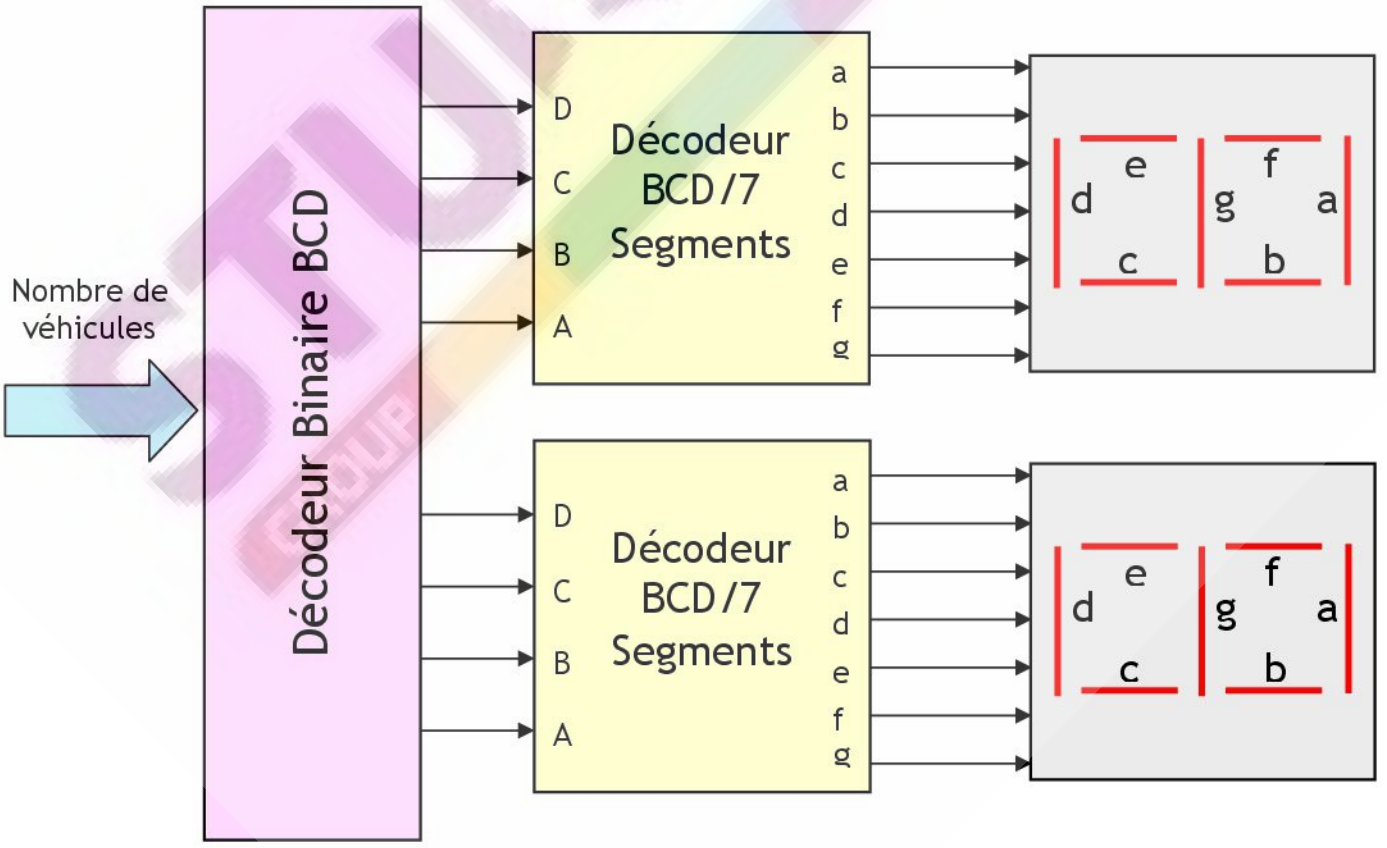
DRES 08

AFFICHAGE DU NOMBRE DE VÉHICULES

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.

