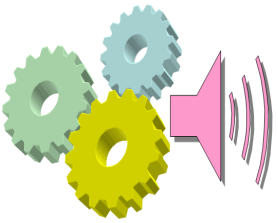
ROYAUME DU MAROC

MINISTERE DE L'ÉDUCATION NATIONALE

Académie de Casablanca Settat

Direction Provinciale de Mohammedia



Nom : ……

Prénom : ……

Classe : **2STE**…

Lycée Qualifiant Technique Mohammedia

Sciences de l'ingénieur

Système n°5 :

Parking Automatique

Sciences et Technologies Électriques Niveau 2

Professeur : **MAHBAB**

المملكة المغربية وزارة التربية الوطنية

الثانوية التأهيلية التقنية المحمدية

Le dossier comporte au total 50 pages :

Sujet : Parking Automatique

☞ Le sujet comporte au total 21 pages.

☞ Le sujet comporte 3 types de documents :

- **Pages 01 à 05** : Socle du sujet comportant les situations d'évaluation (SEV) ;
- **Pages 06 à 12** : Documents ressources portant la mention **DRES XX**
- **Pages 13 à 21** : Documents réponses portant la mention **DREP XX**

21 pages

Unité A.T.C

Fiches cours :

- Fiche cours n°19 : **Le GRAFCET**
- Fiche cours n°20 : **EEPROM du 16 F 84**
- Fiche cours n°21 : **TIMER0 du 16 F 84**
- Fiche cours n°22 : **Programmation des PLD**

12 pages

Activités :

- Activité n°12 : **Partition d'un GRAFCET et notion de tâche**
- Activité n°13 : **PLD 'les fonctions combinatoires'**
- Activité n°14 : **PLD 'les fonctions séquentielles'**

15 pages

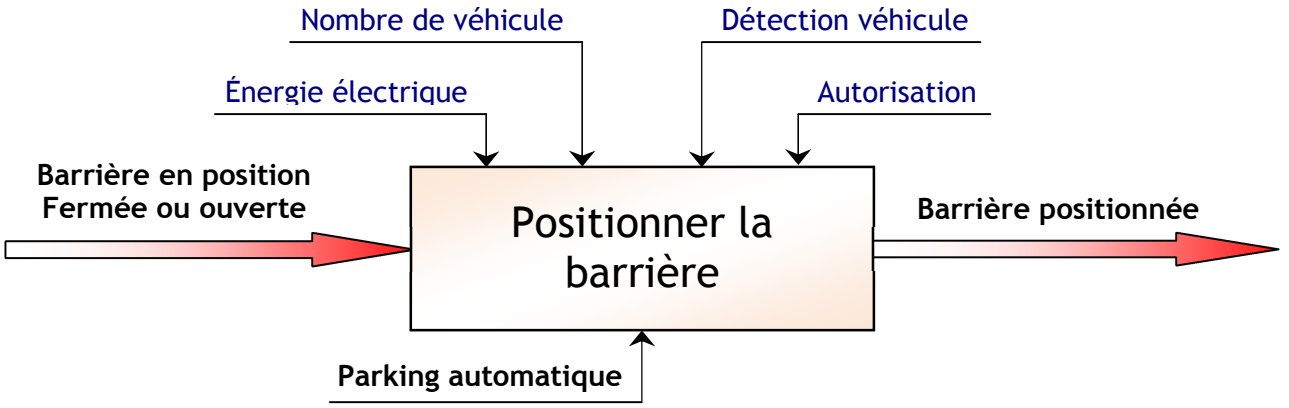
PARKING AUTOMATIQUE

1. PRÉSENTATION DU SYSTÈME :

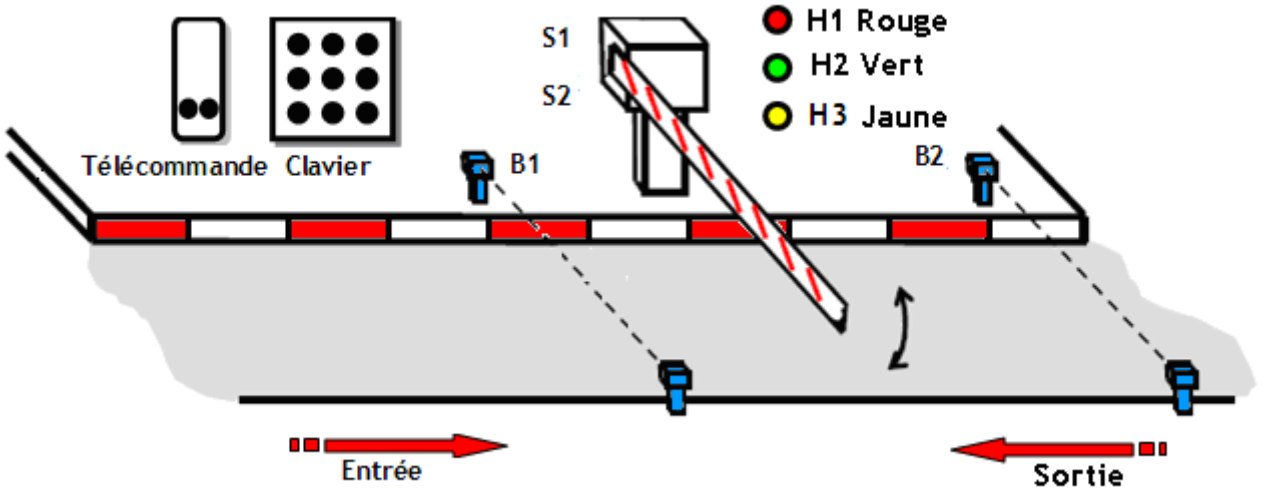
On se propose d'étudier le fonctionnement d'une barrière de parking de stationnement de véhicule.

La barrière fonctionne en tenant compte de diverses informations suivantes :

- Présence d'une voiture devant la barrière (entrée ou sortie) ;
- Autorisation d'ouverture de la barrière (clavier ou télécommande) si entrée, sans autorisation si sortie ;
- Nombre de véhicule dans le parking.



2. DESCRIPTION DU SYSTÈME :



Le système est composé de :

- Une **barrière,** commandée par un moteur asynchrone ;
- Deux cellules photoélectriques B_1 et B_2 associées à deux réflecteurs qui détectent les véhicules en entrée et en sortie du parking ;
- Deux capteurs à galet S_1 et S_2 détectant la position de la barrière (positions haute et basse) ;
- Un moteur asynchrone triphasé (**MAS**) associé à un réducteur mécanique ;
- Un **variateur de vitesse** pour moteur triphasé ;
- Une carte électronique à base du microcontrôleur **16 F 84** ;

- Une **alimentation modulaire** régulée (intégrée dans le boîtier);
- Un **clavier** et une **télécommande** permettent de délivrer les consignes de fonctionnement (autorisation d'entrée des véhicules) :
  - L'autorisation issue du clavier (U_1) est délivrée si le code fourni est correct ;
  - L'autorisation issue de la télécommande (U_2) est délivrée lorsqu' on appuie sur le bouton gauche de la télécommande.
- Trois voyants permettent d'afficher des messages à destination des utilisateurs du système. Ces trois Voyants permettent de délivrer les messages suivants :
  - H_1 'Voyant rouge' : Accès non autorisé au parking ;
  - H_2 'Voyant vert' : Accès autorisé au parking ;
  - H_3 'Voyant jaune' : Parking complet.

3. FONCTIONNEMENT DÉSIRÉ :

3.1. Entrée d'un véhicule dans le parking :

Les étapes de fonctionnement sont les suivantes :

- Présence d'un véhicule devant l'entrée de la barrière détectée par le capteur B_1 ;
- Demande d'autorisation d'ouverture à l'aide de la **télécommande** ou du **clavier** ;
- Si le parking n'est pas complet (**nombre** de véhicule dans le parking **< 50** 'voyant H_3 éteint'), la barrière s'ouvre par fermeture du relais KM_1 et le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Ouverture complète de la barrière, détectée par l'action sur S_1, le voyant H_2 ' Accès autorisé au parking' s'allume et le véhicule peut entrer dans le parking en franchissant la barrière ;
- Entrée du véhicule ;
- Entrée complète du véhicule dans le parking détectée par le capteur B_2 ;
- Le nombre de véhicule présent dans le parking est alors **incrémenté** de **1** ;
- Fermeture de la barrière par fermeture du relais KM_2, après une temporisation de **10 s** ;
- Pendant la fermeture de la barrière, le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Fermeture complète de la barrière, détectée par l'action sur S_2 ;

3.2. Sortie d'un véhicule dans du parking :

Les étapes de fonctionnement sont les suivantes :

- Présence d'un véhicule devant la sortie de la barrière détectée par le capteur B_2 ;
- Ouverture de la barrière s'ouvre par fermeture du relais KM_1 et le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Ouverture complète de la barrière, détectée par l'action sur S_1, le voyant H_2 ' Accès autorisé au parking' s'allume et le véhicule peut sortir du parking en franchissant la barrière ;
- Sortie du véhicule ;
- Sortie complète du véhicule du parking détectée par le capteur B_1 ;
- Le nombre de véhicule présent dans le parking est alors **décrémenté** de **1** ;
- Fermeture de la barrière par fermeture du relais KM_2, après une temporisation de **10 s** ;
- Pendant la fermeture de la barrière, le voyant H_1 ' Accès non autorisé au parking' s'allume ;
- Fermeture complète de la barrière, détectée par l'action sur S_2 ;

4. DESCRIPTION DE LA PARTIE COMMANDE :

La commande de la barrière est réalisée par une carte électronique à base d'un microcontrôleur de type PIC **16F84** dont les caractéristiques sont données au **DRES 01** page 06.

On suppose que le nombre de véhicules présents dans le parking est stocké dans une case mémoire de L'EEPROM (adresse 02_H).

On dispose des 5 sous programmes suivants :

- **Lecture_EEPROM :** La donnée se trouvant dans l'EEPROM à l'adresse 02_H, sera enregistrée dans la RAM à l'adresse $0C_H$ (soit Cp_1) ;
- **Ecriture_EEPROM :** La donnée se trouvant dans la RAM à l'adresse $0C_H$ (soit Cp_1) sera enregistrée dans l'EEPROM (adresse 02_H) ;
- **TEMPO :** Un sous programme de temporisation de 10 s ;

N.B :

- Ce sous programme de temporisation, utilise le TIMER0 pour compter un temps T_1 de **10 s**.
- Les caractéristiques du TIMER0 et l'organigramme du sous programme **Tempo** sont données au **DRES 03, 04** pages 08, 09.

- **Ouvrir_Barrière :** Ce sous programme permet d'ouvrir la barrière et d'allumer le voyant vert (pendant l'ouverture le voyant rouge est allumé) ;
- **Fermer_Barrière :** Ce sous programme permet de fermer la barrière (pendant la fermeture le voyant rouge est allumé).

On utilise ces sous programmes pour structurer le programme de fonctionnement du parking. Le nombre de véhicule présent dans le parking ; est stocké dans la ROM électrique du PIC **16 F 84** ; à fin d'éviter la perte de cette information, lorsqu'il y a coupure du courant dans le parking.

5. TABLEAU D'AFFECTATION DES ENTRÉES/SORTIES :

Mouvement	Actionneur	Préact.	Ordres	Sortie PIC
Ouvrir la barrière	Moteur M_1	Relais KM_1	KM_1	RB_0
Fermer la barrière	Moteur M_1	Relais KM_2	KM_2	RB_1

Message	Voyant	Préact.	Ordres	Sortie PIC
Accès non autorisé au parking	Feu Rouge	Relais H_1	H_1	RB_2
Accès autorisé au parking	Feu Vert	Relais H_2	H_2	RB_3
Parking complet	Feu Jaune	Relais H_3	H_3	RB_4

Compte-rendu	Capteur	Mnem.	Entrée PIC
Présence véhicule à l'entrée	Détecteur photoélectrique reflex	B_1	RA_0
Présence véhicule à la sortie	Détecteur photoélectrique reflex	B_2	RA_1
Barrière ouverte	Détecteur mécanique à levier	S_1	RA_2
Barrière fermée	Détecteur mécanique à levier	S_2	RA_3

Consigne	Constituant	Mnem.	Entrée PIC
Code d'entée	Clavier	U_1	RA_4
Impulsion télécommande	Télécommande	U_2	
Compteur de véhicule	Case mémoire d'adresse $0C_H$	Cp_1	-
Temporisation 10 s	Sous programme Tempo	T_1	-

SEV 1 — GRAFCET DU SYSTÈME

RESSOURCES A EXPLOITER : **DRES 01** page 06
et documents pages 01, 02 et 03.

Tâche 1 — GRAFCET DE FONCTIONNEMENT

1. Sur le document **DREP 01** page 13, compléter le GRAFCET global du parking du point de vue partie **opérative** ;

En se référant au GRAFCET du point de vue partie opérative du **DREP 01**, compléter sur le document **DREP 02** page 14 :

2. Le GRAFCET du point de vue partie **commande**, correspondant au GRAFCET global ;
3. Le GRAFCET du point de vue partie commande **codé PIC**, correspondant au GRAFCET global.

Tâche 2 — PATITION DU GRAFCET GLOBALE

On veut réécrire le GRAFCET globale du système d'une manière simplifiée en considérant les séquences 'Entrée d'un véhicule' et 'Sortie d'un véhicule' comme des tâches (sous programme) intitulées Tâche 1 et Tâche 2 ;

En se référant au GRAFCET du point de vue partie commande, compléter sur le document **DREP 03** page 15 :

4. Les GRAFCETS tâche **1** et tâche **2** ;
5. Le GRAFCET **principal**.

SEV 2 — PROGRAMME DE FONCTIONNEMENT

RESSOURCES A EXPLOITER : **DRES 05, 06** pages 10, 11

Tâche 1 — OUVRIR ET FERMER LA BARRIÈRE

RESSOURCES A EXPLOITER : **'DESCRIPTION DE LA PARTIE COMMANDE'** pages 03

Sur le document **DREP 04** page 16, compléter :

6. L'organigramme **Ouvrir_Barrière** ;
7. L'organigramme **Fermer_Barrière** ;
8. Le sous programme **Ouvrir_Barrière** ;
9. Le sous programme **Fermer_Barrière**.

Tâche 2 — UTILISATION DE L'EEPROM

RESSOURCES A EXPLOITER : **DRES 02** page 07
et **'DESCRIPTION DE LA PARTIE COMMANDE'** pages 03.

Sur le document **DRES 02**, on donne la structure du registre **EECON1** et les procédures d'écriture et de lecture de l'EEPROM.

Sur le document **DREP 05** page 17, compléter :

10. Le sous programme **Lecture_EEPROM** ;
11. Le sous programme **Ecriture_EEPROM**.

Tâche 3 — TEMPORISATION AVEC TIMER0

RESSOURCES A EXPLOITER : **DRES 03, 04** page 08, 09

Dans cette partie, on utilise le TIMER0 pour compter un temps T_1 de 10 s.
L'utilisation du TIMER0 est plus pratique et plus simple que l'utilisation des compteurs programmés.

Sur le document **DREP 06** page 18 :

12. Compléter, le sous programme «**Tempo**» correspondant à l'organigramme Tempo du document ressource **DRES 03** ;

13. Calculer la valeur de **N** (donner N en hexadécimal) à charger dans le compteur **Cp_2**, pour avoir **T_1 = 10 s** ;

14. Trouver les mots de **commande** à mettre dans les registres **OPTION**, **TRISA** et **TRISB** ;

N.B :

Pour la configuration du TIMER0 utiliser les documents **DRES 03** et **04** et pour la configuration des PORTA et B utiliser le document **DRES 05.**

15. Compléter alors le programme de **configuration** du microcontrôleur **16F84**.

Tâche 4 — PROGRAMME PRINCIPAL

16. Compléter l'organigramme de fonctionnement normal du parking automatique, sur le document **DREP 07** page 19 ;

17. Compléter le programme principal du parking, sur le document **DREP 08** page 20.

SEV 3 — COMPTAGE/DÉCOMPTAGE ET AFFICHAGE DU NOMBRE DE VÉHICULE

RESSOURCES A EXPLOITER : **DRES 07** page 12

Pour l'affichage et le comptage/décomptage du nombre de véhicule dans le parking, on utilise un compteur/décompteur modulo 64, un décodeur Binaire/BCD, deux décodeurs BCD/7 segments et 2 afficheurs à 7 segments, Voir document **DRES 07**.

Les deux décodeur BCD/7 segments et le compteur/décompteur sont réalisés par trois circuits logiques programmables de type **GAL 16 V 8**.

Tâche — PROGRAMMATION DU GAL 16 V 8

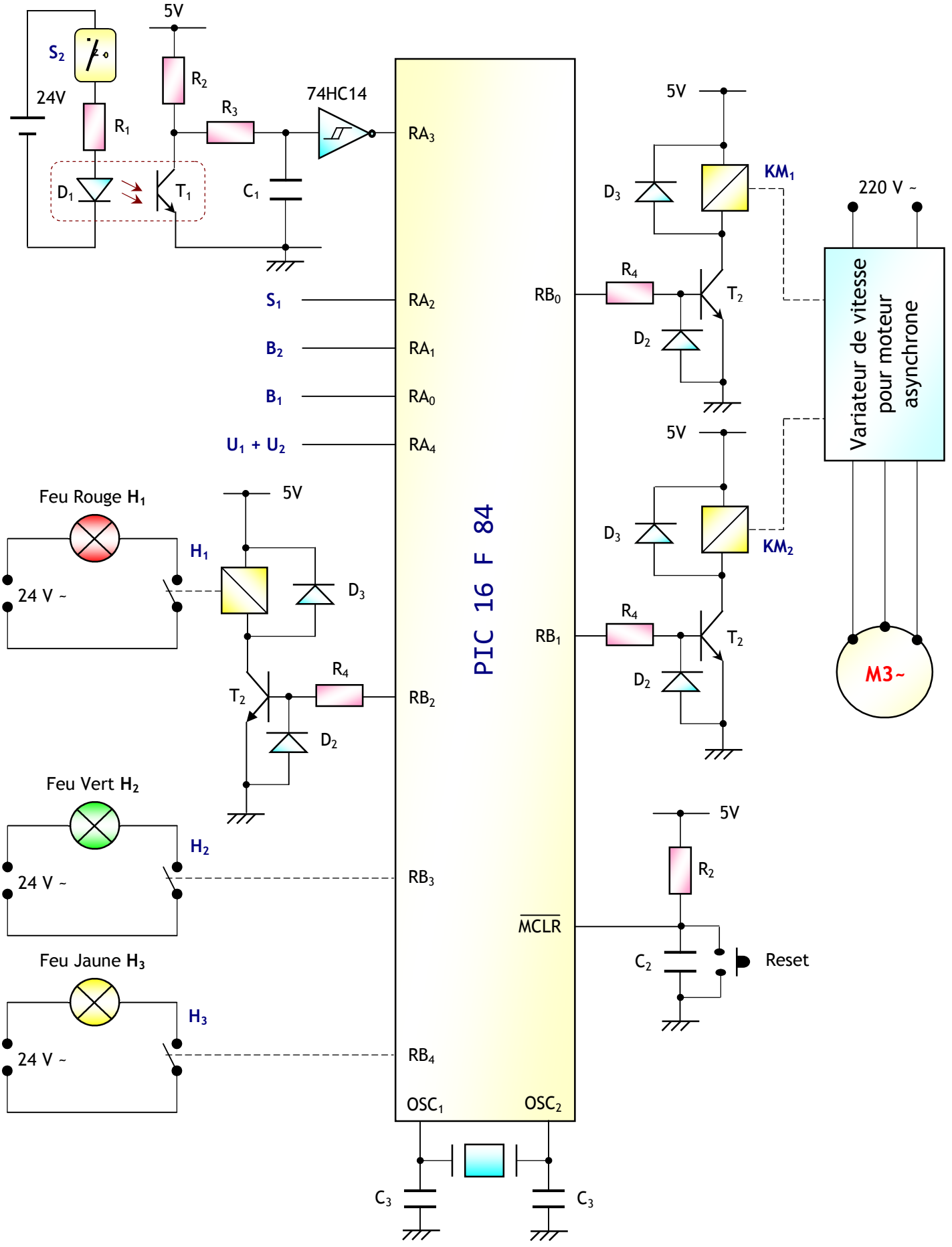
Sur le document **DREP 09** page 21 :

18. Compléter le programme ABEL du décodeur BCD/7 segments, en utilisant une description par équation ;

19. Compléter le programme ABEL du compteur/décompteur modulo 64, en utilisant une description par équation ;

DRES 01

Circuit global de pilotage du système



DRES 02

EEPROM du 16 F 84

1. Structure du registre EECON1 :

7	6	5	4	3	2	1	0
x	x	x	EEIF	WRERR	WREN	WR	RD

Bit 0 : RD - ReaD EEPROM -

Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une lecture de l'EEPROM. Après le cycle de lecture, il est mis automatiquement à 0.

