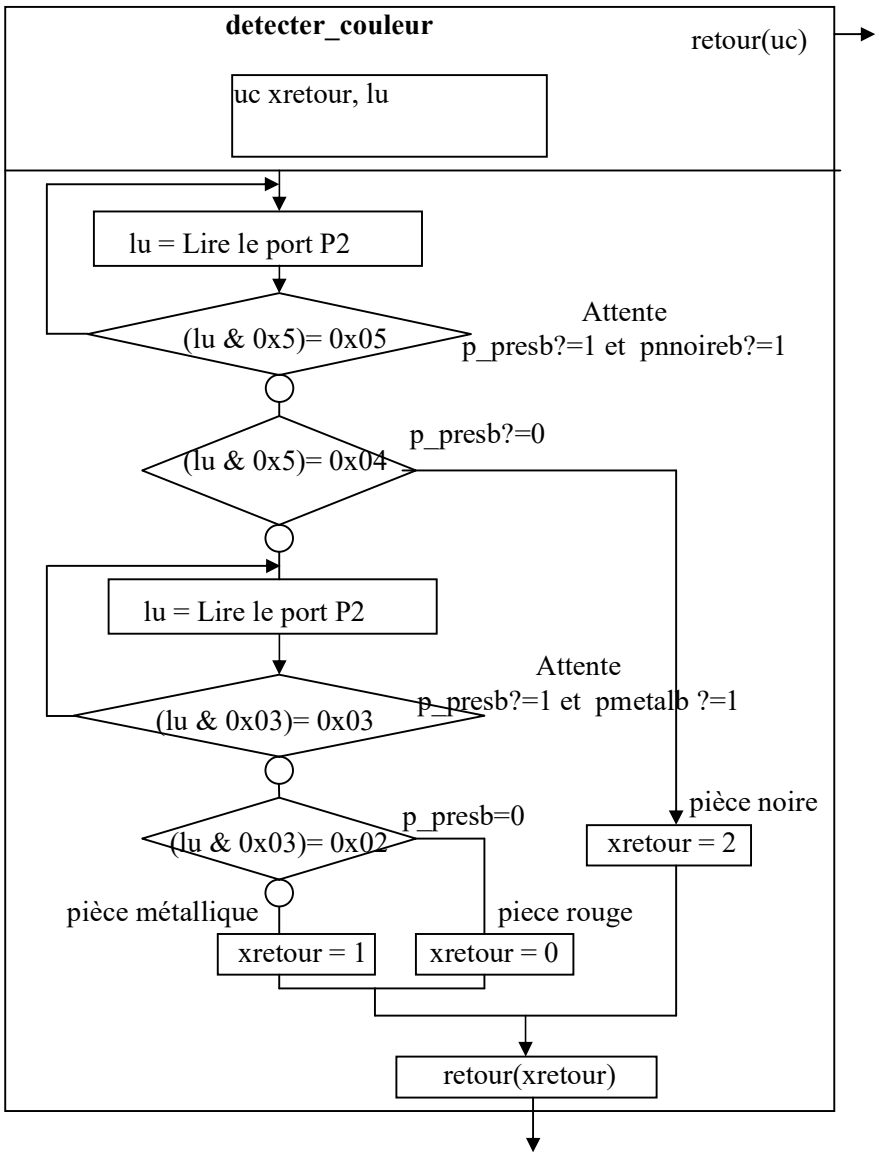
Traduction en langage C

Il faut faire des masques et utiliser les instructions booléennes pour tester les bits

Exemple attendre p_presb =1 et p_nnoiren=1 revient à :

1- lire le port P2
2- faire le masque avec le masque d'isolement de p_presb et p_nnoireb
3- recommencer en 1- si le résultat = masque d'isolement de p_resb et p_nnoireb
⇒ aucun des deux capteurs n'est actif

L'organigramme détaillé devient :



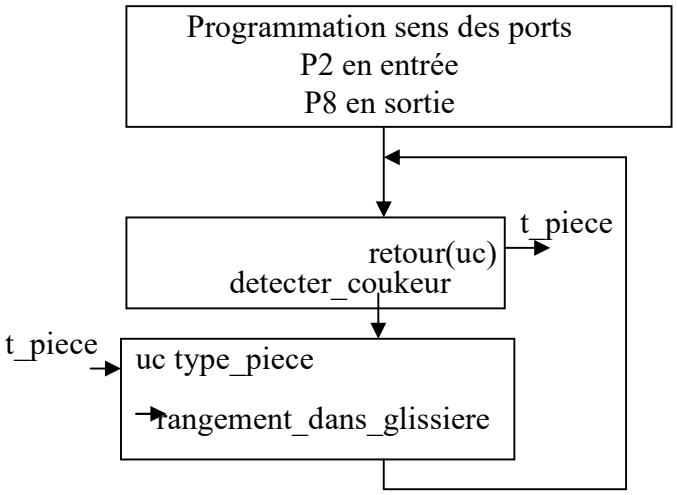
Traduction en langage C

```
unsigned char   detecter_couleur(void)
{ unsigned char lu, xretour;
  do
    lu = P2;
   while (  (lu & 0x05) == 0x05);
   if ((lu & 0x0x5) == 0x04)
          xretour = 0x02;
   else
     {    do
            lu = P2;
          while (  (lu & 0x03) == 0x03);
           if ((lu & 0x0x03) == 0x02)
                  xretour = 0x00;
           else
                  xretour = 0x01;
     }
    return(xretour);
}
```