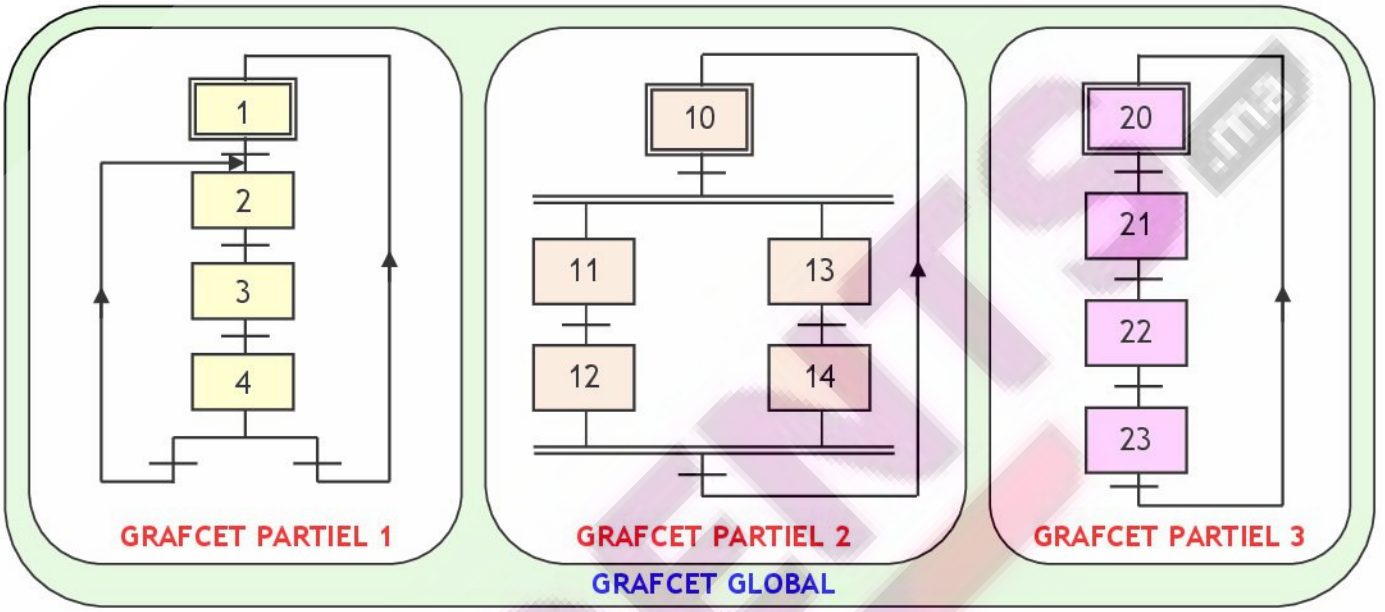
AFFICHAGE DU NOMBRE DE VÉHICULES



GRAFCET 'NOTION DE TÂCHE'

1- Notion de GRAFCET partiel :

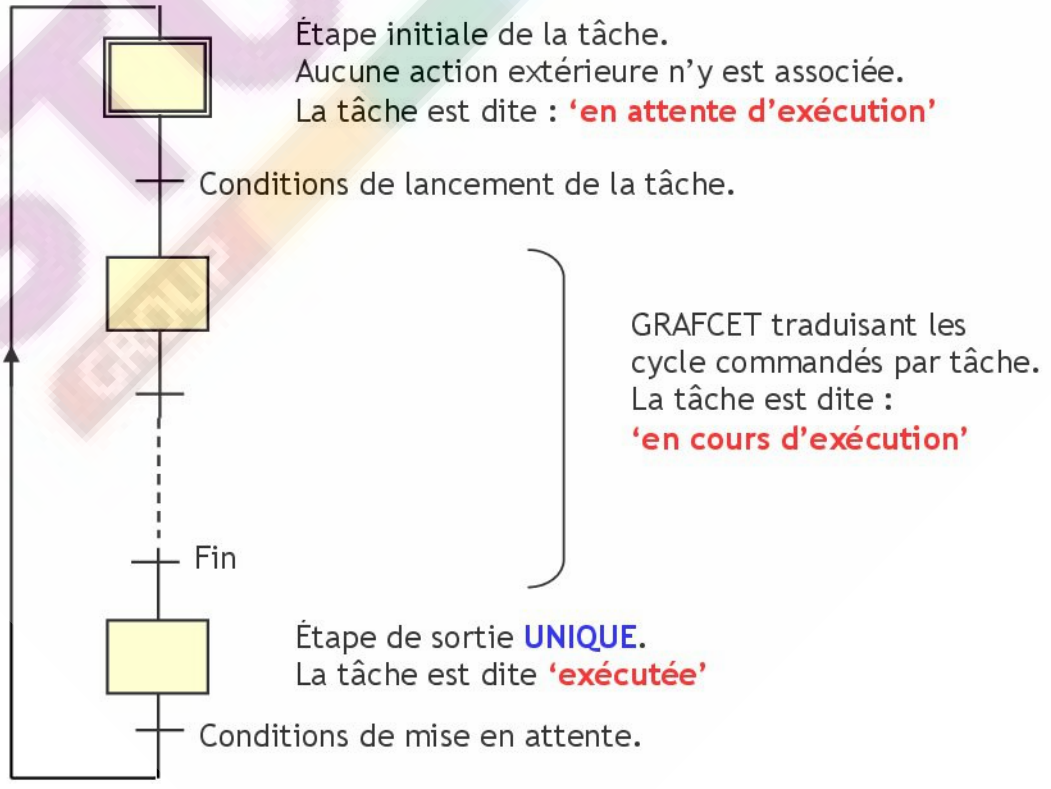
Constitué d'un ou plusieurs GRAFCETS connexes. Un GRAFCET partiel résulte d'une partition, selon des critères méthodologiques, du GRAFCET global décrivant le comportement de la partie séquentielle d'un système.