Bit 1 : WR - WRite EEPROM -

Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une écriture de l'EEPROM. Après le cycle d'écriture, il est mis automatiquement à 0.

Bit 2 : WREN - WRite ENABLE EEPROM -

C'est un bit de confirmation d'écriture dans l'EEPROM. En effet, il ne suffit pas de définir un cycle d'écriture uniquement avec le bit WR. Il faut impérativement valider le bit WREN (WREN = 1) pour autoriser une écriture.

Bit 3 : WRERR - EEPROM WRite ERROR flag -

Ce drapeau indique qu'une erreur s'est produite lors d'un cycle d'écriture dans l'EEPROM.

WRERR = 1 : une opération d'écriture a échouée ;

WRERR = 0 : le cycle d'écriture s'est déroulé normalement.

Bit 4 : EEIF - EEPROM Interrupt Flag -

EEIF est un drapeau qui génère une interruption lorsqu'un cycle d'écriture s'est déroulé normalement. Il doit être mis à 0 lors de la routine d'interruption.

EEIF = 1 : l'opération s'est déroulé correctement ;

EEIF = 0 : soit l'opération n'a pas commencé, soit n'est pas terminée.

2. Lecture d'une donnée :

- Placer l'**adresse** de la donnée à lire dans **EEADR** ;
- Mettre le bit **RD** de EECON1 à **1** ;
- Lire le contenu du registre **EEDATA.**

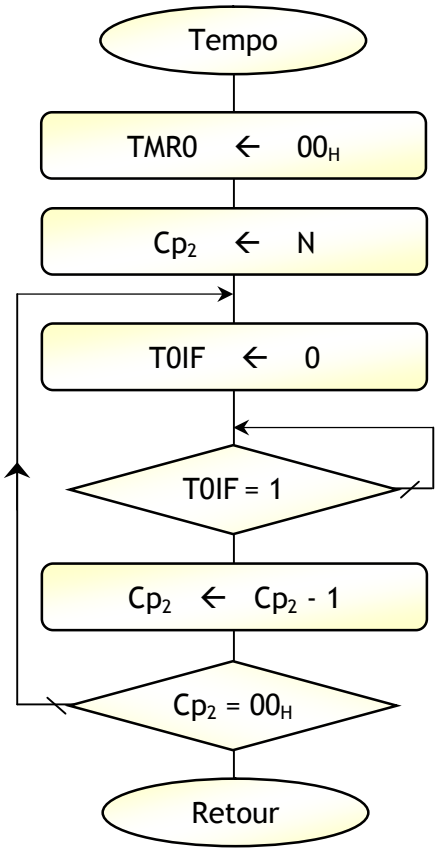
3. Ecriture d'une donnée :

- Placer l'**adresse** de la donnée à écrire dans **EEADR** ;
- Placer la **donnée** à écrire dans **EEDATA** ;
- Mettre le bit **WREN** de EECON1 à **1** pour autoriser l'écriture ;
- Placer **0x55** dans **EECON2** ;
- Placer **0xAA** dans **EECON2** ;
- Mettre le bit **WR** de EECON1 à **1** ;
- Attendre que le bit **EEIF** soit à **1** ;
- On peut utiliser l'interruption produite par **EEIF** en la validant par le bit **EEIE** de **INTCON** ;
- N'oublier pas de remettre **EEIF** à **0.**

DRES 03

TIMER0 et sous programme de temporisation

1. Organigramme de temporisation :



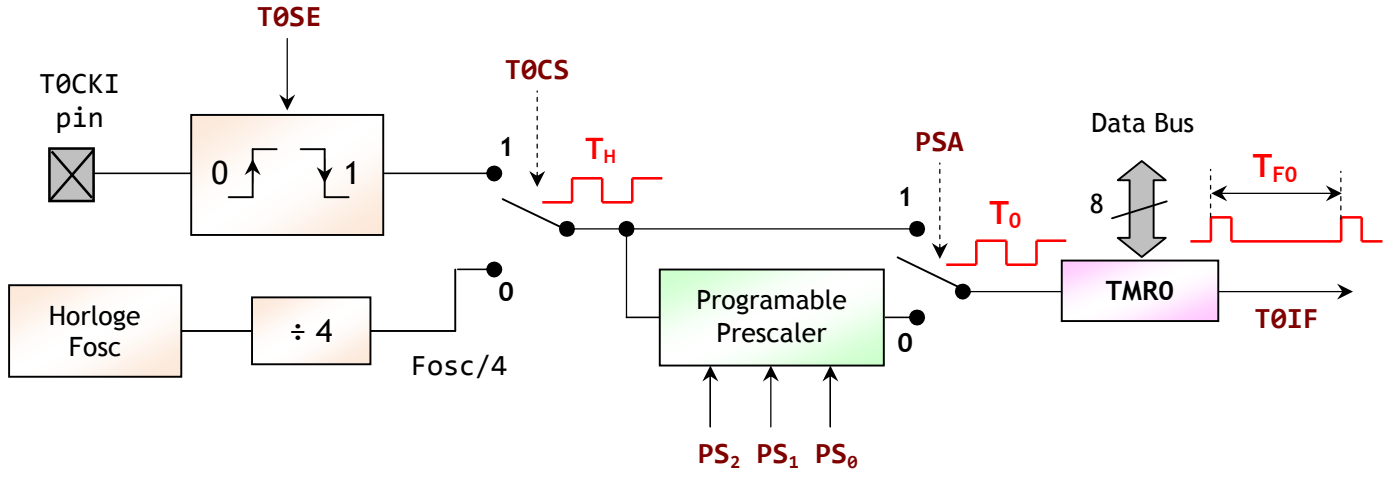
2. Configuration du TIMER0 :

TEMPO est un sous programme de temporisation de 10 s.

Ce sous programme de temporisation, utilise le TIMER 0 pour compter un temps T_1 de 10 s, selon la configuration suivante :

- Le TIMER0 utilise l'horloge interne Fosc (/4) ;
- Fosc = 4 MHz ;
- Rapport de division de l'horloge 256 ;
- Cp_2 : compteur d'adresse 0x0D.

3. Structure simplifiée du TIMER0 :



4. Calcul de la temporisation :

En résumé, chaque fois que le compteur complète un tour, le drapeau **T0IF** se lève.

Si on note T_H la période de l'horloge source, T_0 l'horloge de TMR0 et T_{F0} le temps qui sépare 2 levés de drapeau successifs :

- Sans prédiviseur : $T_{F0} = 256.T_0 = 256.T_H$
- Avec prédiviseur : $T_{F0} = 256.T_0 = 256.(DIV.T_H)$
- Avec prédiviseur et compteur N dans le programme : $T = N.T_{F0} = N.256.DIV.T_H$

DRES 04

Les registres OPTION et INTCON

1. Structure du registre OPTION :

7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS_2	PS_1	PS_0

Bit 7 : RBPU - PORTB Pull-Up -

RBPU = 1 : le "tirage au plus" interne du PORTB est désactivé.

RBPU = 0 : le "tirage au plus" interne du PORTB est activé.

Bit 6 : INTEDG - INTerrupt EDGe -

INTEDG = 1 : la broche RB_0/INT génère une interruption sur un front montant ;

INTEDG = 0 : la broche RB_0/INT génère une interruption sur un front descendant.

Bit 5 : TOCS - TMR0 Clock Source -

Il permet de sélectionner le mode de fonctionnement du TIMER/Compteur :

TOCS = 1 : sélection de l'horloge externe (RA_4) qui correspond au COMPTEUR ;

TOCS = 0 : sélection de l'horloge interne qui correspond au mode TIMER.

Bit 4 : TOSE - TMR0 Source Edge -

Ce bit détermine sur quel front -montant ou descendant- l'entrée RA_4 incrémentera le registre TMRO :

TOSE = 1 : front descendant ;

TOSE = 0 : front montant.

Bit 3 : PSA - PreScaler Assignment -

PSA = 1 : alors le Prescaler est associé avec le WDT ;

PSA = 0 : alors le Prescaler est associé avec le TIMER.

Bit 2, 1, 0 : PS_0, PS_1, PS_2 - Prescaler Select -

Ces trois bits effectuent une division de la fréquence d'horloge du Prescaler :

PS_2	PS_1	PS_0	RATIO TMR0	RATIO WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

2. Structure du registre INTCON :

7	6	5	4	3	2	1	0
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Bit 2 : T0IF - TMR0 Overflow Interrupt Flag -

Ce drapeau indique un dépassement du registre TMR0 (passage de FF à 00) :

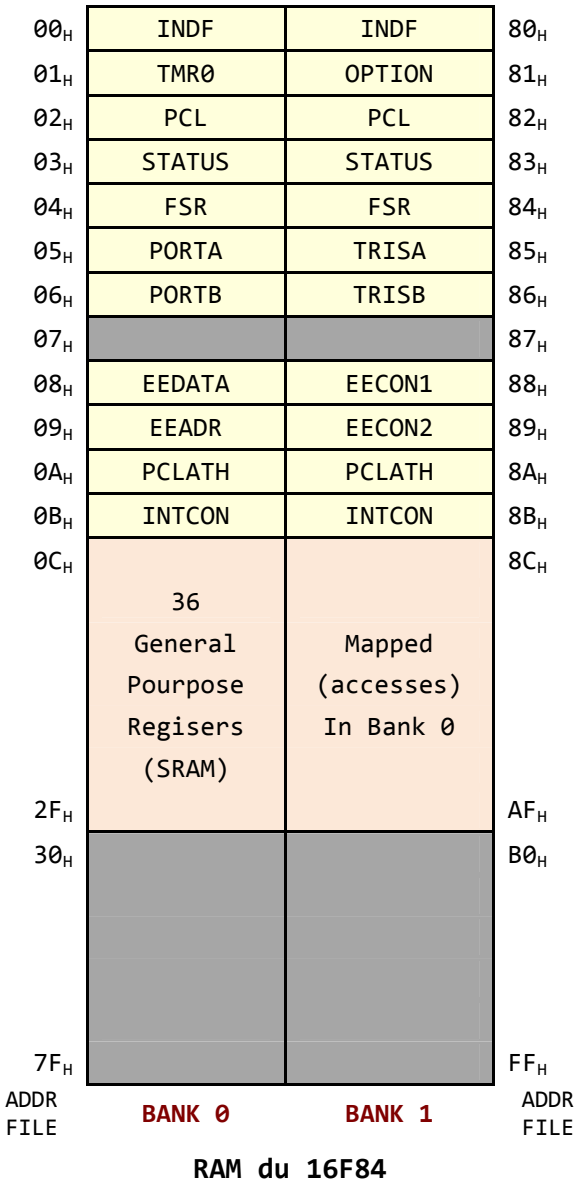
T0IF = 1 : dépassement de TMR0 ;

T0IF = 0 : pas de dépassement.

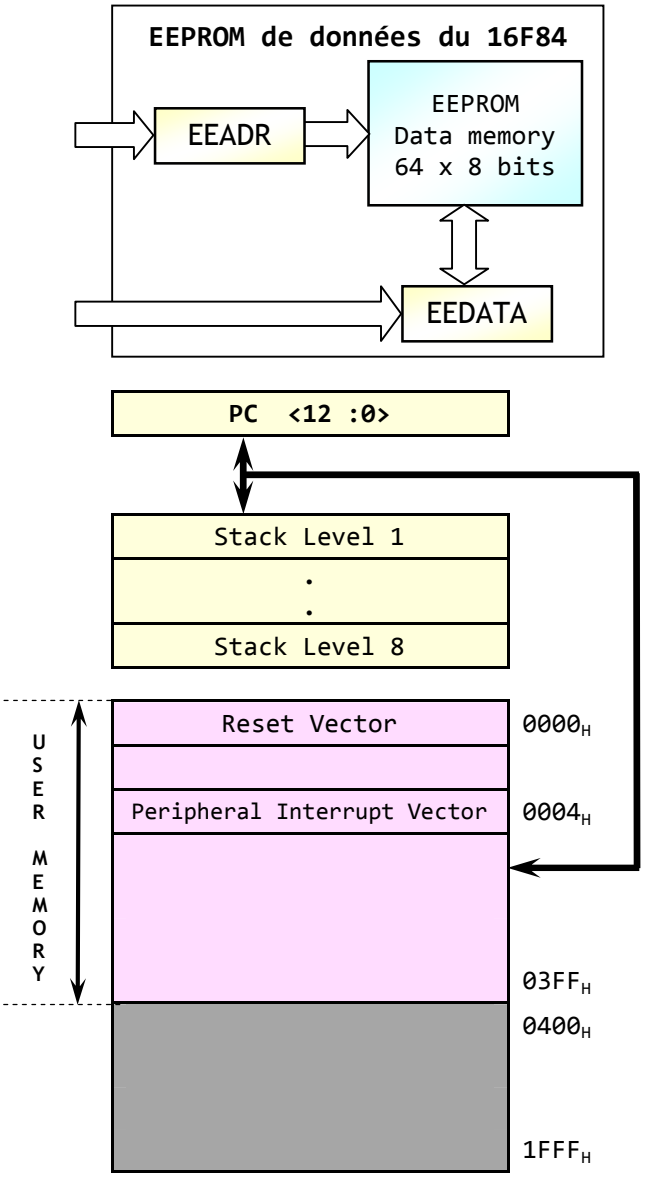
DRES 05

Plan mémoire du 16 F 84

1. Les mémoires du 16 F 84 :



RAM du 16F84



EEPROM programme du 16F84

2. Configuration des PORTS :

Tous les ports sont pilotés par deux registres :

- Le registre de **PORTx**, si le **PORTx** ou certaines lignes de **PORTx** sont configurées en sortie, ce registre détermine l'état logique des sorties.
- Le registre **TRISx**, c'est le registre de direction. Il détermine si le **PORTx** ou certaines lignes de Port sont en entrée ou en sortie. L'écriture d'un **1** logique correspond à une **entrée** (**1** comme **I**nput) et l'écriture d'un **0** logique correspond à une sortie (**0** comme **O**utput).

Remarque :

- Les registres **TRISx** appartiennent à la **BANQUE 1** des **SFR**. Lors de l'initialisation du **µC** il ne faut pas oublier de changer de bank mémoire pour les configurer.
- Pour accéder aux banques mémoire, on utilise le bit RP_0 ($5^{ième}$ bit du registre STATUS) :
  - $RP_0 = 0$: Accès à la BANK 0 ;
  - $RP_0 = 1$: Accès à la BANK 1.

DRES 06

Jeux d'instructions du micro contrôleur 16F84

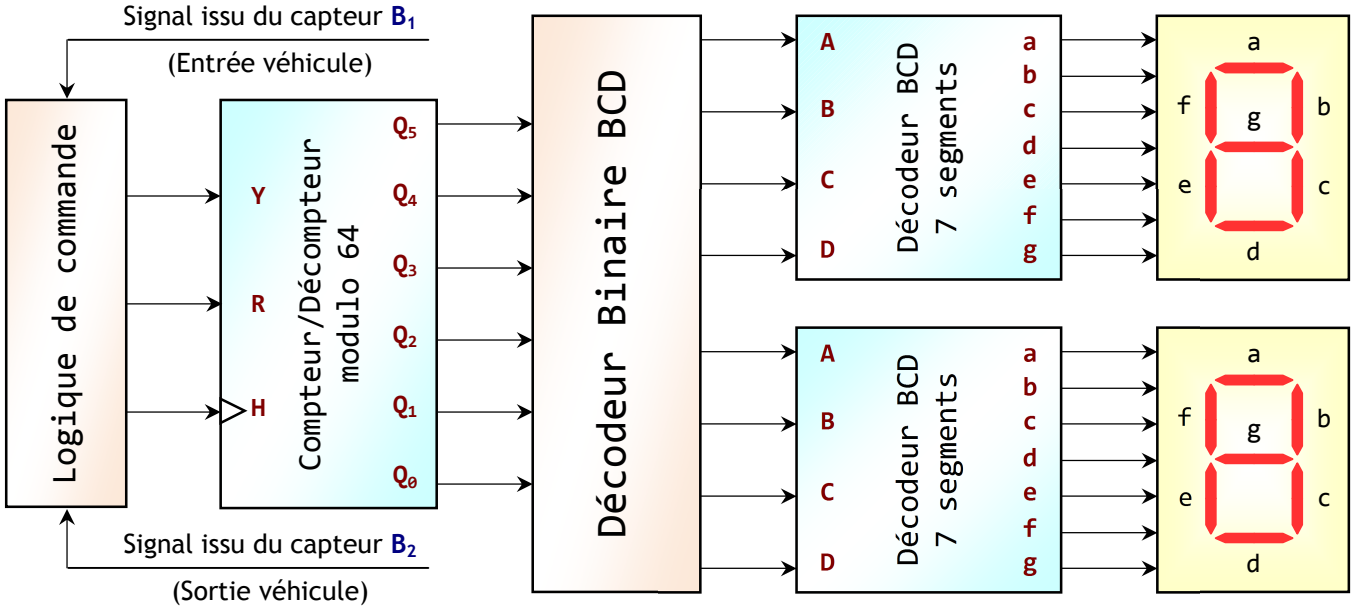
Le PIC16F84 a un jeu d'instructions relativement limité mais possède une architecture interne (RISC) qui permet une programmation efficace et rapide (toutes les instructions, exceptées les sauts, s'exécute en un cycle d'horloge).