c) Programme principal

Le programme principal est simple, c'est l'appel dans le bon ordre des deux fonctions précédentes.

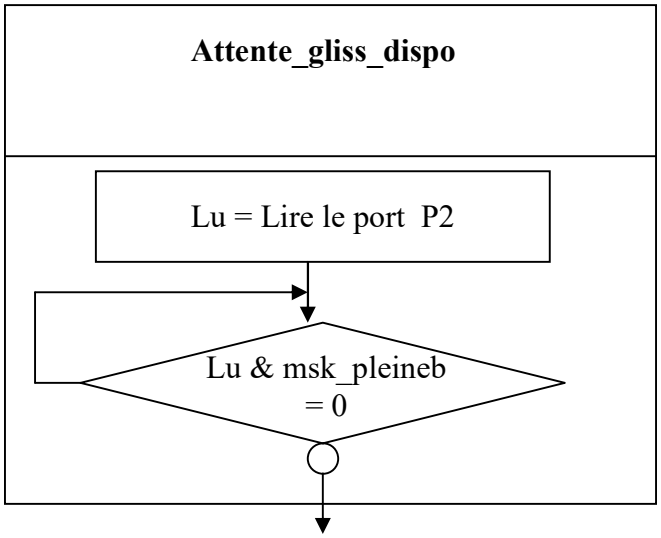
Organigramme



Codage en langage C

```
void main(void)
{
  unsigned char t_piece;
  DP2 = 0x0000;
  DP8 = 0x0FF;
do
  {
     t_piece = detecter_couleur();
     rangement_dans_glissiere(t_piece);
   }
 while(1);
  }
```

d) On souhaite compléter le programme principal précédent pour que le tri s'arrête si une des glissières est pleine. Pour cela, on va écrire une fonction, attente_gliss_dispo, qui va attendre que le capteur gliss_pleineb ne soit pas actif.



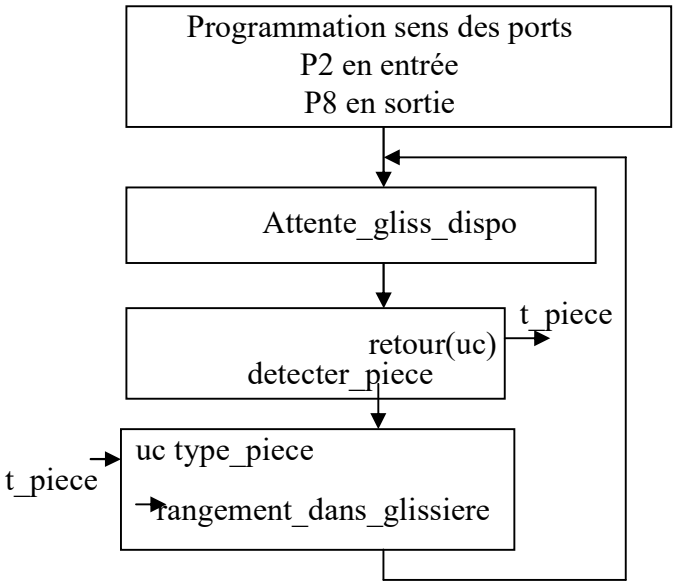
Codage en langage C

```
void attente_gliss_dispo(void)
{  unsigned char lu;
   do
      lu = P2;
    while ( (lu & 0x08) == 0x00 );
}
```

L'organigramme du programme principal devient

Le programme principal est simple, c'est l'appel dans le bon ordre des 3 fonctions précédentes.