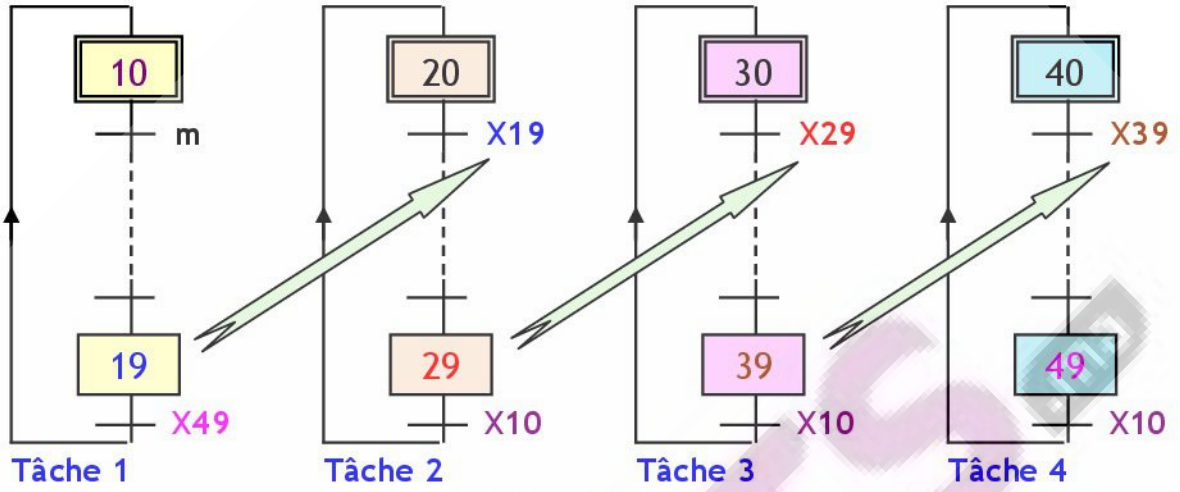
Le GRAFCET **global** est constitué des GRAFCETS partiels **G1**, **G2** et **G3**.

2- Synchronisation de GRAFCET :

2.1- Notion de GRAFCET de Tâche :

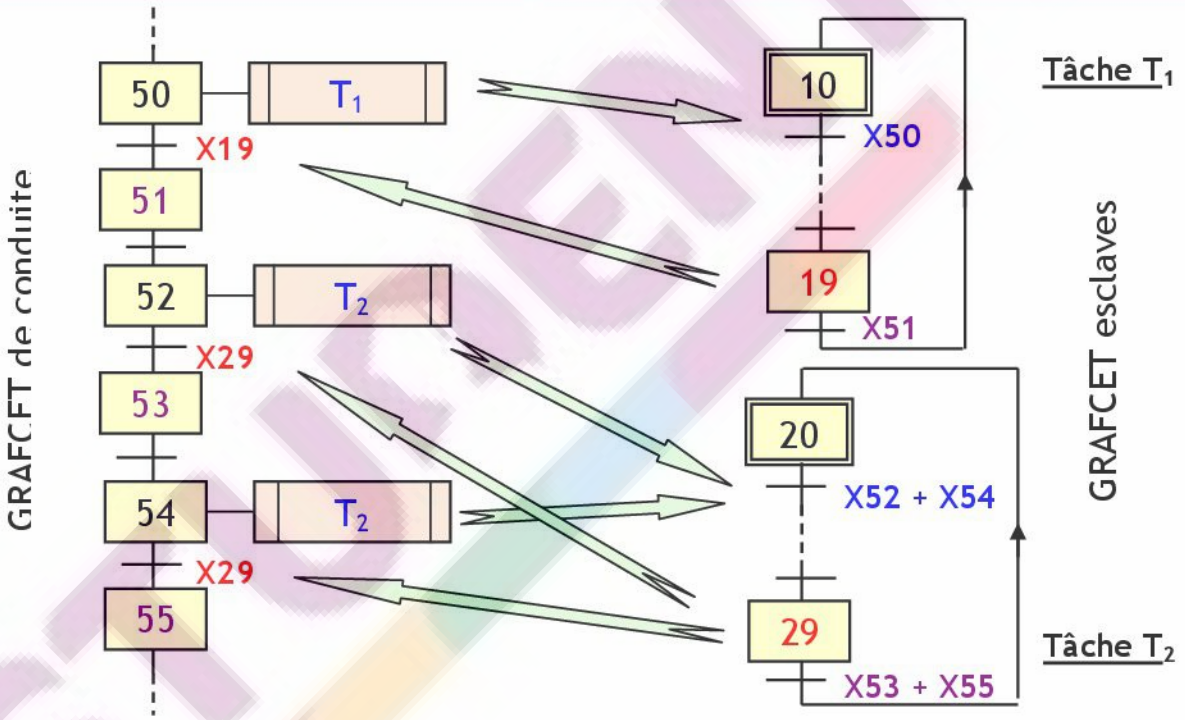


2.2- Coordination horizontale :