Instructions opérant sur les registres		STATUS	Cycles
ADDWF F,d	W+F →{W,F ? d}	C, DC, Z	1
ANDWF F,d	W and F → {W,F ? d}	Z	1
CLRF F	Clear F	Z	1
COMF F,d	Complémente F → {W,F ? d}	Z	1
DECF F,d	Décrémente F → {W,F ? d}	Z	1
DECFSZ F,d	Décrémente F → {W,F ? d} skip if 0		1(2)
INCF F,d	Incrémente F → {W,F ? d}	Z	1
INCFSZ F,d	Incrémente F → {W,F ? d} skip if 0		1(2)
IORWF F,d	W or F → {W,F ? d}	Z	1
MOVF F,d	F → {W,F ? d}	Z	1
MOVWF F	W → F		1
RLF F,d	Rotation à gauche de F à travers C → {W,F? d}	C	1
RRF F,d	Rotation à droite de F à travers C → {W,F? d}	C	1
SUBWF F,d	F - W → {W,F ? d}	C, DC, Z	1
SWAPF F,d	Permute les 2 quartets de F → {W,F ? d}		1
XORWF F,d	W xor F → {W,F ? d}	Z	1
Instructions opérant sur les bits		**STATUS**	**Cycles**
BCF F,b	Mise à 0 du bit b e F		1
BSF F,b	Mise à 1 du bit b de F		1
BTFSC F,b	Teste le bit b de F, si 0 saute une instruction		1(2)
BTFSS F,b	Teste le bit b de F, si 1 saute une instruction		1(2)
Instructions opérant sur les constantes		**STATUS**	**Cycles**
ADDLW K	W + K → W	C, DC, Z	1
ANDLW K	W and K → W	Z	1
IORLW K	W or K → W	Z	1
MOVLW K	K → W	Z	1
SUBLW K	K - W → W	C, DC, Z	1
XORLW K	W xor K → W	Z	1
Autres instructions		**STATUS**	**Cycles**
CLRW	Clear W	Z	1
CLRWDT	Clear Watchdog timer	TO', PD'	1
CALL L	Branchement à un sous programme de label L		2
GOTO L	Branchement à la ligne de label L		2
NOP	Pas d'opération		1
RETURN	Retour d'un sous programme		2
RETFIE	Retour d'interruption		2
RETLW K	Retour d'un sous programme avec K dans W		2
SLEEP	Se met en mode standby	TO', PD'	1

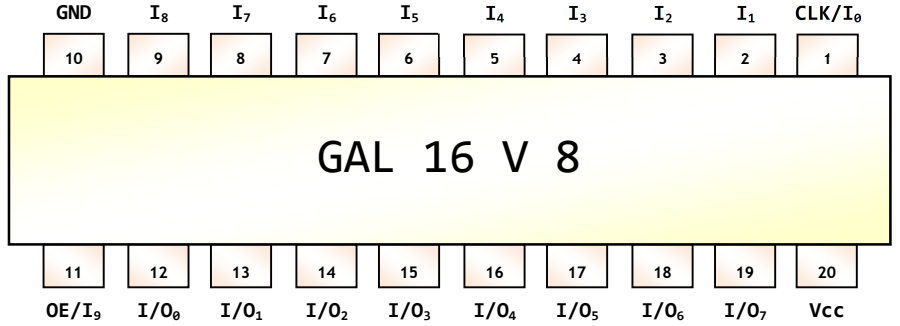
DRES 07

Affichage du nombre de véhicule

1. Affichage du nombre de véhicule :

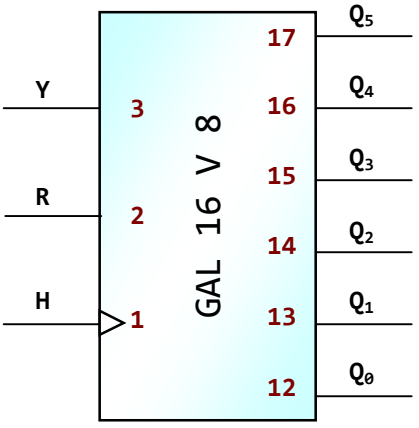


2. Circuit logique programmable GAL 16 V 8 :

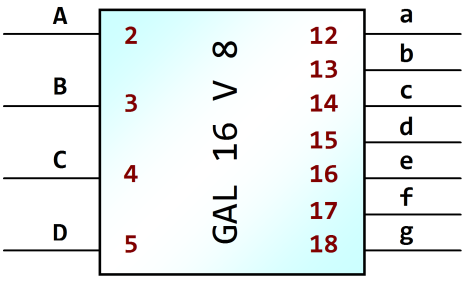


C'est un circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.
La lettre V veut dire que ce circuit est versatile, ce qui veut dire qu'il est possible par programmation de choisir entre une configuration de sortie combinatoire ou séquentielle.

3. Affectation des entrées/sorties du GAL 16 V 8 :



Compteur/Décompteur modulo 64



Décodeur BCD 7 segments

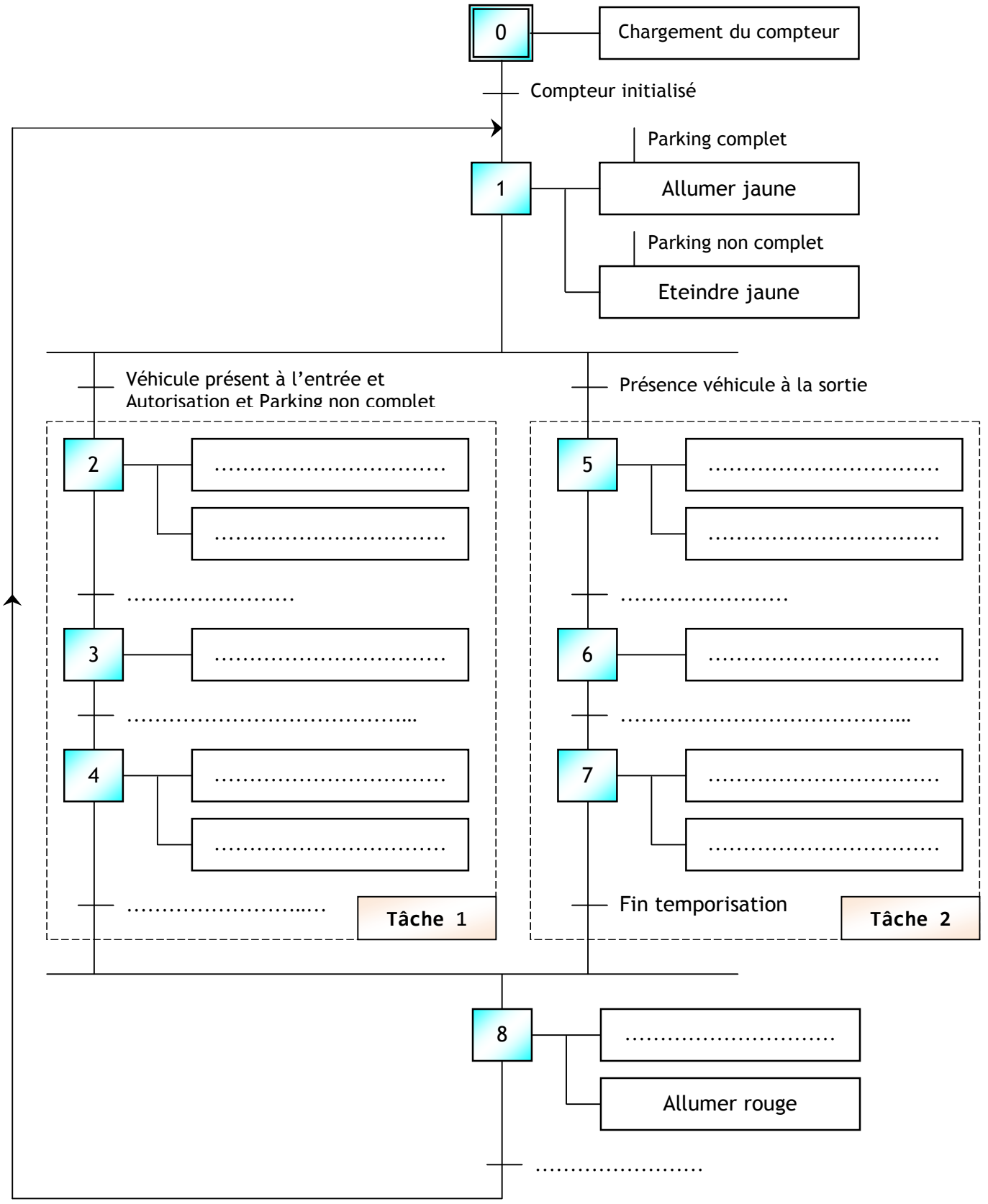
Pour le compteur/décompteur on donne :

- H : entrée d'horloge ;
- R : entrée de remise à zéro ;
- Y : entrée de comptage/décomptage ;
  - Y = 0 → comptage ;
  - Y = 1 → décomptage ;

GRAFCET DE FONCTIONNEMENT

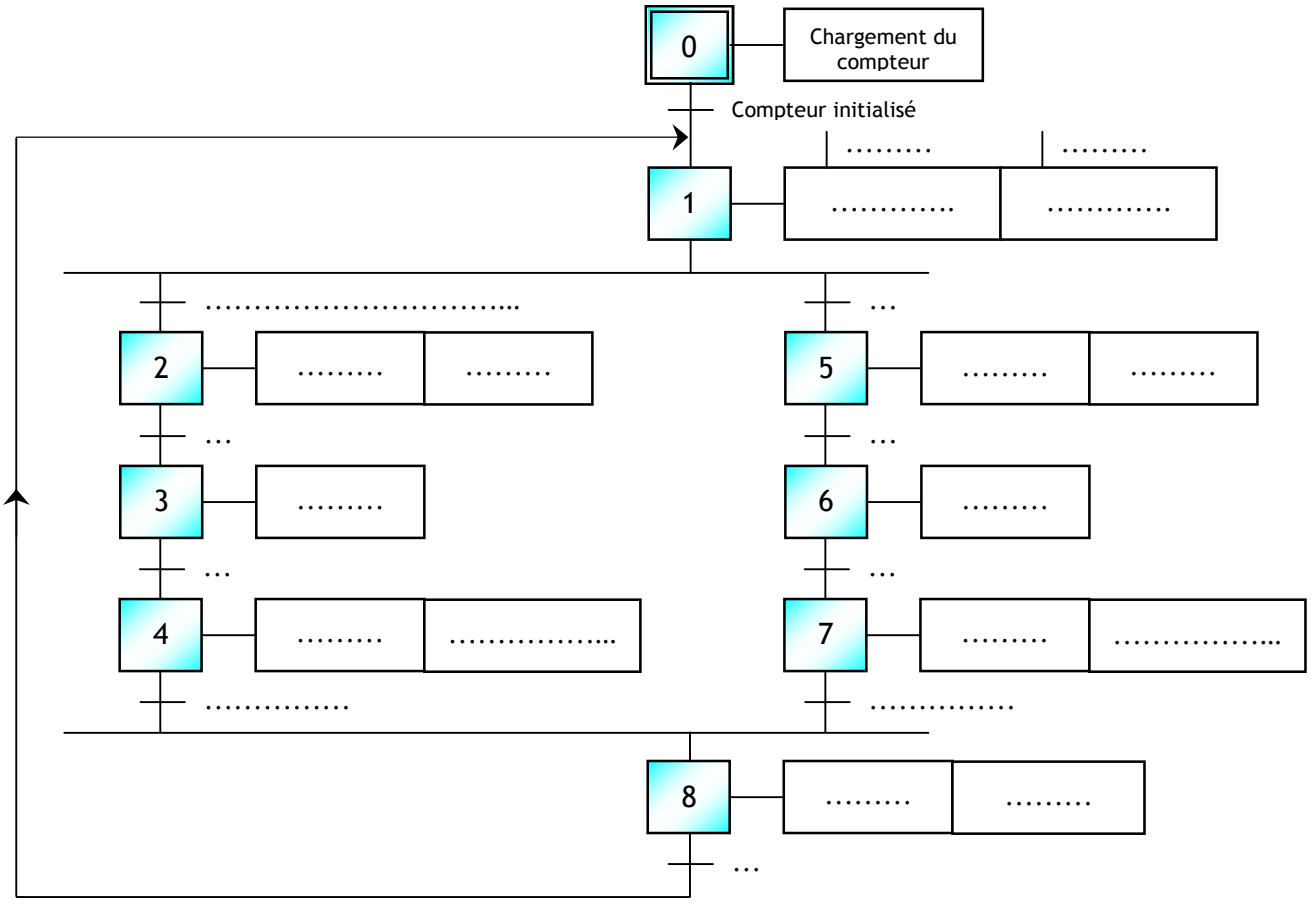
DREP 01

Q.1 :

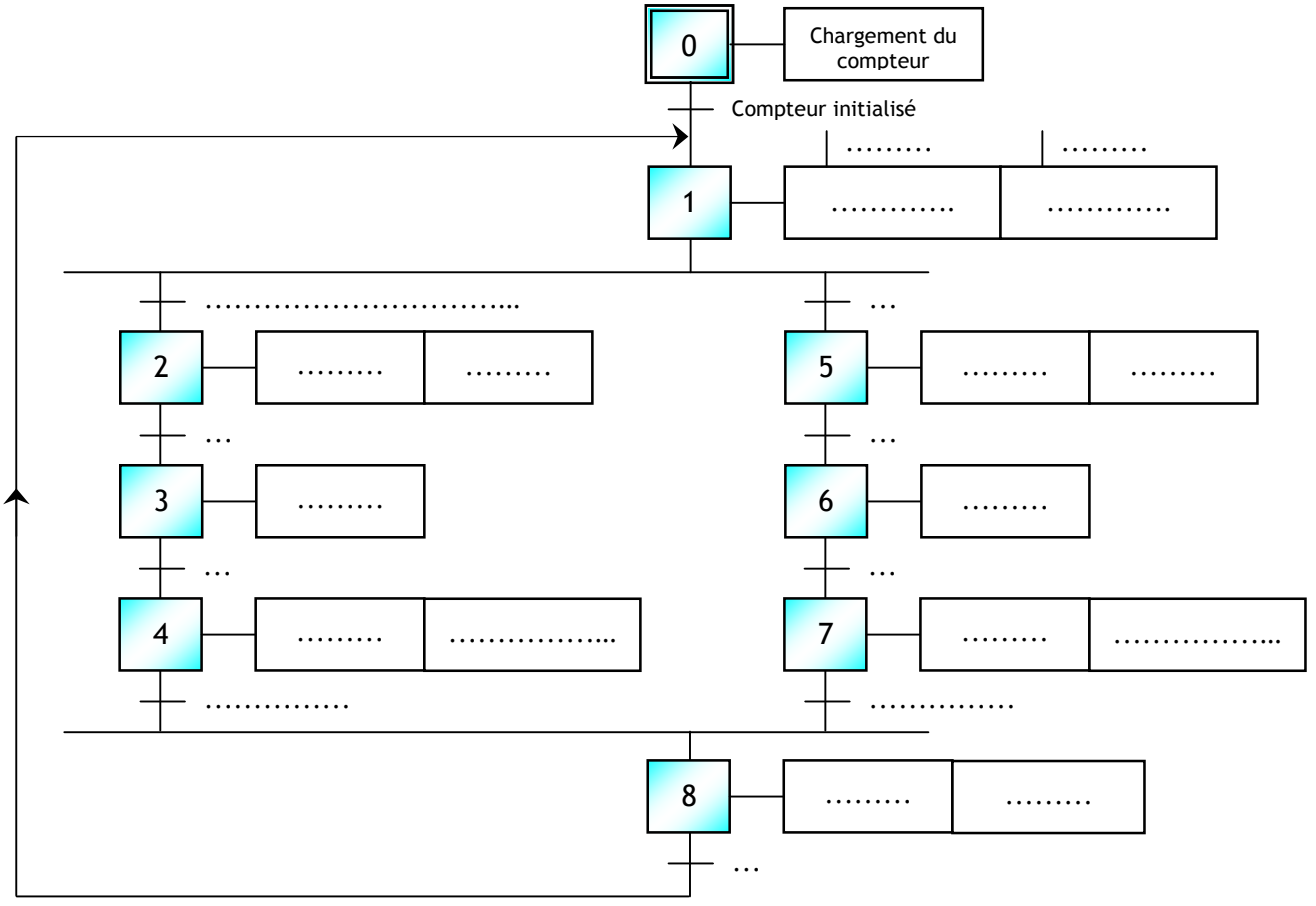


DREP 02

Q.2 :


0
Chargement du compteur
Compteur initialisé
1
2
3
4
5
6
7
8

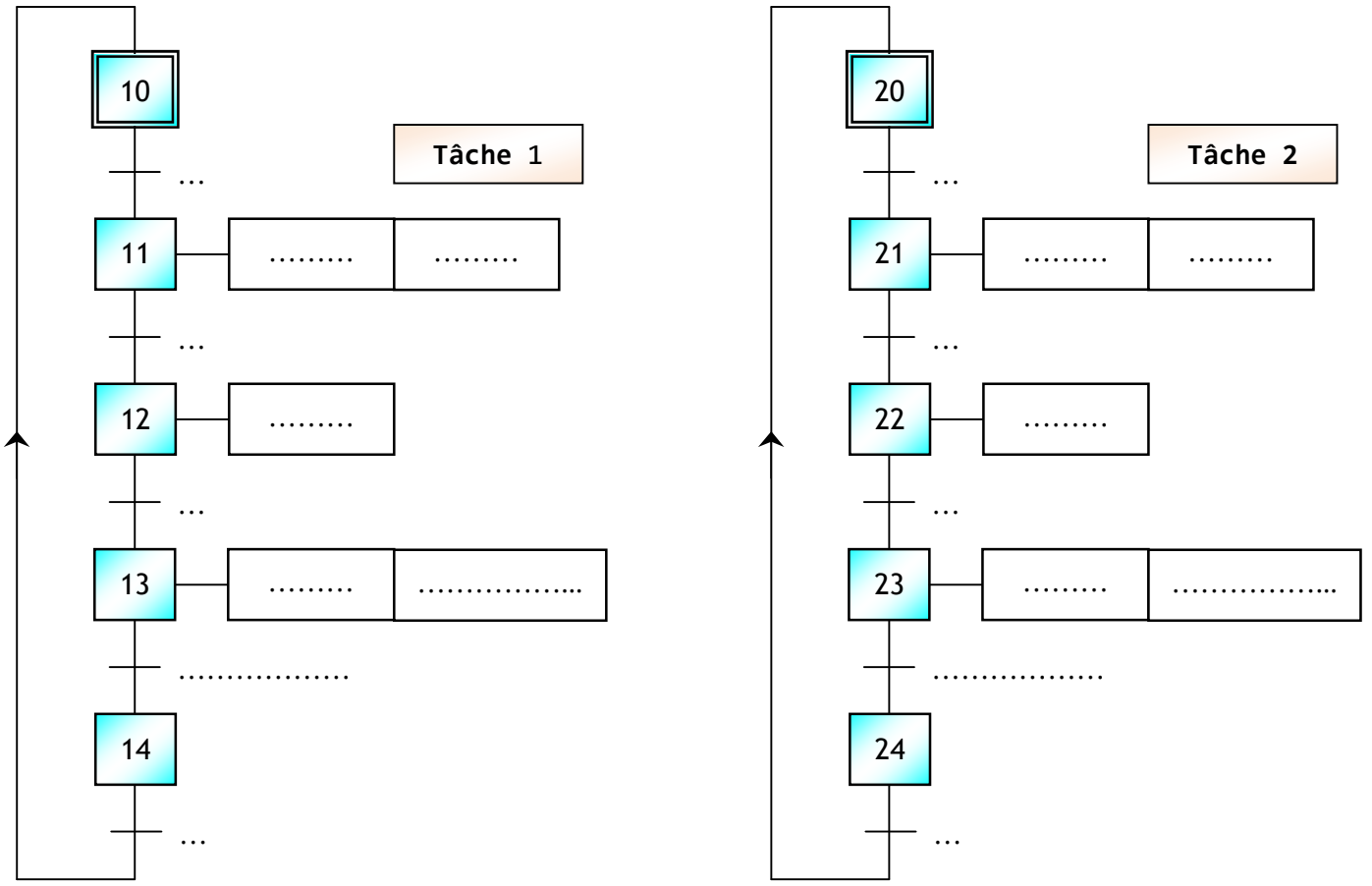
Q.3 :