Organigramme

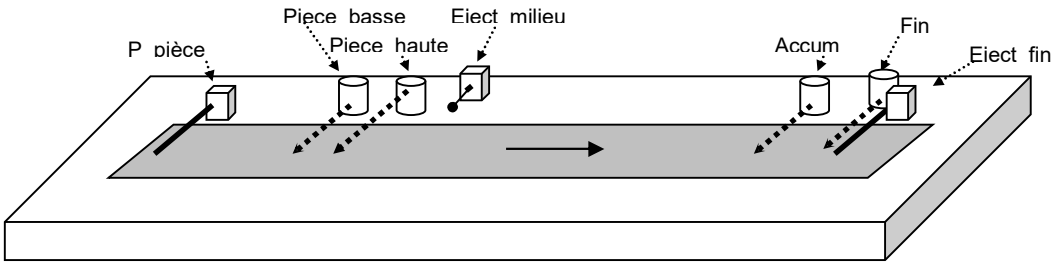


Codage en langage C

```
void main(void)
{
  unsigned char t_piece;
  DP2 = 0x0000;
  DP8 = 0x0FF;
do
  {
    attente_gliss_dispo():
     t_piece = detecter_couleur();
     rangement_dans_glissiere(t_piece);
  }
 while(1);
  }
```

Exemple 4 Gestion de tri de pièces sur un tapis roulant

Nous disposons d'un système constitué :
- d'un ensemble de capteurs
- d'un tapis roulant équipé des éjecteurs pour le tri des pièces



Le tapis roulant permet d'effectuer le tri entre des pièces hautes et des pièces basses. Le tapis est mû par un moteur qui dispose de 2 vitesses (grande ou petite).

Ces éléments se déclinent en entrées-sorties suivantes :

→ **des actionneurs (actifs à 1) : (sorties)**	→ **des capteurs (actifs à 0) : (entrées)**
- marche (commande de mise en marche du tapis petite vitesse) **- GV** (permet de passer en mode grande vitesse avec marche active) **- eject_fin** (pour éjecter une pièce en fin de tapis). Pour éjecter une pièce, on doit maintenir la commande jusqu'à ce le capteur indique que la pièce a été éjectée	**- P_pieceb** (détecte la présence d'une pièce en début de tapis), **- P_basseb** (détecte tout type de pièces), **- P_hauteb** (détecte seulement les pièces hautes), - **Fin_bdeb** (détecte la pièce pour l'éjection en fin de bande).

1 port pour la commande du tapis

Entrées (capteurs)			Sorties	
	Connexions sur la maquette Port d'entrée			**Connexions sur la maquette Port de sortie**
P_pieceb	P2.0			
P_basseb	P2.1			
P_hauteb	P2.2		Marche	P8.2
présenceb	P2.3		Grde_vitesse	P8.3
Fin_bdeb	P2.5		Eject_fin	P8.5

Cahier des charges
Dans cet exercice, le tapis est alimenté manuellement en pièces à trier.

On souhaite réaliser des opérations de tri de pièces en les éjectant en fin de tapis
On ne traite qu'une pièce la fois et toutes les pièces sont éjectées à la fin.
Le tapis est mis en marche dès qu'une pièce est détectée au début du tapis.
Toutes les pièces sont éjectées en fin de tapis.

Par ailleurs,
- on arrête le tapis s'il n'y a pas de pièce à traiter ou en cours de traitement
- on compte le nombre total de pièces traitées
- on compte le nombre de pièces basses
- on compte le nombre de pièces hautes

Chaque compteur est visualisé sur la console dès que le compteur est modifié
PT = toutes les pièces PH = pièces hautes PB = pièces basses

a) Donner la vue externe de la fonction qui permet de commander le tapis et de trier. b) Donner l'organigramme de la fonction, gestion_tapis_hauteur, qui permet de commander le tapis, de trier ces pièces. c) Traduire l'organigramme structuré basé composants en langage C	

Analyse

Les capteurs utiles sont :
p_pieceb qui détecte l'arrivée de toutes les pièces au début du tapis
p_hauteb qui détecte les pièces hautes
fin_bdeb qui détecte l'arrivée des pièces en fin de tapis

Remarque :
Le capteur p_basseb n'est pas utile dans cet exercice car toutes les pièces sont éjectées en fin de tapis et par ailleurs, il voit aussi bien les pièces hautes que basses.
P_presence n'est pas non plus utile..

Analysons la séquence pour chaque type de pièce :

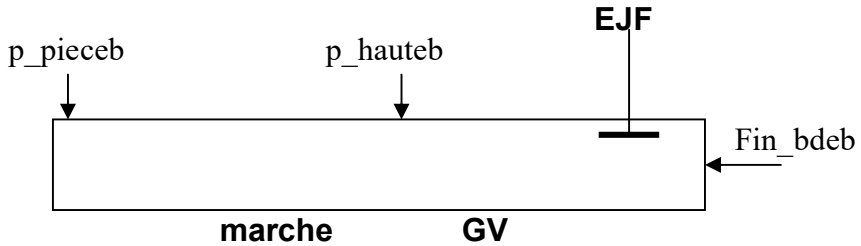
a) Piéce haute
elle est détectée en début de tapis puis par le capteur p_hauteb et par le capteur finbdeb puis éjectée

b) Piéce basse
elle est détectée en début de tapis puis par le capteur finbdeb puis éjectée.