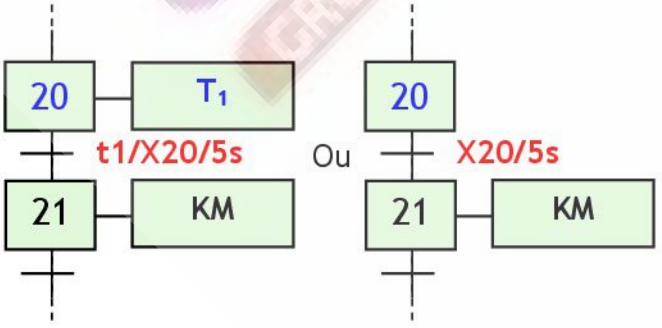
1 seule tâche à la fois

2.3- Coordination Verticale asynchrone :



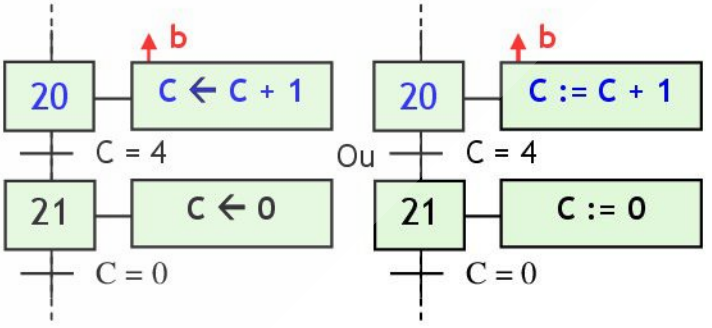
3- Différents types d'action :

3.1- Temporisations :



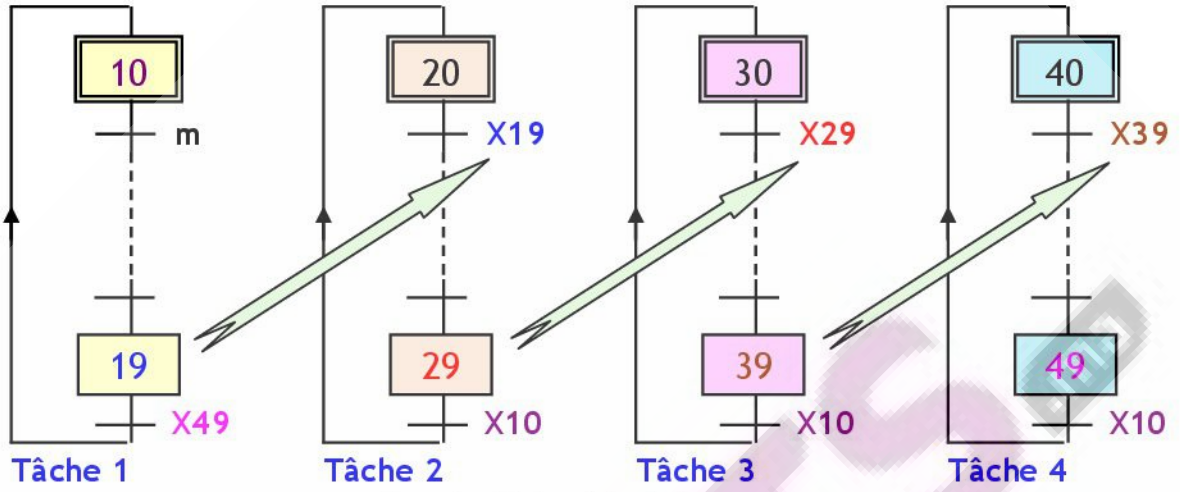
La transition 20 - 21 est franchie lorsque la temporisation, démarrée à l'étape 20 est écoulée, soit au bout de 5s

3.2- Comptage :



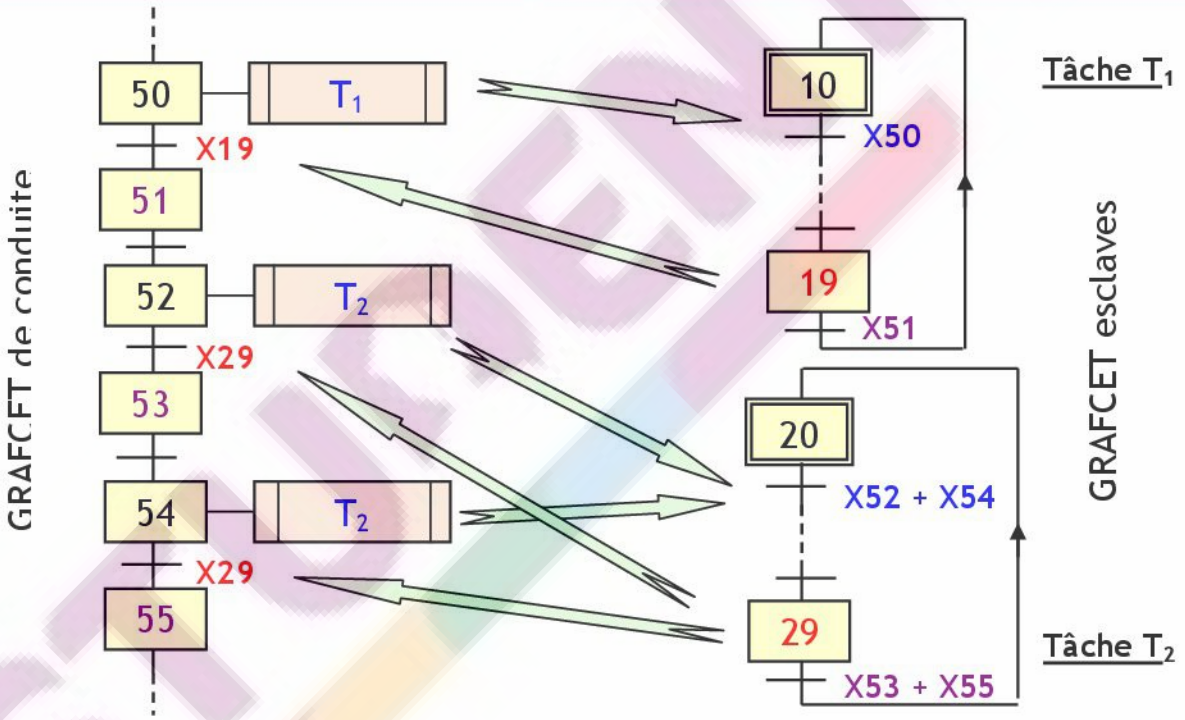
La transition 20 - 21 est franchie lorsque le contenu du compteur C est égal à 4. Le compteur est incrémenté sur front montant du signal b. Il est mis à zéro à l'étape 21.