0
Chargement du compteur
Compteur initialisé
1
2
3
4
5
6
7
8

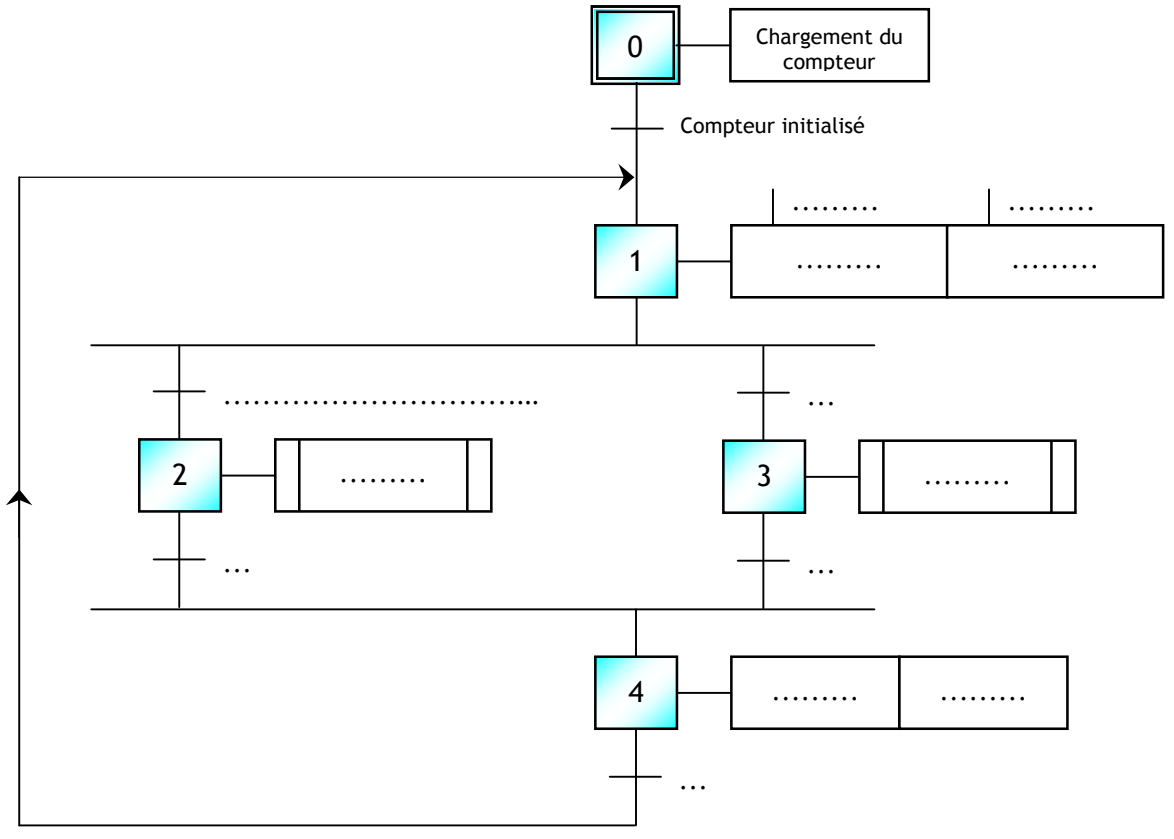
DREP 03

PATITION DU GRAFCET GLOBALE

Q.4 :



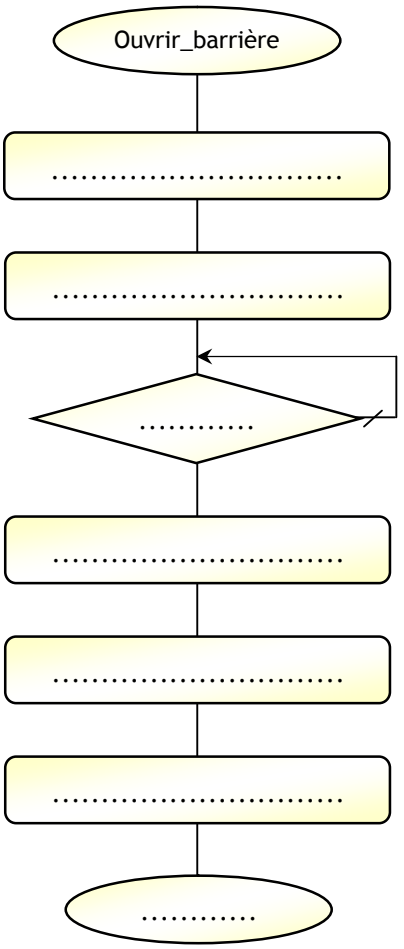
Q.5 :



DREP 04

OUVRIR ET FERMER LA BARRIÈRE

Q.6 :



Q.7 :

_____Sous programme Ouvrir_Barrière_____

Ouvrir_barriere .....

.....

LAB_7 .....

.....

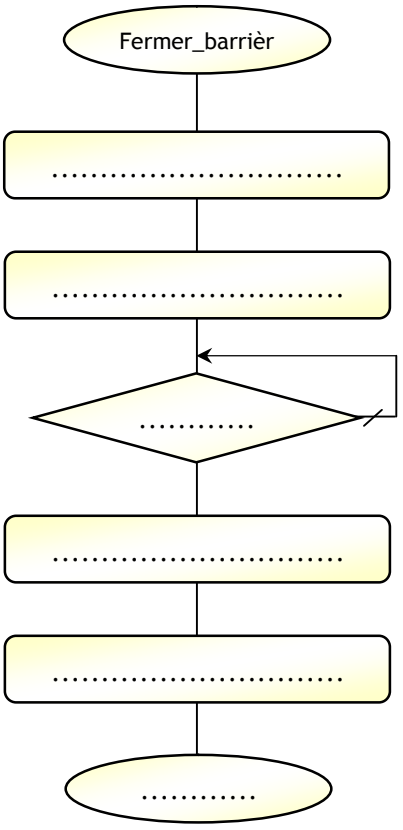
.....

.....

.....

.....

Q.8 :



Q.9 :

_____Sous programme Fermer_Barrière _____

Fermer_barriere .....

.....

LAB_8 .....

.....

.....

.....

.....

DREP 05

UTILISATION DE L'EEPROM

Q.10 :

_____Sous programme Lecture_EEPROM _____

```
Lecture_EEPROM   .....   ;
                 .....   ; L'adresse à lire
                 .....   ; Bank 1
                 .....   ; Lecture EPROM
                 .....   ; Bank 0
                 .....   ; EEDATA dans W
                 .....   ; W dans Cp1
                 RETURN  ; Retour
```

Q.11 :

_____Sous programme Ecriture_EEPROM _____

```
Ecriture_EEPROM  .....   ;
                 .....   ; Définition de l'adresse
                 .....   ;
                 .....   ; Définition de la donnée
                 .....   ; Bank 1
                 .....   ; Autorisation de l'écriture
                 .....   ;
                 .....   ; Écriture de 0x55
                 .....   ;
                 .....   ; Écriture de 0xAA
                 .....   ; Écriture dans EEPROM
        LAB9     .....   ;
                 .....   ; Écriture terminée ?
                 BCF     EECON, EEIF   ; Remise à zéro du témoin EEIF
                 BCF     STATUS, RP0   ; Bank 0
                 RETURN                ; Retour
```

DREP 06

TEMPORISATION AVEC TIMER0

Q.12 :

Sous programme de Temporisation

```
TEMPO     .....     ; Remise à zéro du TIMER
          .....     ;
          .....     ; Chargement du compteur Cp2
LAB10     .....     ; Indicateur de dépassement à 0
LAB11     .....     ; Test dépassement ?
          .....     ;
          .....     ; Décrémentation du compteur Cp2
          .....     ; Test fin de tempo ?
          .....     ; Retour
```

Q.13 :

.....
.....
.....
.....
.....
.....
.....
.....

..... | 16
.... | 16
....

N = ...10

N = ...16

Q.14 :

Registre	Mot de commande	
TRISA	B'.....'	H'.....'
TRISB	B'.....'	H'.....'
OPTION	B'.....'	H'.....'

Q.15 :

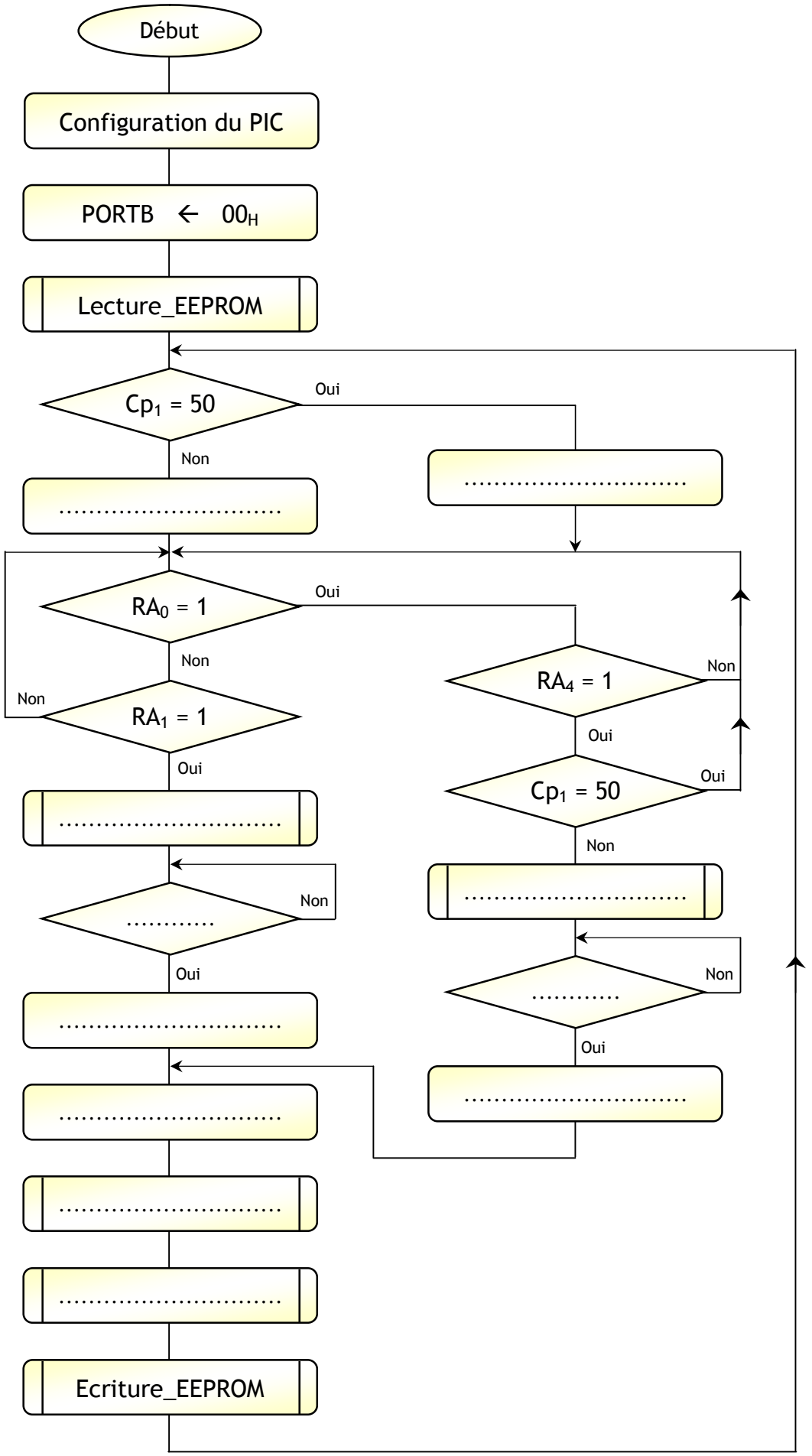
Programme de configuration

```
          .....     ; accès à la BANK 1
          .....     ;
          .....     ; configuration du PORTA
          .....     ;
          .....     ; configuration du PORTB
          .....     ;
          .....     ; configuration du TIMER 0
          .....     ; accès à la BANK 0
```

DREP 07

PROGRAMME PRINCIPAL

Q.16 :



DREP 08

Q.17 :

Programme principal

```
                CLRF      PORTB          ; moteur et voyants au repos
                CALL      Lecture_EPROM  ; nombre de véhicule dans Cp1
LAB1            .....                    ;
                .....                    ; comparer Cp1 et 50
                .....                    ; saut si Cp1 = 50 ?
                .....                    ; éteindre jaune
                .....                    ; saut si Cp1 ≠ 50 ?
                .....                    ; allumer jaune
LAB2            BTFSS     PORTA, 0       ; test si entrée?
                GOTO      SORTIE         ;
                GOTO      ENTRÉE         ;
SORTIE          .....                    ; test si sortie
                .....                    ;
                .....                    ; ouvrir la barrière
LAB3            .....                    ; test si véhicule est sortie ?
                .....                    ;
                .....                    ; décrémenter compteur
                .....                    ;
ENTREE          BTFSS     PORTA, 4       ; test si autorisation ?
                GOTO      LAB2           ;
                .....                    ;
                .....                    ; comparer Cp1 et 50
                .....                    ; saut si Cp1 = 50 ?
                .....                    ; parking disponible
                GOTO      LAB2           ; parking complet
LAB4            .....                    ; ouvrir la barrière
LAB5            .....                    ; test si véhicule est entré ?
                .....                    ;
                .....                    ; incrémenter compteur
LAB6            .....                    ; éteindre vert
                .....                    ; Temporisation 10 s
                CALL      Fermer_barriere ; fermer la barrière
                CALL      Ecriture_EPROM  ; enregistrer compteur
                GOTO      LAB1           ;
```

DREP 09

PROGRAMMATION DU GAL 16 V 8

Q.18 :

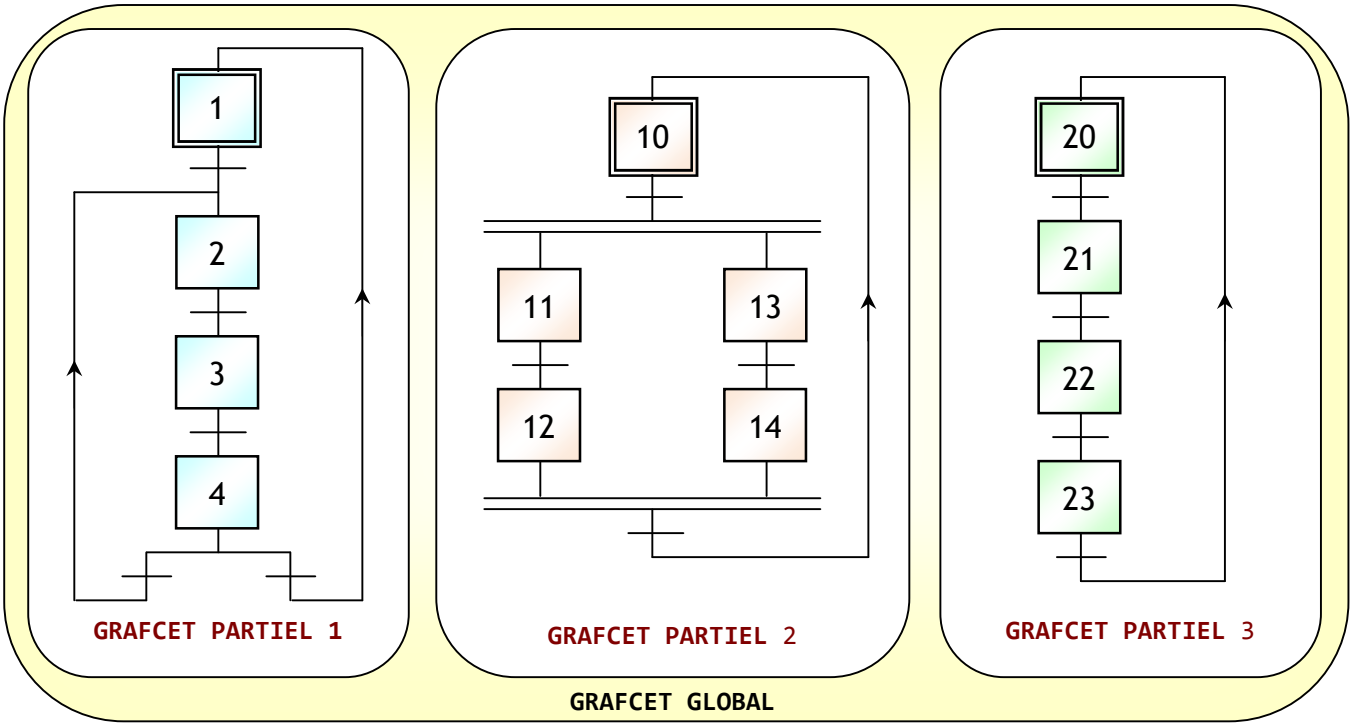
```
MODULE BCD
TITLE 'Décodeur BCD 7 segments '
DECLARATIONS
    ..........                    // PLD à programmer
    ..........                    // Variables d'entrée/sortie
    ..........                    // Définition du bus N
    ..........                    // Définition du bus S
EQUATIONS
    When N = 0 then  S = [1, 1, 1, 1, 1, 1, 0]  else
    When N = 1 then  S = [0, 1, 1, 0, 0, 0, 0]  else
    When N = 2 then  S = [1, 1, 0, 1, 1, 0, 1]  else
    When N = 3 then  S = [1, 1, 1, 1, 0, 0, 1]  else
    ..........
    ..........
    ..........
    ..........
    ..........
    When N = 9 THEN S = [1, 1, 1, 0, 0, 1, 1];
TEST_VECTORS(N -> [a, b, c, d, e, f, g])
                    0 -> [1, 1, 1, 1, 1, 1, 0] ;
                    1 -> [0, 1, 1, 0, 0, 0, 0] ;
                    ..........
                    ..........
                    ..........
                    ..........
                    ..........
                    ..........
                    ..........
                    ..........
END BCD.
```

Q.19 :

```
MODULE Compteur_reversible
TITLE  'Compteur  modulo 64'
DECLARATIONS
    Compteur device 'P16V8' ;                  // PLD à programmer
    H, R, Y pin  1, 2, 3 ;                     // Variables d'entrée/sortie
    Q0, Q1, Q2, Q3 , Q4, Q5 pin 14, 15, 16 Istype 'reg, buffer ';
    ..........                                 // Définition du bus Q
EQUATIONS
    ..........
    ..........
    ..........
  END Compteur_reversible.
```

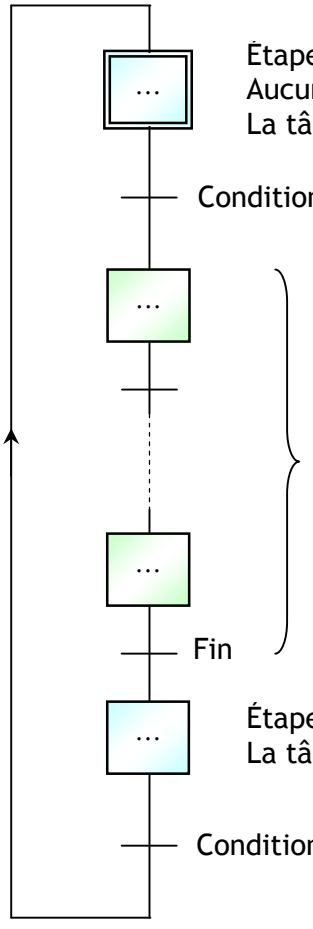
1. Notion de GRAFCET partiel :

Constitué d'un ou plusieurs GRAFCETS connexes. Un GRAFCET partiel résulte d'une partition, selon des critères méthodologiques, du GRAFCET global décrivant le comportement de la partie séquentielle d'un système.



2. Synchronisation de GRAFCET :

2.1. Notion de GRAFCET de Tâche :



Étape initiale de la tâche.
Aucune action extérieure n'y est associée.
La tâche est dite : **'en attente d'exécution'**

Conditions de lancement de la tâche.