Conclusion

La pièce basse ne sera reconnue que lorsqu'elle atteint le capteur finbdeb.

La pièce haute est détectée par le capteur p_hauteb.

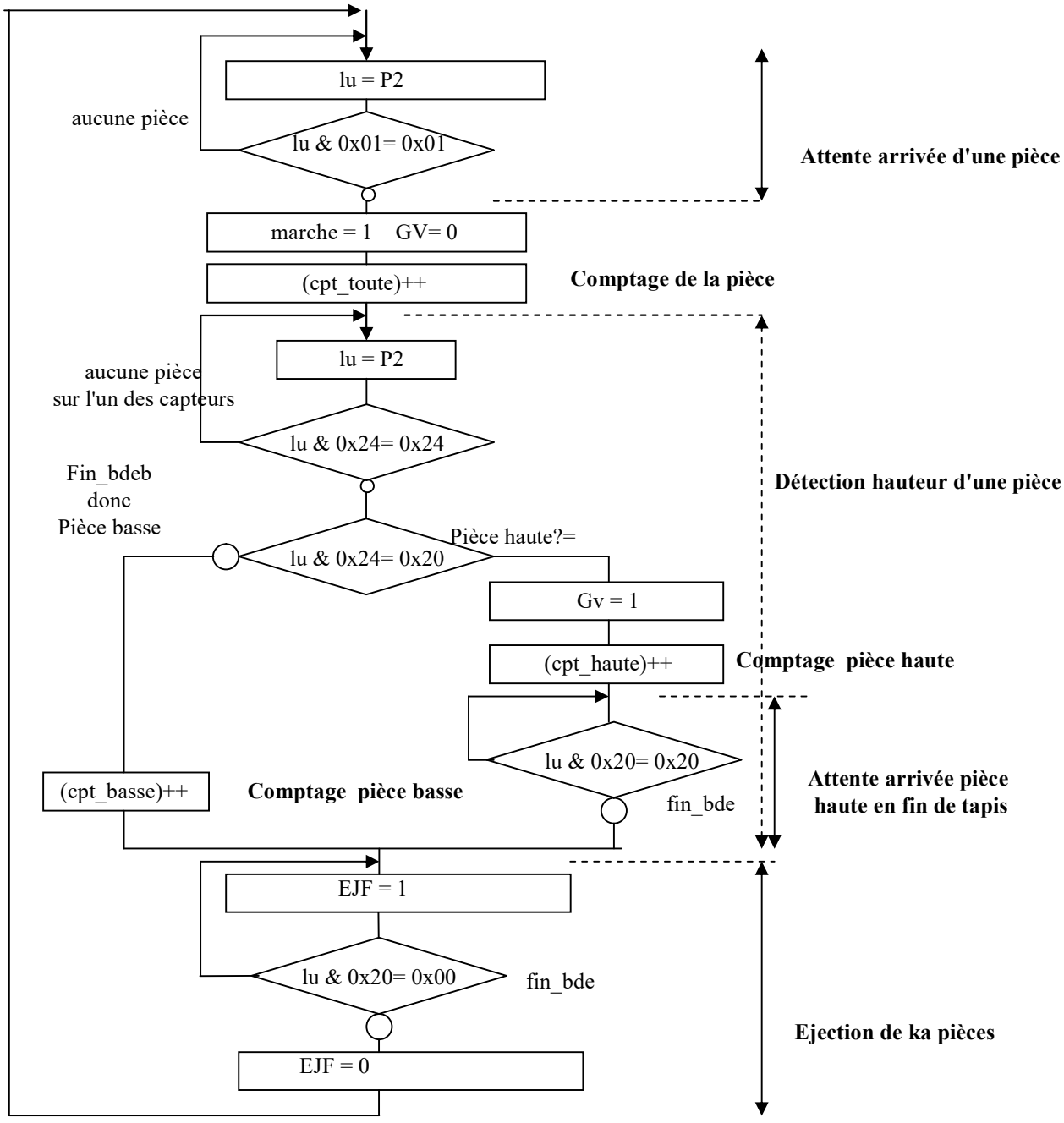


On doit donc

☞ surveiller p_pieceb pour détecter l'arrivée des pièces

☞ surveiller en même temps p_hauteb et fin_bdeb pour détecter les pièces hautes et les pièces basses

Organigramme



Traduction en langage C