2.2- Coordination horizontale :



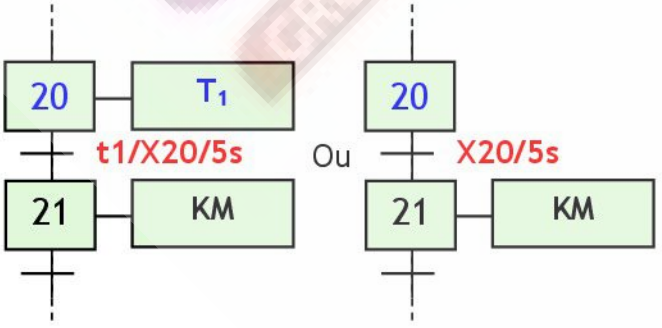
1 seule tâche à la fois

2.3- Coordination Verticale asynchrone :



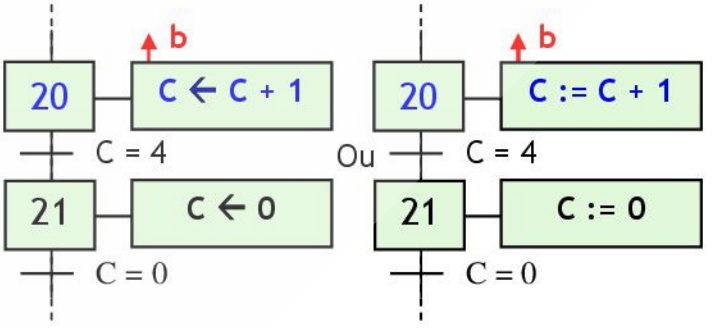
3- Différents types d'action :

3.1- Temporisations :



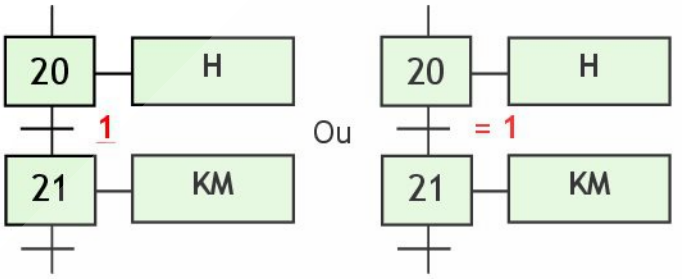
La transition 20 - 21 est franchie lorsque la temporisation, démarrée à l'étape 20 est écoulée, soit au bout de 5s

3.2- Comptage :



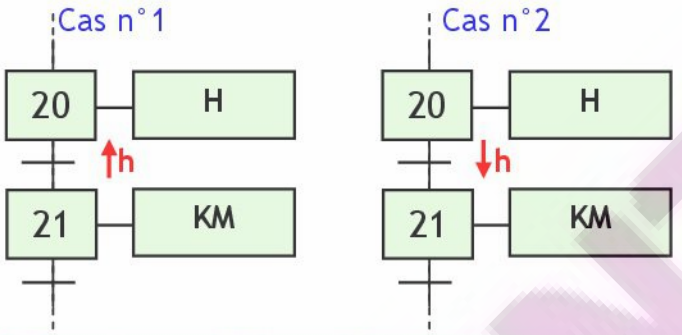
La transition 20 - 21 est franchie lorsque le contenu du compteur C est égal à 4. Le compteur est incrémenté sur front montant du signal b. Il est mis à zéro à l'étape 21.