Structure d'une tâche :

GRAFCET traduisant les cycles commandés par tâche.
La tâche est dite : **'en cours d'exécution'**

Fin

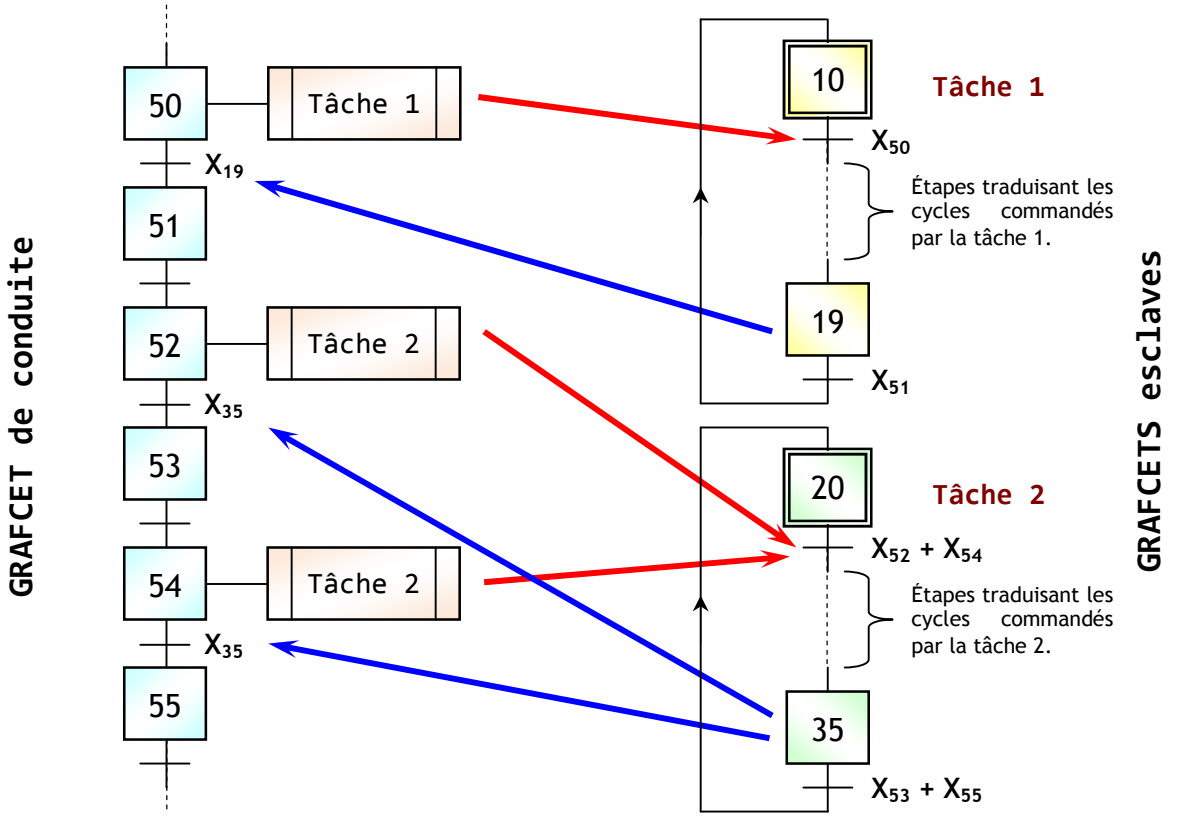
Étape de sortie **UNIQUE**.
La tâche est dite **'exécutée'**

Conditions de mise en attente.

Principe :

- Dans les automatismes séquentiels, il est fréquent de rencontrer des séquences répétitives dans le même cycle.
- Une séquence répétitive peut être représentée par un sous-GRAFCET ou GRAFCET sous-programme.
- Un GRAFCET sous-programme (de tâche) est écrit sous la forme d'un GRAFCET indépendant, connecté au GRAFCET principal (coordination).

2.2. Coordination Verticale asynchrone :



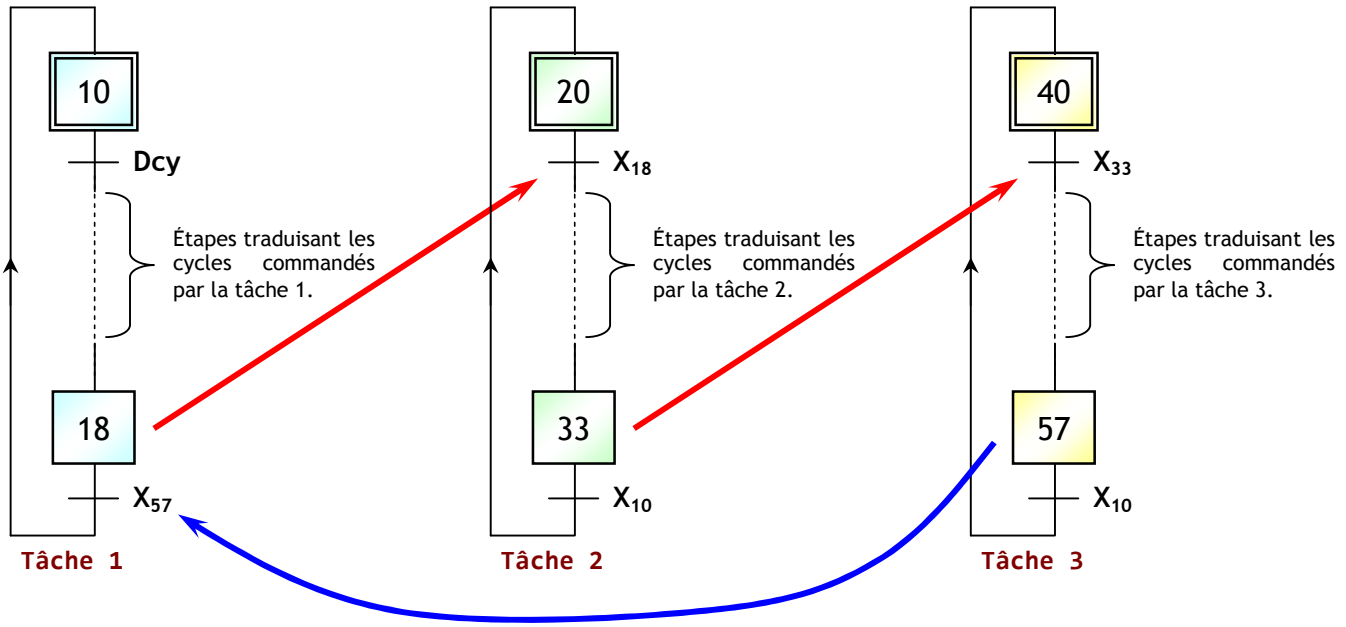
Tâche n°1 :

- L'activation de l'étape 50 lance la tâche 1(appel de la tâche) ;
- L'activation de l'étape 19 désactive l'étape 50 (fin de la tâche) ;
- La désactivation de l'étape 50, remet la tâche 1 en attente (activation de l'étape 10) ;

Tâche n°2 :

- L'activation de l'étape 52 ou 54 lance la tâche 2(appel de la tâche) ;
- L'activation de l'étape 35 désactive l'étape 52 ou 54 (fin de la tâche) ;
- La désactivation de l'étape 52 ou 54, remet la tâche 2 en attente (activation de l'étape 20) ;

2.3. Coordination horizontale :

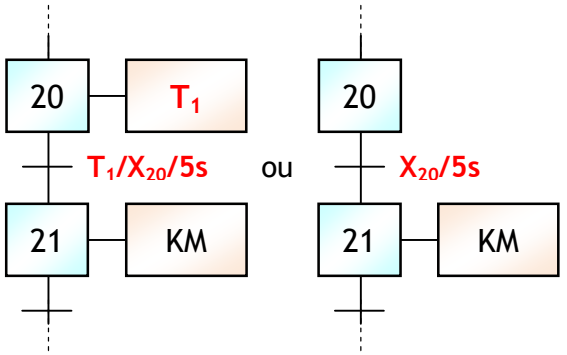


- Le GRAFCET global est constitué de trois GRAFCETS partiels intitulés tâche 1, tâche 2 et tâche 3 ;

- Une seule tâche est exécutée à la fois ;
- La réceptivité **Dcy** lance la tâche 1 ;
- La fin de la tâche 1 (étape 18 active), lance la tâche 2 ;
- La fin de la tâche 2 (étape 33 active), lance la tâche 3 ;
- La fin de la tâche 3 (étape 57 active), remet la tâche 1 en attente (activation de l'étape 10) ;
- L'activation de l'étape 10, remet les tâches 2 et 3 en attente (activation des étapes 20 et 40).

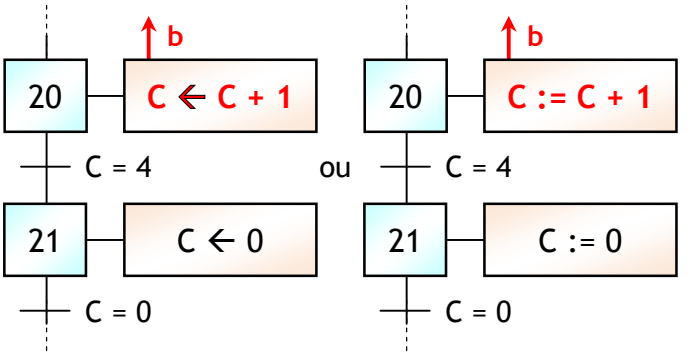
3. Différents types d'action :

3.1. Temporisation :



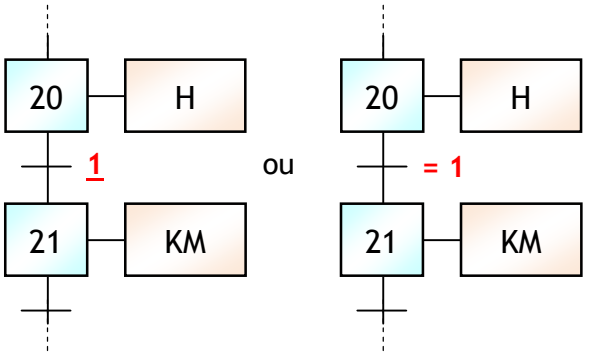
La transition 20 - 21 est franchie lorsque la temporisation, démarrée à l'étape 20 est écoulée, soit au bout de 5s

3.2. Comptage :



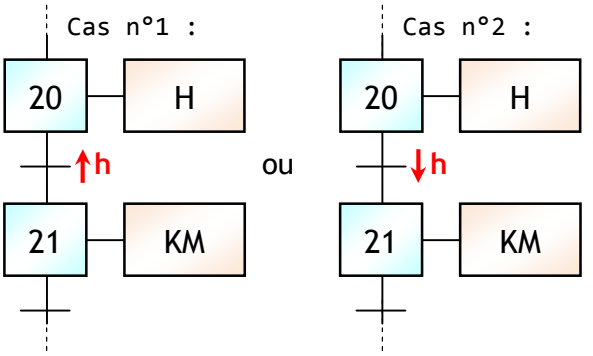
La transition 20 - 21 est franchie lorsque le contenu du compteur C est égal à 4. Le compteur est incrémenté sur front montant du signal b et il est mis à zéro à l'étape 21.

3.3. Réceptivité toujours vraie :



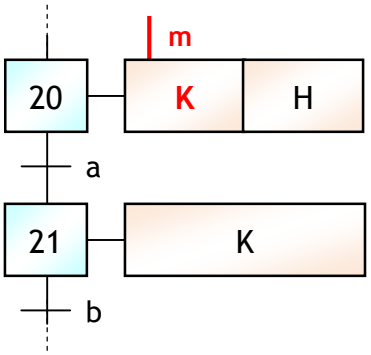
L'étape 21 est active, à l'activation de l'étape 20. La réceptivité associée à la transition 20-21 est toujours vraie.

3.4. Évènements (fronts) :



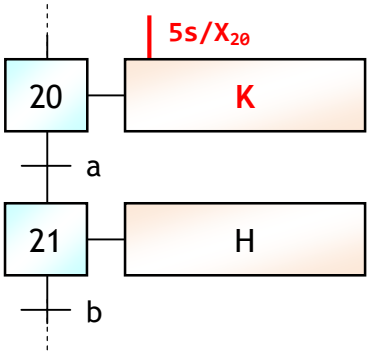
La transition 20 - 21 est franchie lors d'un front montant sur h (cas n°1), ou lors d'un front descendant sur h (cas n°2).

3.5. Action conditionnelle :

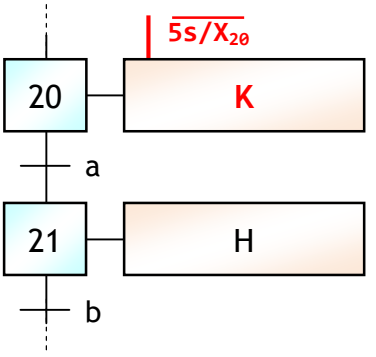


L'action K devient effective à l'étape 20, lorsque la condition m est vraie.

$K = X_{20}.\ m + X_{21}$

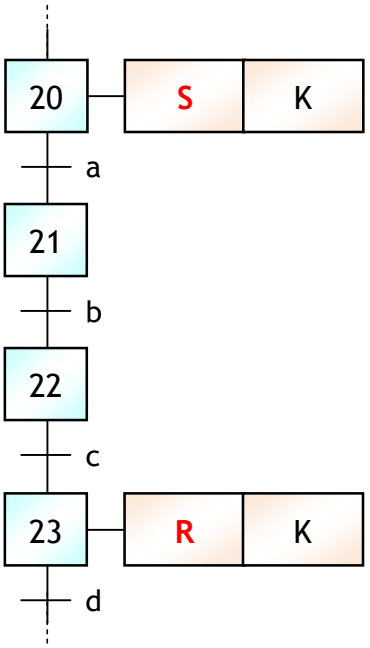


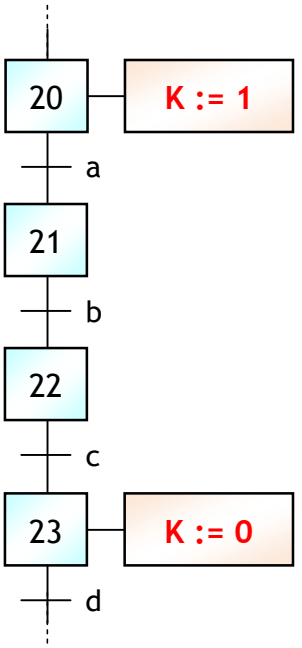
L'action K devient effective à l'étape 20, après 5 secondes.



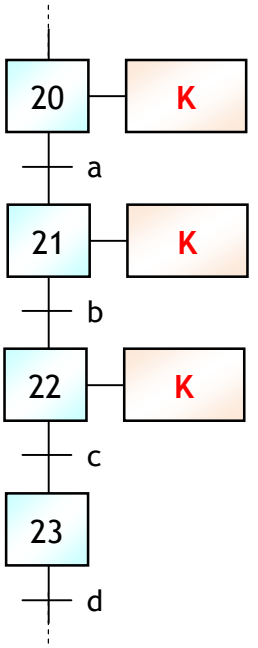
L'action K devient effective à l'étape 20, et dure 5 secondes.

3.6. Action mémorisée :



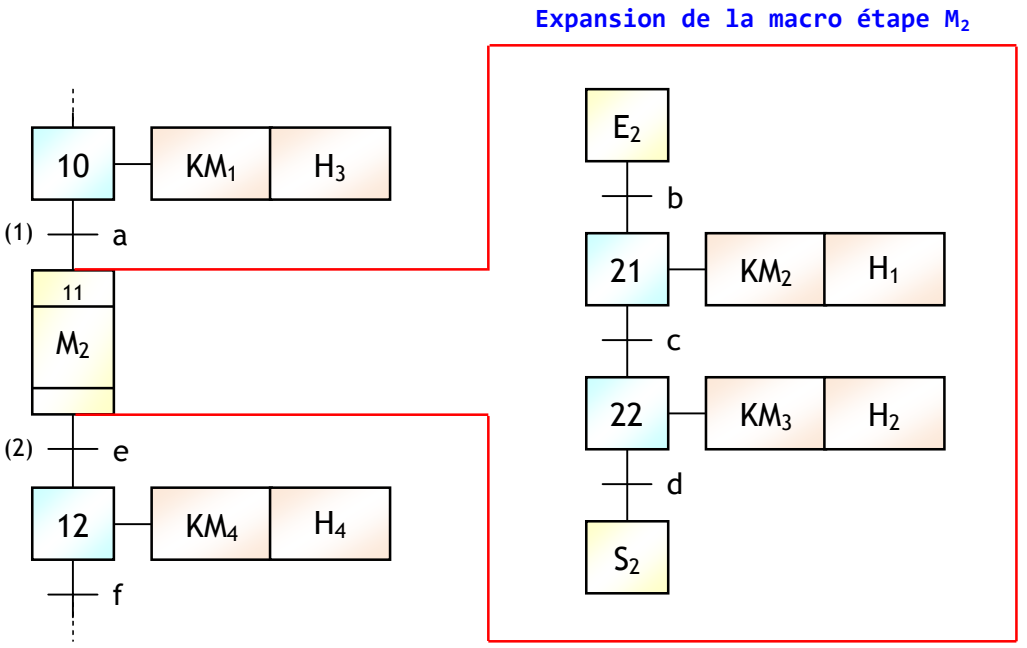


Mise à 1 de l'action par la lettre S (set)
Mise à 0 de l'action par la lettre R (reset)



L'action K est active aux étapes 20, 21 et 22.

4. Notion de Macro- étape :



L'expansion de la macro-étape commence par une seule étape d'entrée et se termine par une seule étape de sortie, étapes qui représentent les seuls liens possibles avec le GRAFCET auquel elle appartient.

Remarque :

- Le franchissement de la transition (1) active l'étape E_2 ;
- La transition (2) ne sera validée que lorsque l'étape S_2 sera active ;
- Le franchissement de la transition (2) désactive l'étape S_2 ;
- L'utilisation de la macro-étape dans un GRAFCET permet non seulement de simplifier la représentation mais aussi d'éviter les séquences linéaires trop longues.

1. Présentation :

Le PIC 16F84 possède une EEPROM de **64** octets pour y stocker les données. Contrairement à l'EEPROM de programme, elle n'est pas adressée par le PC, mais par un registre séparé appelé **EEADR** se trouvant à l'adresse 09_H dans le fichier des registres. Les données sont accessibles par le registre **EEDATA** d'adresse 08_H. 2 registres de contrôle sont associés à cette mémoire **EECON1** et **EECON2** d'adresse 88_H et 89_H. La durée d'écriture d'un octet est de l'ordre de **10 ms**.

2. Les registres :

- **Le registre EEDATA :**

Ce registre permet de lire ou d'écrire une donnée dans la mémoire non volatile (EEPROM).

- **Le registre EEADR :**

Ce registre contient l'adresse de la donnée se trouvant dans l'EEPROM.

- **Le registre EECON1 :**

C'est un registre de contrôle qui permet d'exécuter une lecture ou une écriture dans l'EEPROM. Seuls les 5 bits de poids faible sont utilisés.

- **Le registre EECON2 :**

Ce registre est exclusivement utilisé pour les séquences d'écritures dans l'EEPROM. Il n'a pas d'adresse physique et la lecture de ce registre retourne une valeur nulle.

Une donnés ne peut être écrite qu'après avoir écrit successivement **0x55** et **0xAA** dans **EECON2**.

3. Structure du registre EECON1 :

7	6	5	4	3	2	1	0
x	x	x	EEIF	WRERR	WREN	WR	RD

Bit 0 : RD - ReaD EEPROM -

Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une lecture de l'EEPROM. Après le cycle de lecture, il est mis automatiquement à 0.

Bit 1 : WR - WRite EEPROM -

Lorsque ce bit est mis à "1", il indique au microcontrôleur que l'on souhaite une écriture de l'EEPROM. Après le cycle d'écriture, il est mis automatiquement à 0.

Bit 2 : WREN - WRite ENABLE EEPROM -

C'est un bit de confirmation d'écriture dans l'EEPROM. En effet, il ne suffit pas de définir un cycle d'écriture uniquement avec le bit WR. Il faut impérativement valider le bit WREN (WREN = 1) pour autoriser une écriture.

Bit 3 : WRERR - EEPROM WRite ERROR flag -

Ce drapeau indique qu'une erreur s'est produite lors d'un cycle d'écriture dans l'EEPROM.

WRERR = 1 : une opération d'écriture a échouée ;

WRERR = 0 : le cycle d'écriture s'est déroulé normalement.

Bit 4 : EEIF - EEPROM Interrupt Flag -

EEIF est un drapeau qui génère une interruption lorsqu'un cycle d'écriture s'est déroulé normalement. Il doit être mis à 0 lors de la routine d'interruption.

EEIF = 1 : l'opération s'est déroulé correctement ;

EEIF = 0 : soit l'opération n'a pas commencé, soit n'est pas terminée.