3.3- Réceptivité toujours vraie :



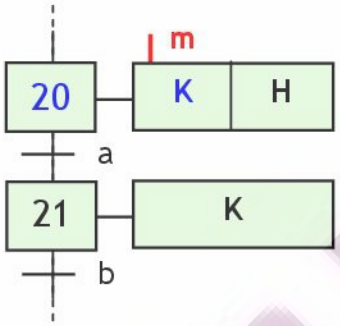
L'étape 21 est active, à l'activation de l'étape 20

3.4- Évènements (fronts) :

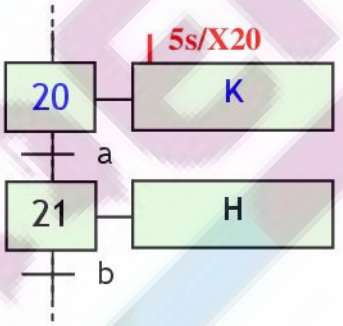


La transition 20 - 21 est franchie lors d'un front montant sur h (cas n°1), ou lors d'un front descendant sur h (cas n°2).

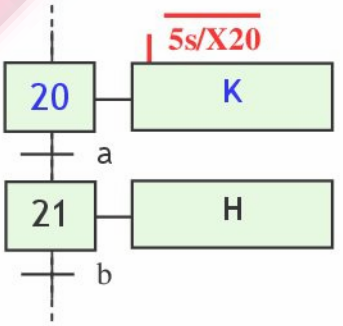
3.5- Action conditionnelle :



L'action K devient effective à l'étape 20, lorsque la condition m est vraie.
K = X20. m + X21

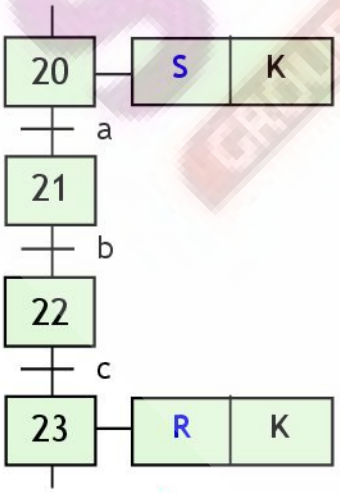


L'action K devient effective à l'étape 20, après 5 secondes

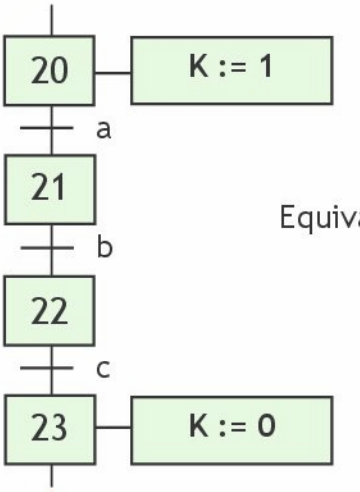


L'action K devient effective à l'étape 20, et dure 5 secondes

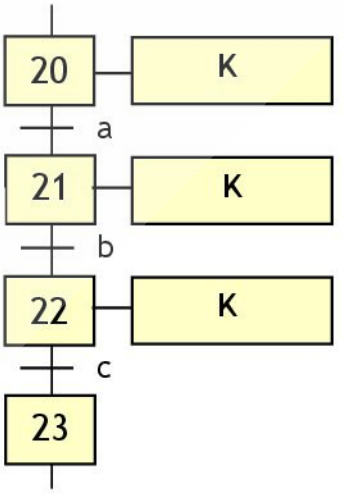
3.6- Action mémorisée :



OU



Equivalent à



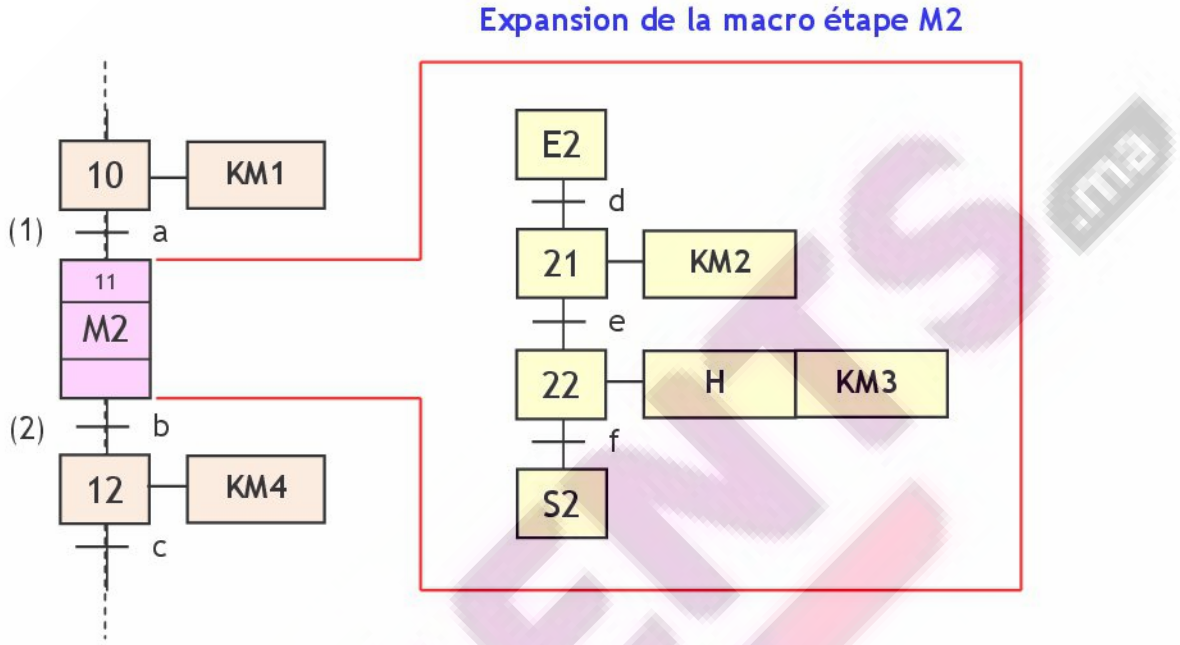
Mise à 1 de l'action par la lettre S (set)
Mise à 0 de l'action par la lettre R (reset)

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

4- Notion de Macro- étape :

Avec la notion de macro représentation, on se donne le moyen de reporter à plus tard ou sur une autre page la description détaillée de certaines séquences.

La macro-étape est la représentation unique d'un ensemble d'étapes et de transitions nommé expansion 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.

Le franchissement de la transition (1) active l'étape E2.

La transition (2) ne sera validée que lorsque l'étape S2 sera active.

Le franchissement de la transition (2) désactive l'étape S2.

EEPROM DU 16 F 84

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 88H et 89H. La durée d'écriture d'un octet est de l'ordre de 10ms.

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 :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 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 échoué.
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- Ecriture 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 EECONN1 à **1.**
- Lire le contenu du registre **EEDATA.**

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

4.2- 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 EECONN1 à **1** pour autoriser l'écriture.
- Placer **0x55** dans **EECON2.**
- Placer **0xAA** dans **EECON2.**
- Mettre le bit **WR** de EECONN1 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
     .....        ; Bank 0
```

TIMER DU 16 F 84

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 TMRO :

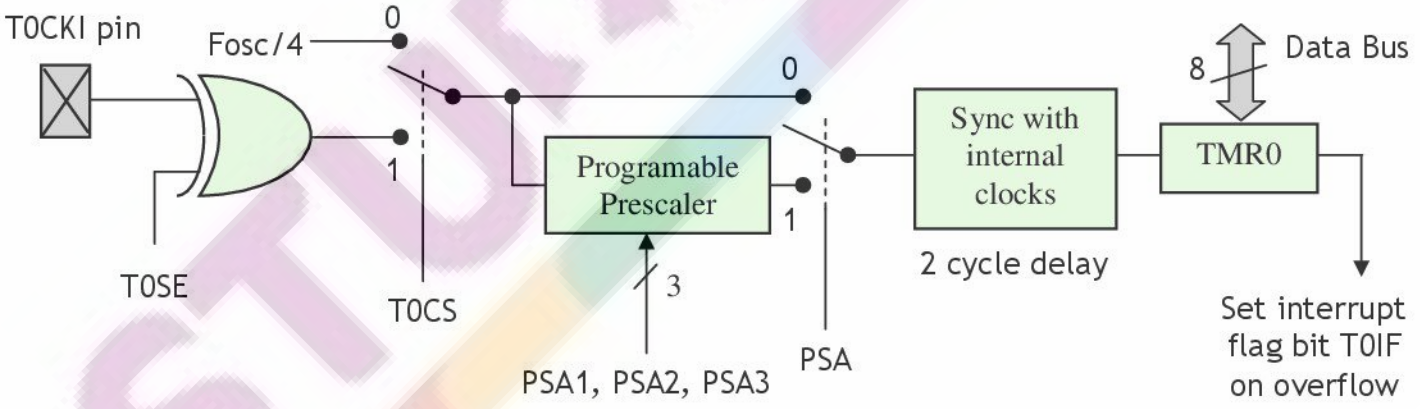
Ce registre de 8 bits s'incrémente de "**1**" a chaque impulsion de l'horloge **interne** (**Fosc/4**) ou par une horloge **externe** appliquée sur la broche **TOCKI/ RA4.** Il est associé au module Timer/Compteur. Ce registre se trouve à l'adresse 01H.

❖ 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 81H.Ce registre se trouve à l'adresse 81H.

3- Fonctionnement :

3.1- Schéma simplifié :



3.2- Structure du registre OPTION :

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

BIT 7: RBPU -PORTB Pull-Up-.

RBPU=1 Le "tirage au plus" interne du PORT B est désactivé.

RBPU=0 Le "tirage au plus" interne du PORT B est activé.

BIT 6: INTEDG -INTerrupt EDGe-

INTEDG=1 alors la broche RBO/INT génère une interruption sur un front montant.

INTEDG=0 alors la broche RBO/INT génère une interruption sur un front descendant.

BIT 5: **TOCS -TMRO Clock Source-**

Il permet de sélectionner le mode de fonctionnement du Timer/Compteur.

TOCS=1 sélection de l'horloge externe (broche RA4) qui correspond au COMPTEUR.

TOCS=0 sélection de l'horloge interne et permet au module de travailler en mode TIMER.

BIT 4: **TOSE -TMRO Source Edge-**

Ce bit détermine sur quel front -montant ou descendant- l'entrée RA4 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 0, 1, 2: **PS0, PS1, PS2 - Prescaler Select -**

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

PS2	PS1	PS0	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

Remarque :

Lorsque le TIMER0 déborde, le BIT TOIF du registre INTCON passe à 1; après utilisation, n'oublier pas de remettre TOIF à zéro.

4- Le WATCHDOG :

C'est un compteur incrémenté en permanence - Free running - par l'horloge interne. Ce compteur 8 bits, quand il arrive à **FF** - WDT Time Out - est capable de réinitialiser le microcontrôleur.

La durée d'un cycle de comptage est réglable grâce au Prescaler partagé avec le Timer 0.