4. Écriture et lecture de l'EEPROM :

4.1. Lecture d'une donnée :

- Placer l'**adresse** de la donnée à lire dans **EEADR** ;
- Mettre le bit **RD** de **EECON1** à **1** ;
- Lire le contenu du registre **EEDATA**.

```
        .....   ; Bank 0
        .....   ;
        .....   ; l'adresse à lire
        .....   ; Bank 1
        .....   ; lecture EPROM
        .....   ; Bank 0
        .....   ; W ← EEDATA
```

4.2. Écriture d'une donnée :

- Placer l'**adresse** de la donnée à écrire dans **EEADR** ;
- Placer la **donnée** à écrire dans **EEDATA** ;
- Mettre le bit **WREN** de **EECON1** à **1** pour autoriser l'écriture ;
- Placer **0x55** dans **EECON2** ;
- Placer **0xAA** dans **EECON2** ;
- Mettre le bit **WR** de **EECON1** à **1** ;
- Attendre que le bit **EEIF** soit à **1** ;
- On peut utiliser l'interruption produite par **EEIF** en la validant par le bit **EEIE** de **INTCON** ;
- N'oublier pas de remettre **EEIF** à **0**.

```
        .....   ; Bank 0
        .....   ;
        .....   ; définition de l'adresse
        .....   ;
        .....   ; définition de la donnée
        .....   ; Bank 1
        .....   ; autorisation de l'écriture
        .....   ;
        .....   ; écriture de 0x55
        .....   ;
        .....   ; écriture de 0xAA
        .....   ; écriture dans EEPROM
Lab     .....   ;
        .....   ; écriture terminée
        .....   ; remise à 0 de EEIF
        .....   ; Bank 0
```

1. Présentation :

Le PIC 16F84 dispose d'un TIMER, c'est un module programmable dont les fonctions principales sont :

- La génération de signaux périodiques (astable) ;
- La génération d'impulsions (monostable) ;
- Le comptage d'évènements (compteur) ;
- La génération de signaux PWM (modulation de largeur d'impulsions pour les Mcc).

2. Les registres du TIMER0 :

- **Le registre TMR0 :**

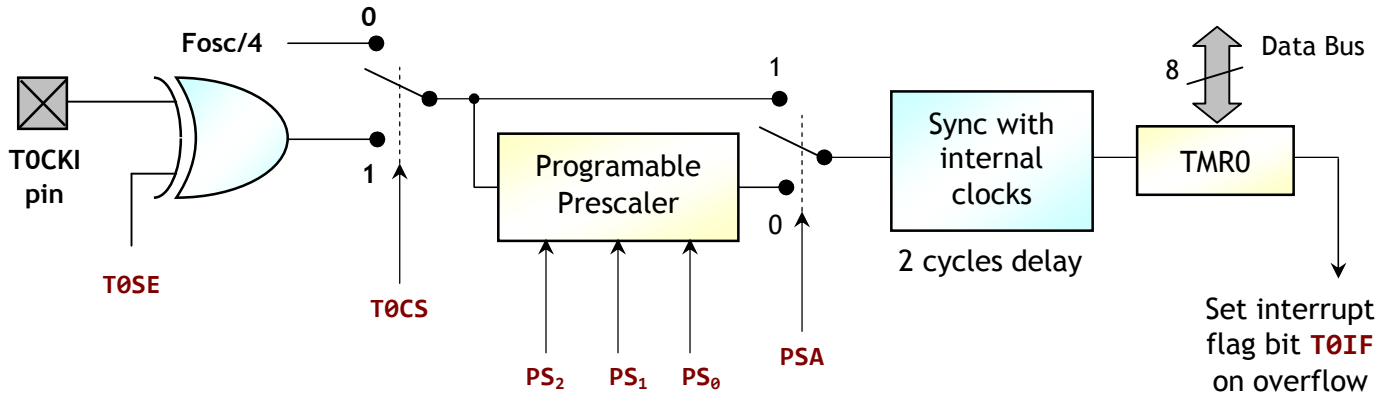
Ce registre de 8 bits s'incrémente de "**1**" a chaque impulsion de l'horloge **interne** (**Fosc/4**) ou par une horloge **externe** appliquée sur la broche **TOCKI/ RA_4**. Il est associé au module TIMER/Compteur. Ce registre se trouve à l'adresse 01_H.

- **Le registre OPTION :**

Ce registre contiens les bits de contrôles du **PRESCALER**, de l'interruption externe **INT**, de la sélection **TIMER/Compteur** et du "**tirage au plus**" du PORTB. Ce registre se trouve à l'adresse 81_H.

3. Fonctionnement :

3.1. Schéma simplifié :



3.2. Structure du registre OPTION :

7	6	5	4	3	2	1	0
RBPU	INTEDG	TOCS	TOSE	PSA	PS_2	PS_1	PS_0

Bit 7 : RBPU - PORTB Pull-Up -

RBPU = 1 : le "tirage au plus" interne du PORTB est désactivé.

RBPU = 0 : le "tirage au plus" interne du PORTB est activé.

Bit 6 : INTEDG - INTerrupt EDGe -

INTEDG = 1 : la broche RB_0/INT génère une interruption sur un front montant ;

INTEDG = 0 : la broche RB_0/INT génère une interruption sur un front descendant.

Bit 5 : TOCS - TMR0 Clock Source -

Il permet de sélectionner le mode de fonctionnement du TIMER/Compteur :

TOCS = 1 : sélection de l'horloge externe (RA_4) qui correspond au COMPTEUR ;

TOCS = 0 : sélection de l'horloge interne qui correspond au mode TIMER.

Bit 4 : TOSE - TMR0 Source Edge –

Ce bit détermine sur quel front -montant ou descendant- l'entrée RA_4 incrémentera le registre TMR0 :

TOSE = 1 : front descendant ;

TOSE = 0 : front montant.

Bit 3 : PSA - PreScaler Assignment -

PSA = 1 : alors le Prescaler est associé avec le WDT ;

PSA = 0 : alors le Prescaler est associé avec le TIMER.

Bit 2, 1, 0 : PS_0, PS_1, PS_2 - Prescaler Select -

Ces trois bits effectuent une division de la fréquence d'horloge du Prescaler.

Remarque :

- Lorsque le TIMER0 déborde, le BIT **TOIF** du registre INTCON passe à 1. Après utilisation, n'oublier pas de remettre **TOIF** à zéro.
- Le Prescaler est partagé avec le Timer WDT (WATCHDOG)

PS_2	PS_1	PS_0	RATIO TMRO	RATIO WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

4. Le WATCHDOG Timer WDT (Chien de garde):

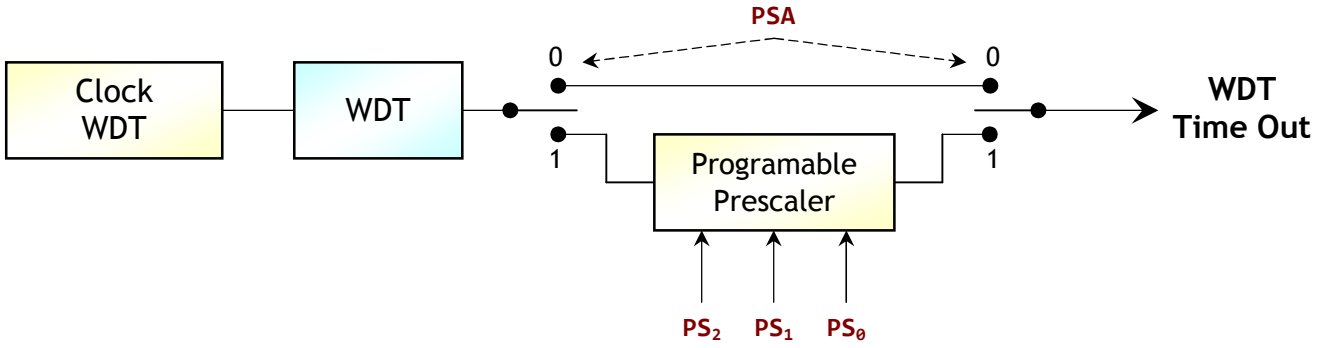
C'est un compteur 8 bits incrémenté en permanence (même si le µC est en mode sleep) par une horloge RC intégrée indépendante de l'horloge système. Lorsqu'il déborde, (**WDT Time Out**), deux situations sont possibles :

- Si le µC est en fonctionnement normal, le **WDT Time-out** provoque un **RESET**. Ceci permet d'éviter de rester planté en cas de blocage du microcontrôleur.
- Si le µC est en mode SLEEP, le **WDT Time-out** provoque un **WAKE-UP**, l'exécution du programme continue normalement là où elle s'est arrêtée avant de rentrer en mode SLEEP. Cette situation est souvent exploitée pour réaliser des temporisations.

L'horloge du WDT a une période voisine de **70 µs** ce donne un **Time-Out** toutes les **18 ms**. Il est cependant possible d'augmenter cette durée en faisant passer le signal **Time-Out** dans le Prescaler programmable, partagé avec le timer TMR0.

L'usage du WDT doit se faire avec précaution pour éviter la réinitialisation inattendue et répétée du programme. Pour éviter un **WDT Time Out** lors de l'exécution d'un programme, on a deux possibilités :

- Inhiber le WDT de façon permanente en mettant à 0 le bit **WDTE** dans le registre de configuration ;
- Remettre le WDT à 0 périodiquement dans le programme principal à l'aide de l'instruction **CLRWDT** pour éviter qu'il ne déborde.

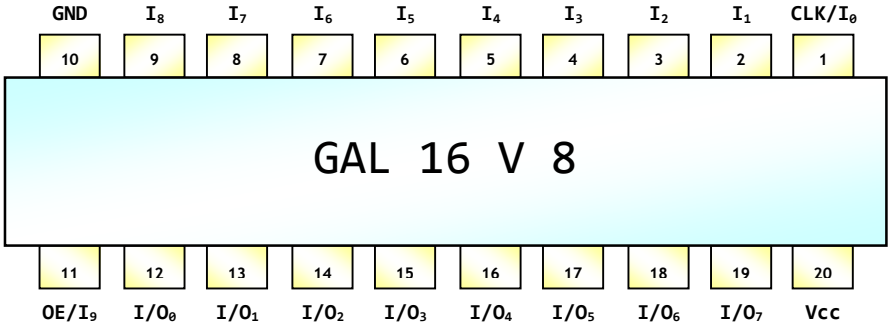


1. Introduction :

La programmation d'une fonction logique dans un PLD doit passer par les étapes suivantes :

- Définition de la fonction par des équations logiques et ou des logigrammes ;
- Choix du PLD (PAL, GAL, MACH…) doit être choisit selon les critères suivants :
  - Coût du système ;
  - Matériel de programmation ;
  - Outil de développement (programme) ;
  - Nombre d'entrées et de sorties nécessaire pour réaliser la fonction ;
  - Type de la fonction et degrés de complexité de sa structure interne… ;
- L'écriture du programme de description en utilisant différentes solutions (équations, tables de vérité, diagrammes d'état) ;
- Test du programme par une éventuelle simulation ;
- Transfert du programme au composant PLD.

2. Exemple de PLD :



C'est un circuit logique programmable et effaçable électriquement.

Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.

La lettre V veut dire que ce circuit est versatile, ce qui veut dire qu'il est possible par programmation de choisir entre une configuration de sortie combinatoire ou séquentielle.

3. Initiation au langage ABEL :

3.1. Structure :

Section d'en-tête :
```
MODULE logique1
TITLE 'alarme de phares'
```

Section des déclarations :
```
DECLARATIONS
        C, L, P pin 18, 19, 20 ;          // Variables d'entrée
        S1, S2 pin 16, 17 istype 'com'; // Variables de sortie
```

Section de description :
```
EQUATIONS
        S1 = !C & L & P;
        S2 = P;
```

Section de test :
```
TEST_VECTORS
        ([C, L, P ] -> [S1, S2]) ;
          [ 0, 0, 0 ] -> [.X.,.X.] ;
          [ 0, 1, 1 ] -> [.X.,.X.] ;
```

Section de fin :
```
END logique1
```

Un programme en langage ABEL se compose de 5 sections principales : l'entête, les déclarations, la description, le test et la fin ;

3.2. Entête et fin de fichier :

L'entête est reconnu par le mot clé **MODULE** suivi du nom du fichier ; on peut aussi ajouter au programme un titre précédé du mot **TITLE** ; la section de la fin est identifiée par le mot clé **END** suivi du nom de fichier qui a été attribué au MODULE.
Dans tout le programme on peut ajouter des commentaires précédés du symbole **//.**

3.3. Partie Déclarations :

Elle commence par le mot clé **DECLARATIONS**. Dans cette section, on indiquera :

- La spécification du PLD à programmer ;
- L'identification des broches utilisées -noms et numéros- ;
- La spécification du type des sorties.

La directive **device** permet de spécifier le type du PLD à programmer.

Exemple : Decodeur device 'P22V10 ; //il s'agit du GAL 22V10

La directive **pin** permet d'affecter un numéro de broche à une variable (entrée ou sortie).

Exemple : C, L, P pin 18, 19, 20 ;

Le mot **istype**, suivi d'un argument définit le type de la variable de sortie :

- **Istype 'com, buffer'** indique que la sortie est combinatoire active sur niveau haut.
- **Istype 'com, invert'** indique que la sortie est combinatoire active sur niveau bas.
- **Istype 'reg, buffer '** indique que la sortie est séquentielle - registre - active sur niveau haut.
- **Istype 'reg, invert '** indique que la sortie est séquentielle - registre - active sur niveau bas.

Exemple: S_1, S_2 **pin** 16, 17 **istype 'com, buffer'**;

On peut aussi trouver dans cette section les déclarations de groupement de variables –bus-.

Exemple :

$N = [E_3, E_2, E_1, E_0]$;

$S = [Q_7, Q_6, Q_5, Q_4, Q_3, Q_2, Q_1, Q_0]$;

- Le mot N est constitué des bits E_3, E_2, E_1, E_0 ;
- E_3 constitue le MSB (bit de poids fort) et E_0 le LSB (bit de poids faible) ;
- Le mot S est constitué des bits Q_7, Q_6, Q_5, Q_4, Q_3, Q_2, Q_1, Q_0 ;
- Q_7 constitue le MSB et Q_0 le LSB.

Le langage ABEL permettra de traiter ces variables individuellement ou le nombre résultant de leur association. Ceci permettra de simplifier considérablement l'écriture de certaines équations.

3.4. Partie test :

Est commencé par le mot clé **TEST_VECTORS** ; l'utilisateur vérifie avec des exemples de valeurs données aux entrées les états des sorties et s'assure du bon fonctionnement du programme ;

Exemple :

```
TEST_VECTORS    ([a, b, c, d] => [S1, S2])// Variables d'entrée et variables de sortie
([0, 1, 0, 1] => [1, 1]) ; // Les sorties sont à 1 pour la combinaison 0101 des entrées
([1, 0, 1, 0] => [0, 0]) ; // Les sorties sont à 0 pour la combinaison 1010 des entrées
```

3.5. Partie de description logique :

C'est dans cette partie qui est décrite la fonction à programmer, la description peut se faire de plusieurs moyens (équations, table de vérité, diagramme d'état)

3.5.1. Description par équations :

Elle est annoncée par le mot clé **EQUATIONS**, elle décrit les relations liant les sorties aux entrées ; ces relations peuvent être des fonctions logiques, arithmétiques ou relationnelles :
Le langage ABEL met à la disposition de l'utilisateur un jeu d'opérateurs complet en voici les détails :

Opérateurs logiques	
!	Complément
#	OU
&	ET
$	OU exclusif
=	Affectation combinatoire
:=	Affectation séquentielle

Opérateurs arithmétiques	
+	Addition
-	Soustraction
*	Multiplication
/	Division entière
%	Reste de la division entière
<<	Décalage à gauche
>>	Décalage à droite

Opérateurs relationnels	
==	Test d'égalité
!=	Test différent
<	Test inférieur
>	Test supérieur
<=	Test inférieur ou égal
>=	Test supérieur ou égal

Pour travailler avec les nombres, ABEL précise la base du système de numération adoptée en utilisant les symboles suivants :

- Nombre binaire : ^b exemple : ^b1001
- Nombre octal : ^o exemple : ^o 45
- Nombre hexadécimal : ^h exemple : ^h FA
- Nombre décimal : ^d ou rien exemple : ^ 95

Exemple 1 :

```
EQUATIONS
    S1 = a &! b & ! c # ! a & b & c;
    S2 = ! (a # b & c);
    L = A1 == B1 ;
```

Exemple 2 :

```
EQUATIONS
    S = A > B ; // S = 1 si A > B
    I = A < B ; // I = 1 si A < B
    E = A = B ; // S =1 si A = B
```

Exemple 3 :

Utilisation de **When …. Then ….**

```
EQUATIONS
    When A > B then S = 1;
    When A = = B then I = 1;
    When A < B then E = 1;
```

Exemple 4 :

```
DECLARATIONS
    A3,A2,A1,A0 pin 2,3,4,5;
    S3,S2,S1,S0 pin 8,9,10,11 istype 'com';
    S=[S3,S2,S1,S0] ;  // déclaration du groupe S
    A=[A3,A2,A1,A0] ; // et du groupe A
EQUATIONS
    A= S<<1 ;  //décalage à gauche de 1 bit
               //A3=S2  A2=S1 A1=S0  A0=0
    S= A & [0, 1, 0,0] ;
// masque (test du bit A2) : S= [0, A2, 0,0]
```

Exemple 5 :

Utilisation de **When …. Then …. Else ….**

```
EQUATIONS
    When A > B then S = 1 Else
        When A = = B then I = 1 Else
            E = 1;
```

3.5.2. Description par table de vérité :

Elle est annoncée par le mot clé **Truth_Table** suivi des nombres de variables d'entrée et de sortie, on établit juste après la table de vérité en respectant les dispositions des variables d'entrée et de sortie comme le montre l'exemple suivant :

Truth_table ([A, B] -> [S_1, S_2, S_3])

[0, 0] -> [0, 1, 0] ;

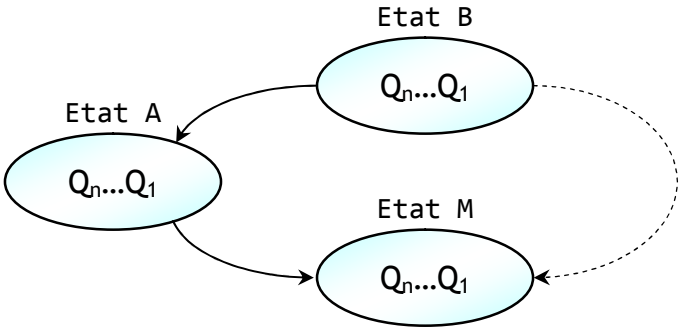
[0, 1] -> [0, 0, 1] ;

[1, 0] -> [1, 0, 0] ;

[1, 1] -> [1, 1, 1] ;

3.5.3. Description par diagramme d'état :

Le diagramme d'état est souvent avantagé dans le cas des machines séquentielles dont le fonctionnement est décrit par l'enchaînement séquentiel des états de sortie.



Instruction GOTO : (aller à)

Utilisée pour les transitions d'état inconditionnelles. **GOTO** peut être utilisée si l'état présent n'offre qu'un seul état suivant.

Format du diagramme :

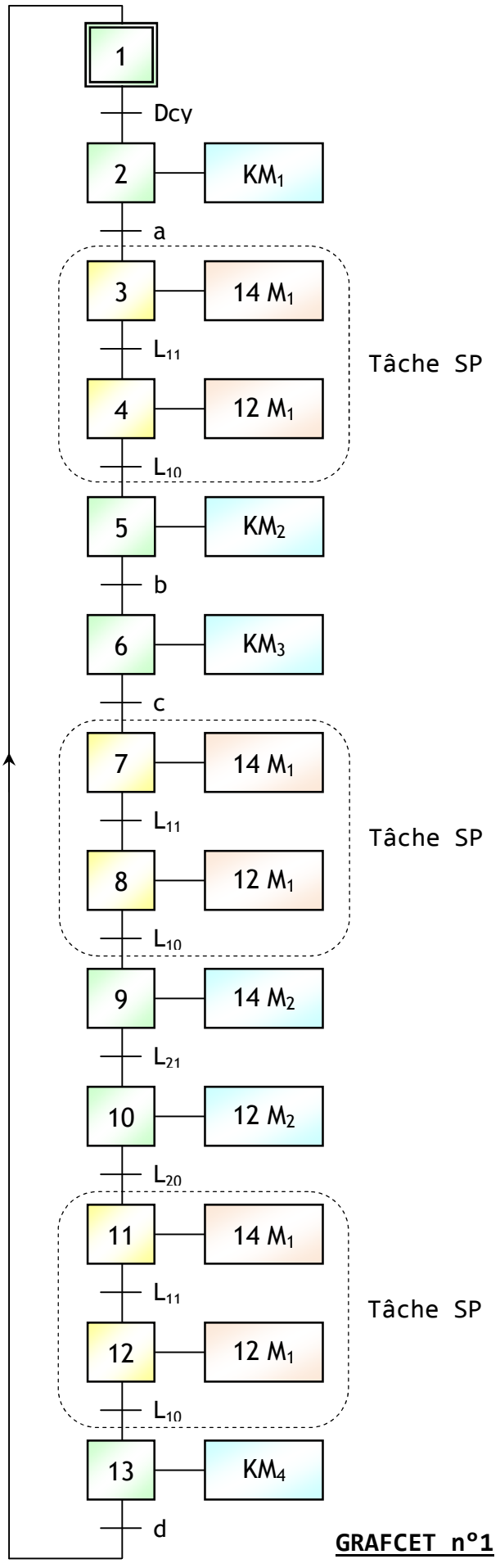
```
QSTATE      =     [Qn ...Q1] ;
    A       =     [Qn ...Q1] ;
    B       =     [Qn ...Q1] ;
    :
    M       =     [Qn ...Q1] ;
STATE_DIAGRAM  QSTATE
    State A :  GOTO B;
    State B :  When Y = = 0 Then C Else A;
    State C :  GOTO D;
    State D :  When Y = = 0 Then A Else C;
```

3.5.4. Constantes spéciales et extensions :

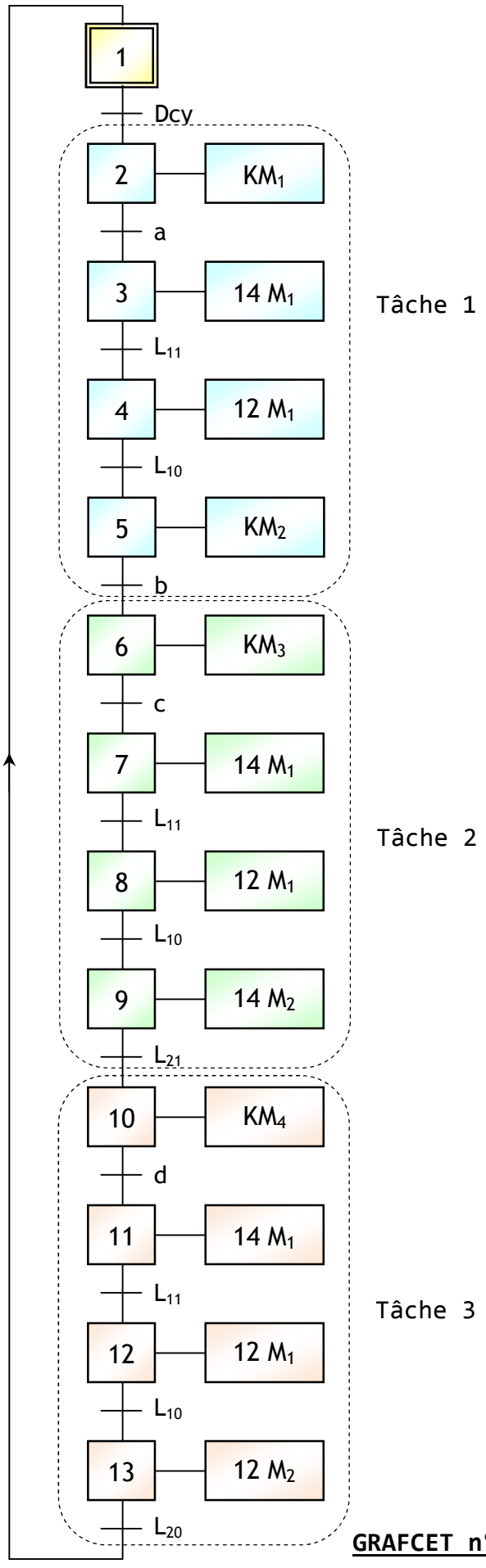
Constantes	
.C.	entrée d'horloge bas-haut-bas (monostable)
.K.	entrée d'horloge haut-bas-haut (monostable)
.U.	front montant d'une entrée horloge
.D.	front descendant d'une entrée horloge
.X.	valeur indéterminée à calculer
.Z.	valeur trois état
.P.	préchargement dans un registre

extensions	
.clk	entrée d'horloge pour bascule
.AR	reset asynchrone de la bascule
.AP	preset asynchrone de la bascule

Soient les deux GRAFCETS ci-dessous :



GRAFCET n°1

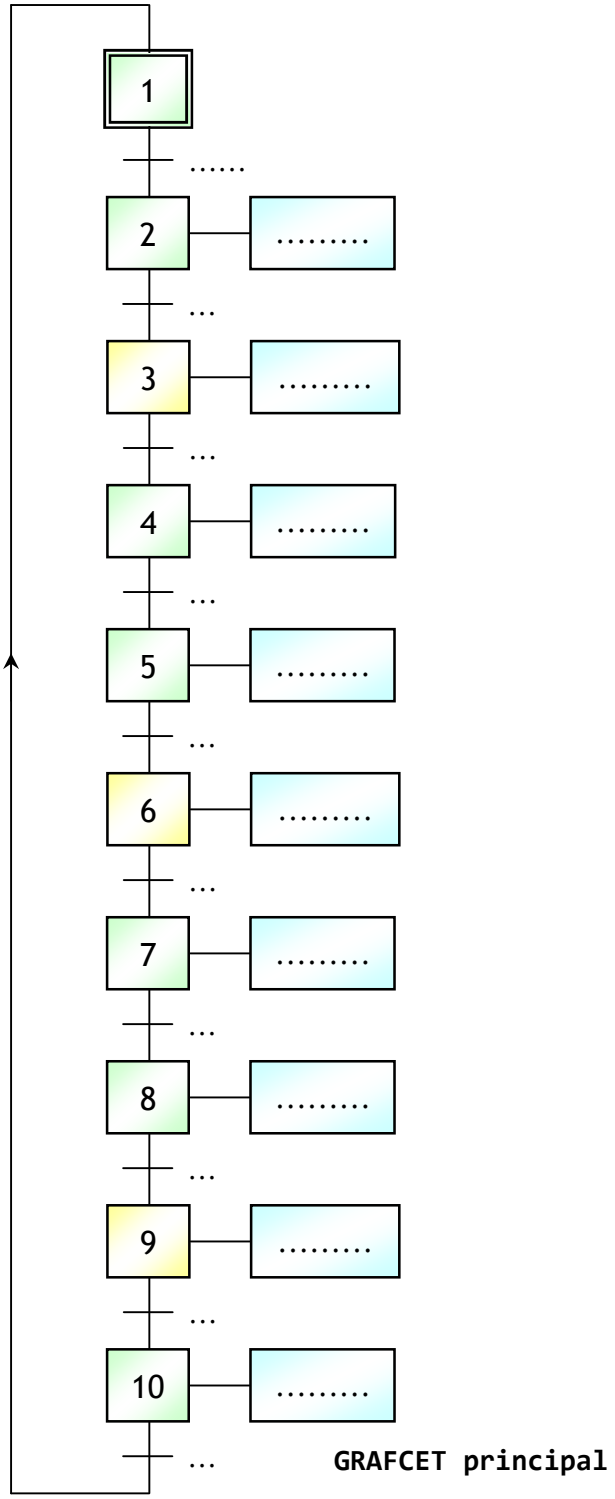


GRAFCET n°2

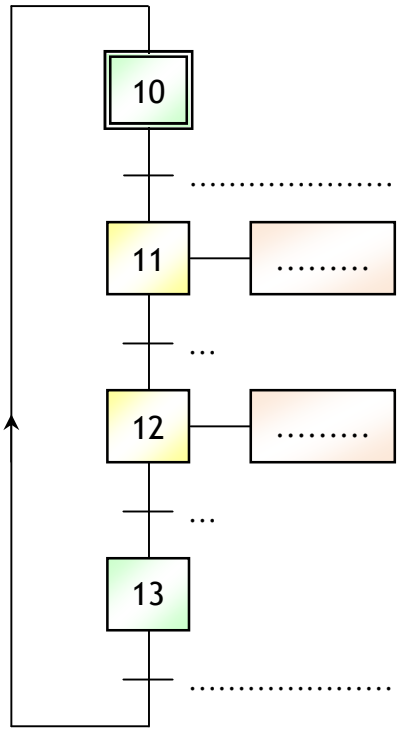
1. Coordination Verticale asynchrone :

1.1. GRAFCET n°1 :

Le GRAFCET n°1 contient une séquence répétitive. On veut réécrire ce GRAFCET d'une manière simplifiée en considérant la séquence répétitive comme une tâche (sous programme) intitulée SP. Compléter le GRAFCET **principal** et le GRAFCET de **tâche SP**.



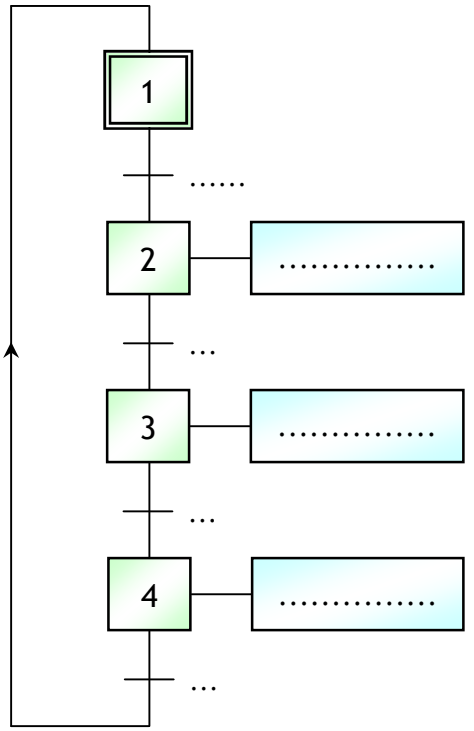
GRAFCET principal



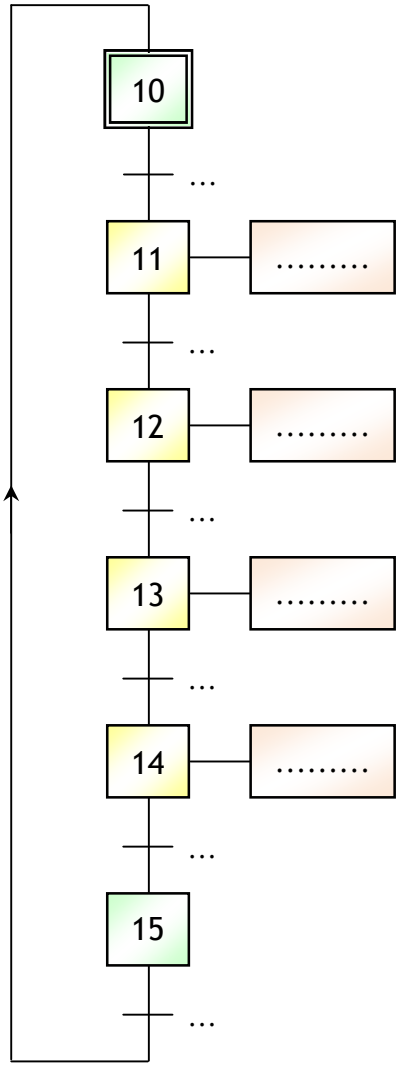
GRAFCET de tâche SP

1.2. GRAFCET n°2 :

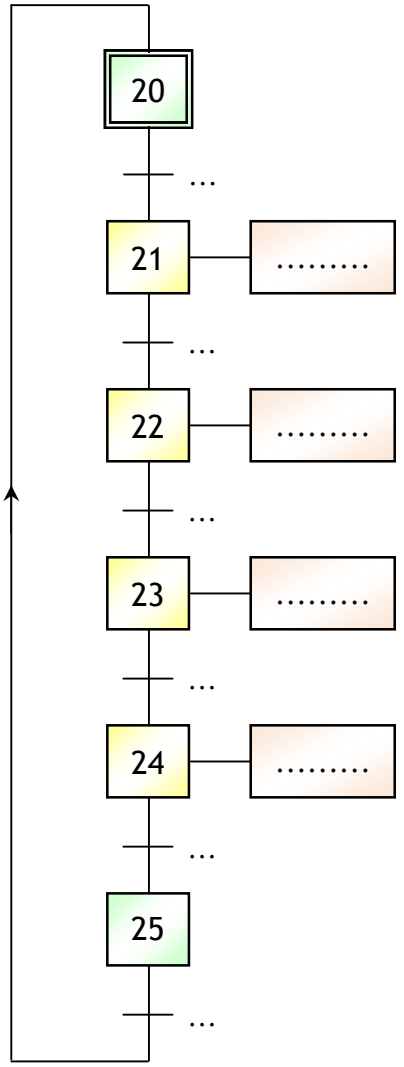
On veut réécrire le GRAFCET n°2 d'une manière simplifiée. Compléter le GRAFCET **principal** et les GRAFCETS de tâche **1, 2 et 3**.



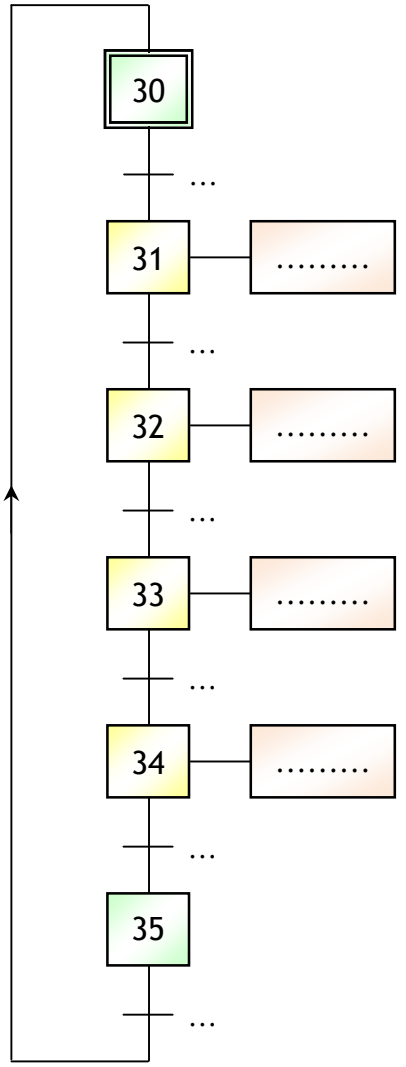
GRAFCET principal



GRAFCET tâche 1



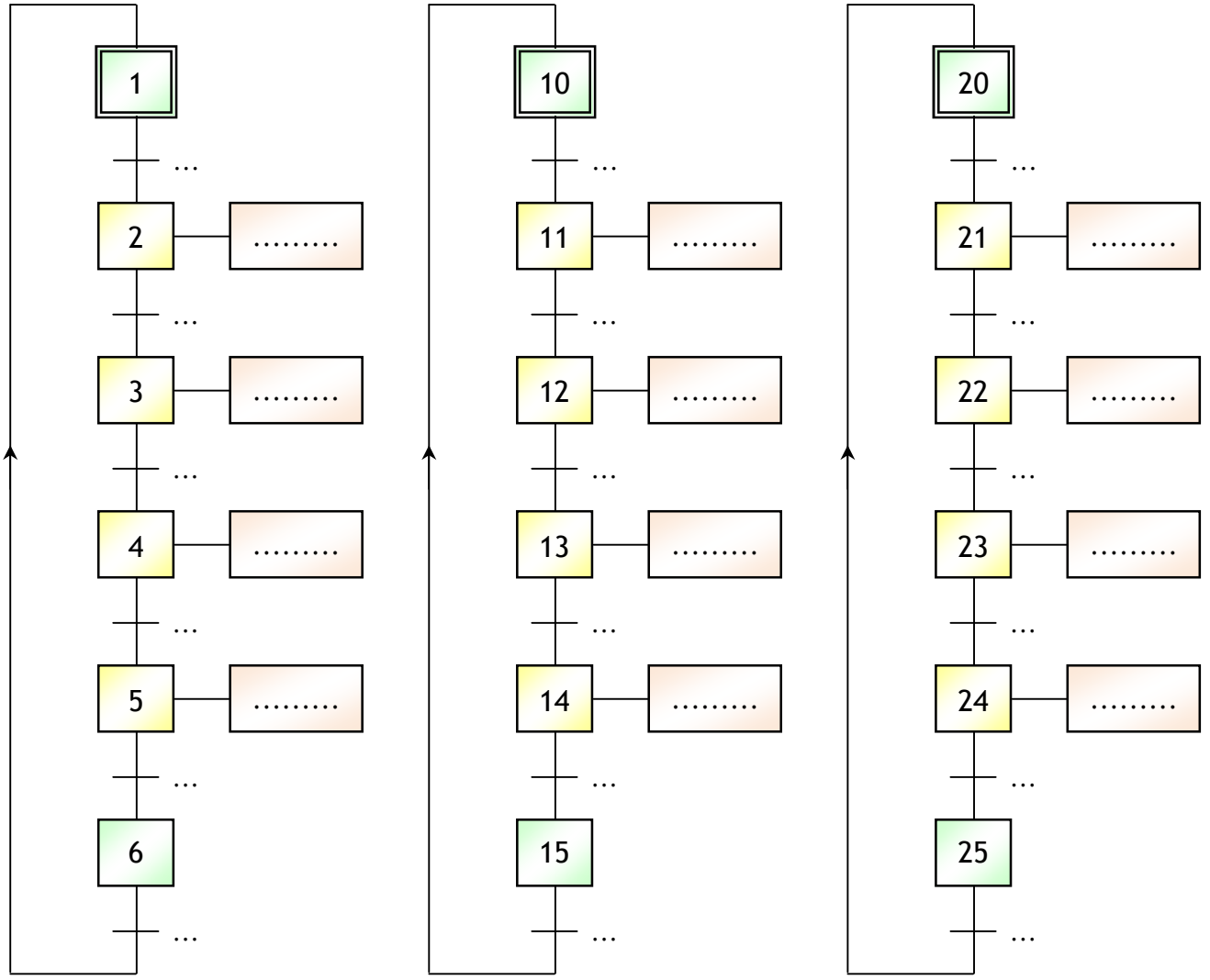
GRAFCET tâche 2



GRAFCET tâche 3

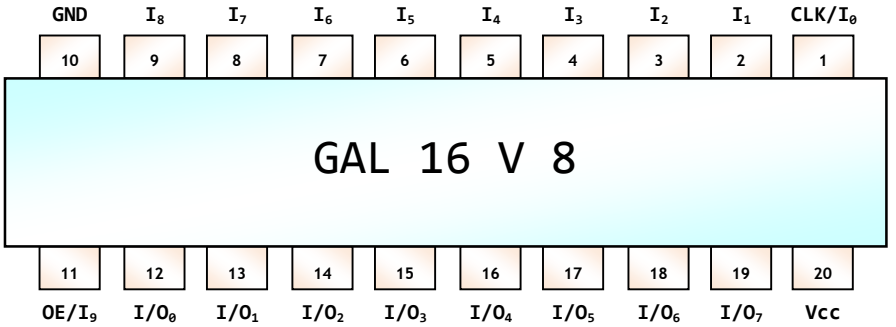
2. Coordination Horizontale :

On veut réécrire ce GRAFCET n°2 d'une autre manière, en le partitionnant en trois GRAFCETS partiels. Compléter, alors les GRAFCETS partiels intitulés **1**, **2** et **3**.



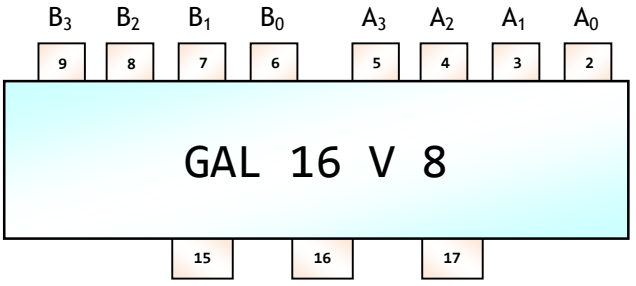
GRAFCET partiel 1 **GRAFCET partiel 2** **GRAFCET partiel 3**

1. Brochage du GAL 16 V 8 :

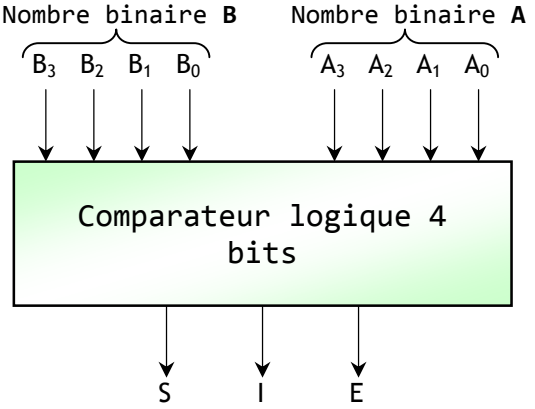


C'est un circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.
La lettre V veut dire que ce circuit est versatile, ce qui veut dire qu'il est possible par programmation de choisir entre une configuration de sortie combinatoire ou séquentielle.

2. Comparateur binaire 4 bits :



S = 1 si A > B
I = 1 si A < B
E = 1 si A = B



Description par équations

.....

.....

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables d'entrée

..... //Définition du bus A

..... //Définition du bus B

..... //Variables de sortie

EQUATIONS

.....

.....

.....

TEST_VECTORS ([A, B] -> [S, E, I]) ;

.....

.....

.....

END comparateur.

Utilisation de When...Then...

MODULE comparateur
TITLE 'comparateur logique 4 bits'
DECLARATIONS
EQUATIONS

.....

.....

.....

TEST_VECTORS ([A, B] -> [S, E, I]) ;

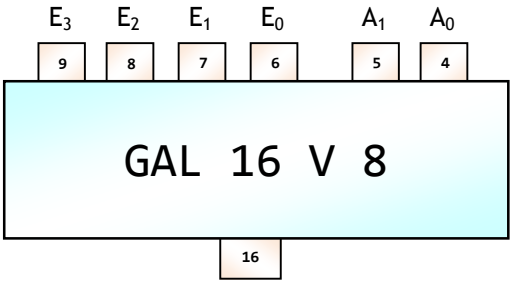
[5, 13] -> .....

[15, 12] -> .....

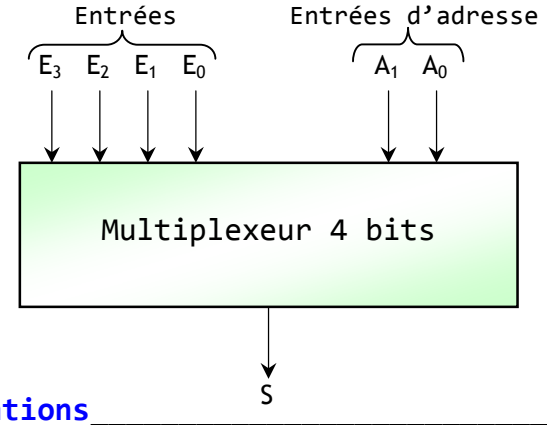
[8, 8] -> .....

END comparateur.

3. Multiplexeur 4 bits :



$S = E_i$ si $A = i$
Avec $A = [A_1, A_0]$



Description par équations

.....

.....

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables d'entrée

..... //Variables de sortie

EQUATIONS

S = ...

TEST_VECTORS .....

[0, 0, 0, 0, .X., .X.] -> .....

[0, 0, 0, 1, 0, 0] -> .....

[0, 0, 1, 0, 0, 1] -> .....

[0, 1, 0, 0, 1, 0] -> .....

[1, 0, 0, 0, 1, 1] -> .....

.....

Utilisation de When...Then...

MODULE .....

TITLE .....

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables de sortie

EQUATIONS

.....

.....

.....

.....

TEST_VECTORS .....

.....

.....

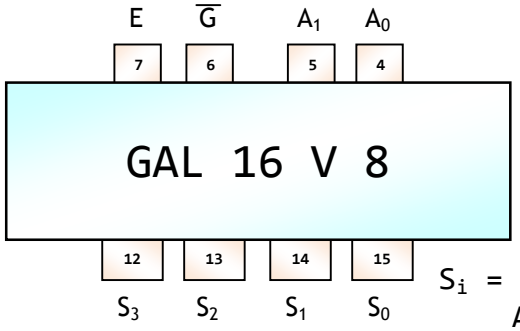
.....

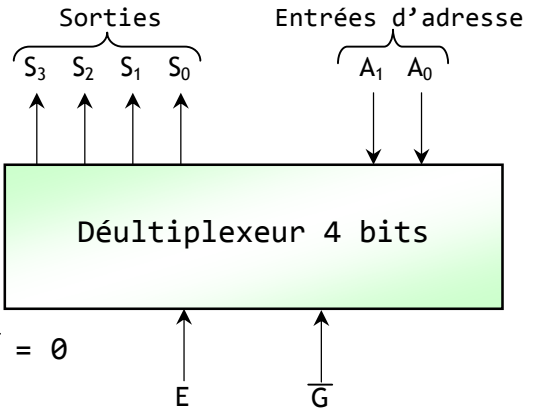
.....

.....

END Multiplexeur.

4. Démultiplexeur 4 bits :





$S_i = E$ si $A = i$ et $\overline{G} = 0$

Avec $A = [A_1, A_0]$

Description par équations

MODULE Demultiplexeur

TITLE 'Démultiplexeur 4 bits'

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables de sortie

EQUATIONS

S_3 = ...

S_2 = ...

S_1 = ...

S_0 = ...

TEST_VECTORS .....

.....

.....

.....

.....

.....

.....

END Demultiplexeur.

Utilisation de When...Then...

.....

.....

DECLARATIONS

..... //PLD à programmer

E, G, A_1, A_0 pin 7, 6, 5, 4; //Variables d'entrée

S_3, S_2, S_1, S_0 pin 12, 13, 14, 15 istype 'com, buffer '; //Variables de sortie

..... //Définition du bus A

..... //Définition du bus S

EQUATIONS

.....

.....

.....

.....

TEST_VECTORS ([G, E, A] -> [S]) ;

[1,.X.,.X.] -> [0] ;

.....

.....

.....

.....

.....

.....

Description par table de vérité

MODULE Démultiplexeur

TITLE 'Démultiplexeur 4 bits'

DECLARATIONS

Démultiplexeur device 'P16V8'; //PLD à programmer

E, G, A_1, A_0 pin 7, 6, 5, 4; //Variables d'entrée

S_3, S_2, S_1, S_0 pin 12, 13, 14, 15 istype 'com, buffer '; //Variables de sortie

TRUTH_TABLE ([G, E, A_1, A_0] -> [S_3, S_2, S_1, S_0]) ;

.....

.....

.....

.....

.....

.....

TEST_VECTORS ([G, E, A_1, A_0] -> [S_3, S_2, S_1, S_0]) ;

.....

.....

.....

.....

.....

.....

END Démultiplexeur.

Description par table de vérité (autre solution)

.....

.....

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables de sortie

TRUTH_TABLE ([G, A_1, A_0] -> [S_3, S_2, S_1, S_0]) ;

.....

.....

.....

.....

.....

TEST_VECTORS ([G, E, A_1, A_0] -> [S_3, S_2, S_1, S_0]) ;

.....

.....

.....

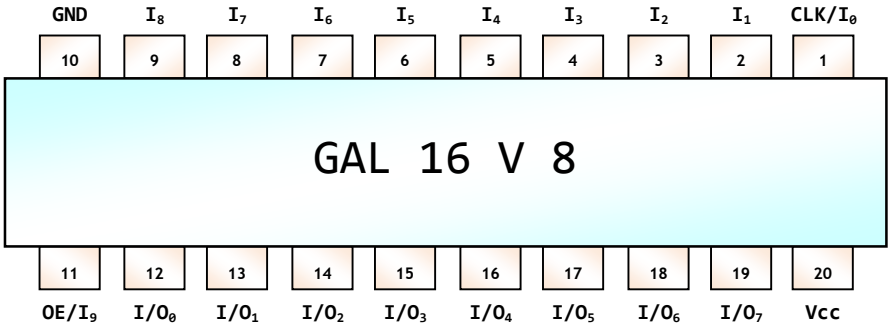
.....

.....

.....

.....

1. Brochage du GAL 16 V 8 :

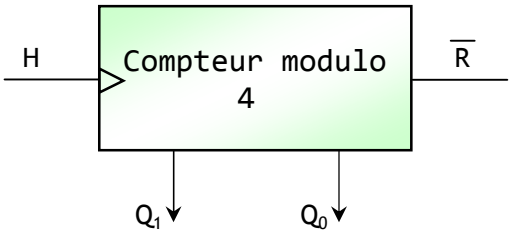


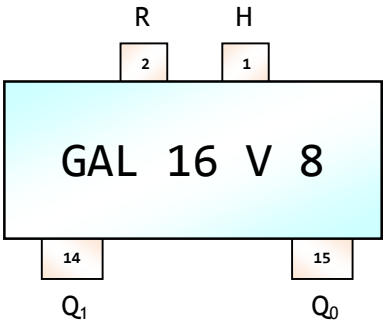
C'est un circuit logique programmable et effaçable électriquement.

Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.

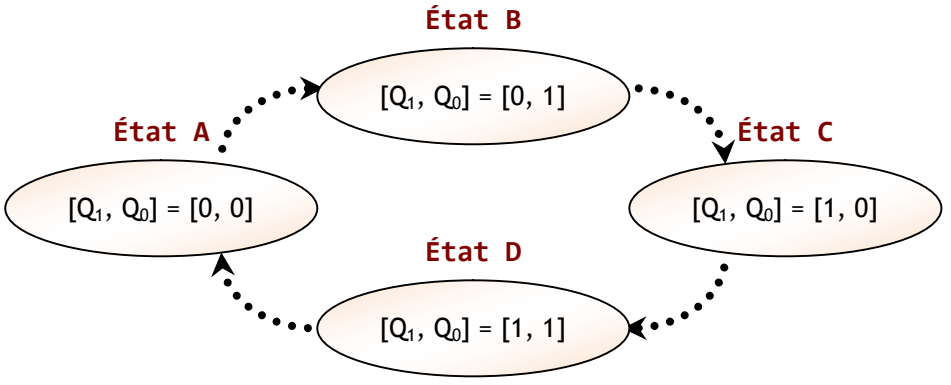
La lettre V veut dire que ce circuit est versatile, ce qui veut dire qu'il est possible par programmation de choisir entre une configuration de sortie combinatoire ou séquentielle.

2. Compteur modulo 4 :





Description par diagramme d'état



MODULE .....

TITLE .....

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables de sortie

EQUATIONS

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TEST_VECTORS ([R, H,] -> [Q_1, Q_0])

.....

.....

.....

.....

.....

.....

Description par équation

MODULE Compteur

TITLE 'Compteur modulo 4'

DECLARATIONS

Compteur device 'P16V8' ; //PLD à programmer

H, R pin 1, 2 ; //Variables d'entrée

..... //Variables de sortie

..... //Définition du bus Q

EQUATIONS

.....

.....

.....

TEST_VECTORS ([R, H,] -> [Q_1, Q_0])

[0, .X.] -> [0, 0] ;

[1, .C.] -> [0, 1] ;

[1, .C.] -> [1, 0] ;

.....

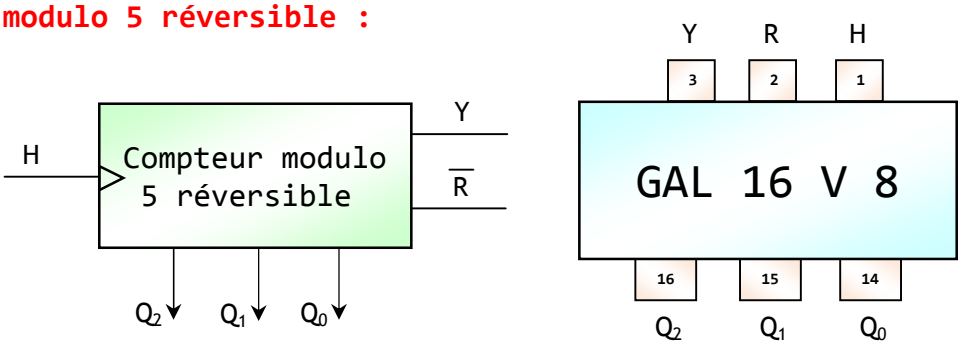
.....

END Compteur.

Description par table de vérité

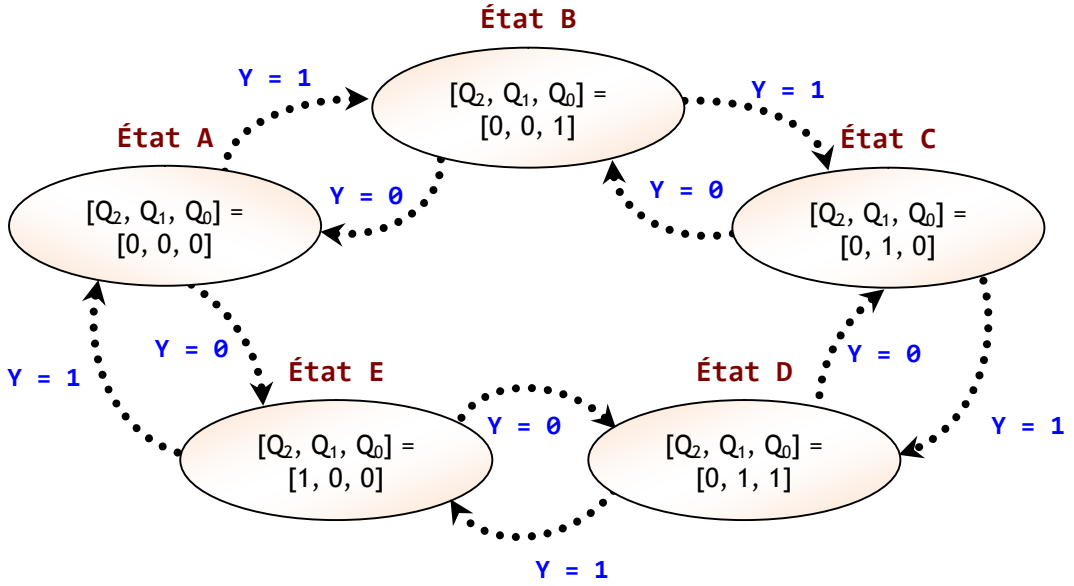
```
MODULE Compteur
TITLE 'Compteur  modulo 4'
DECLARATIONS
        Compteur device 'P16V8' ;                    //PLD à programmer
        H, R pin  1, 2 ;                             //Variables d'entrée
        Q0, Q1 pin 15, 14 Istype 'reg, buffer ';     //Variables de sortie
EQUATIONS
        ……
        ……
……
                ……
                ……
                ……
                ……
                ……
TEST_VECTORS    ([R, H,]    ->    [Q1, Q0])
                [0, .X.]    ->    [0, 0] ;
                [1, .C.]    ->    [0, 1] ;
                [1, .C.]    ->    [1, 0] ;
                [1, .C.]    ->    [1, 1] ;
                [1, .C.]    ->    [0, 0] ;
END Compteur.
```

3. Compteur modulo 5 réversible :



Conception d'un compteur modulo 5 réversible suivant une entrée de sélection du mode de fonctionnement Y = comptage/$\overline{\text{décomptage}}$; par le GAL P16V8.

- Si Y = 1 c'est le comptage de 000 à 100 ;
- Si Y = 0 c'est le décomptage de 100 à 000.



Description par diagramme d'état

MODULE .....

TITLE .....

DECLARATIONS

..... //PLD à programmer

..... //Variables d'entrée

..... //Variables de sortie

EQUATIONS

.....

.....

QSTATE = [Q_2, Q_1, Q_0] ;

.....

.....

.....

.....

.....

STATE_DIAGRAM QSTATE

.....

.....

.....

.....

.....

TEST_VECTORS ([R, H, Y] -> [Q_2, Q_1, Q_0])

[0, .X., .X.] -> .....

[1, .C., 1] -> .....

[1, .C., 1] -> .....
[1, .C., 1] -> .....
[1, .C., 1] -> .....
[1, .C., 1] -> .....
[1, .C., 0] -> .....
[1, .C., 0] -> .....
[1, .C., 0] -> .....
[1, .C., 0] -> .....
[1, .C., 0] -> .....

.....

Description par table de vérité

.....

.....

DECLARATIONS

..... //PLD à programmer
..... //Variables d'entrée
..... //Variables de sortie

EQUATIONS

.....
.....

TRUTH_TABLE ([R, Y, Q_2, Q_1, Q_0] :> [Q_2, Q_1, Q_0])

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

TEST_VECTORS ([R, H, Y] -> [Q_2, Q_1, Q_0])

[0, .X., .X.] -> [0, 0, 0] ;
[1, .C., 1] -> [0, 0, 1] ;
[1, .C., 1] -> [0, 1, 0] ;

```
                    [1, .C., 1]     ->    [0, 1, 1] ;
                    [1, .C., 1]     ->    [1, 0, 0] ;
                    [1, .C., 1]     ->    [0, 0, 0] ;
                    [1, .C., 0]     ->    [1, 0, 0] ;
                    [1, .C., 0]     ->    [0, 1, 1] ;
                    [1, .C., 0]     ->    [0, 1, 0] ;
                    [1, .C., 0]     ->    [0, 0, 1] ;
                    [1, .C., 0]     ->    [0, 0, 0] ;
END Compteur_reversible.
```

Description par équation

```
MODULE Compteur_reversible
TITLE 'Compteur  modulo 5'
DECLARATIONS
    Compteur device 'P16V8' ;                          //PLD à programmer
    H, R, Y pin  1, 2, 3 ;                             //Variables d'entrée
    Q0, Q1, Q2 pin 14, 15, 16 Istype 'reg, buffer ';   //Variables de sortie
    ..........................................................  //Définition du bus Q
EQUATIONS
    ......................................
    ......................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
TEST_VECTORS    ([R, H, Y]      ->    [Q2, Q1, Q0])
                  [0, .X., .X.]   ->    [0, 0, 0] ;
                  [1, .C., 1]     ->    [0, 0, 1] ;
                  [1, .C., 1]     ->    [0, 1, 0] ;
                  [1, .C., 1]     ->    [0, 1, 1] ;
                  [1, .C., 1]     ->    [1, 0, 0] ;
                  [1, .C., 1]     ->    [0, 0, 0] ;
                  [1, .C., 0]     ->    [1, 0, 0] ;
                  [1, .C., 0]     ->    [0, 1, 1] ;
                  [1, .C., 0]     ->    [0, 1, 0] ;
                  [1, .C., 0]     ->    [0, 0, 1] ;
                  [1, .C., 0]     ->    [0, 0, 0] ;
END Compteur_reversible.
```