- **Caractéristiques:**
  - ✓ Désactivation au démarrage ;
  - ✓ Remis à 0 sur appel de CLRWTD ;
  - ✓ Redémarre le PIC placé en mode SLEEP: Signalé par bit TO de statut
- **Utilisation**
  - ✓ Détection boucle dans programme (possible sur faute électrique ou du programmeur)
  - ✓ Réveil après SLEEP.

Interruptions du 16 F 84

1- Interruptions du PIC 16 F 84 :

Le PIC 16 F 84 possède 4 sources d'interruption :

- Changement d'état du PORTB (RB4 à RB7)
- Front montant ou descendant sur la broche RB0/INT
- Dépassement du registre TMRO (passage de FF à 00)
- Déroulement normal d'un cycle d'écriture dans L'EEPROM.

Ces interruptions sont validées :

- Globalement par le bit GIE du registre INTCON.
- Localement par les bits EEIE, RTIE, INTE et RBIE du registre INTCON.

4 drapeaux pouvant provoquer l'interruption correspondante :

- EEIF du registre EECON1.
- T0IF, INTF, RBIF du registre INTCON.

2- Service d'une interruption :

Le service d'une interruption comporte les actions suivantes :

- Le drapeau correspondant passe a 1.
- PC est empilé puis affecté de l'adresse 004.
- Le masque GIE est automatiquement mis a 0 pour interdire d'autres interruptions.
- Fin (instruction RETIE)
- PC est dépilé
- GIE est remis a 1

Le programme doit:

- Identifier la source de l'interruption en consultant les drapeaux.
- Remettre à 0 le drapeau qui a provoqué l'interruption.
- Sauvegarder éventuellement certains registres.

3- Le registre INTCON:

Le registre INTCON contient tous les bits de validation de chaque source d'interruption ainsi que leur drapeau (Flag). Les drapeaux doivent être mise à 0 après l'interruption.

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF

Bit 0: **RBIF -RB port change Interrupt Flag-**

C'est un drapeau d'interruption qui indique un changement d'état du PORT B (RB4 à RB7).

RBIF=1 une broche (RB4 à RB7) à changé d'état.

RBIF=0 pas de changement d'état.

Bit 1: **INTF -INT Interrupt Flag-**

Drapeau d'interruption de l'entrée RB0.

RBIF=1 une interruption est apparue.

RBIF=0 pas d'interruption.

Bit 2: **TOIF -TMRO Overflow Interrupt Flag-**

Ce drapeau indique un dépassement du registre TMRO (passage de FF à 00).

TOIF=1 dépassement de TMRO.

TOIF=0 pas de dépassement.

Bit 3: **RBIE -RB Interrupt Enable-**

Valide ou non de l'interruption généré par le changement d'état du PORT B (RB4 à RB7).

RBIE=1 Autorise l'interruption.

RBIE=0 Les changement d'état du PORTB (RB4 à RB7) ne généront pas d'interruption.

Bit 4: **INTE -INT Interrupt Enable-**

Valide ou non de l'interruption généré par la broche RB0/INT

INTE=1 Valide l'interruption INT.

INTE=0 Pas d'interruption provenant de INT.

BIT 5: **TOIE -TMRO Overflow Interrupt Enable-**

Autorise ou non l'interruption provoqué par le dépassement du registre TMRO (passage de FF à 00)

TOIE=1 Valide l'interruption.

TOIE=0 Pas d'interruption provenant du registre TMRO.

BIT 6: **EEIE -EE write Interrupt Enable-**

Autorise ou non une interruption lorsqu'un cycle d'écriture dans L'EEPROM s'est déroulé normalement.

EEIE=1 Génère une interruption lorsqu'une cycle d'écriture c'est déroulé normalement.

EEIE=0 Pas d'interruption.

BIT 7: **GIE -Global Interrupt Enable-**

Autorise ou non toutes les interruptions.

GIE=1 Toutes les interruptions sont prises en compte par le microcontrôleur.

GIE=0 Aucune interruption ne sera validée.

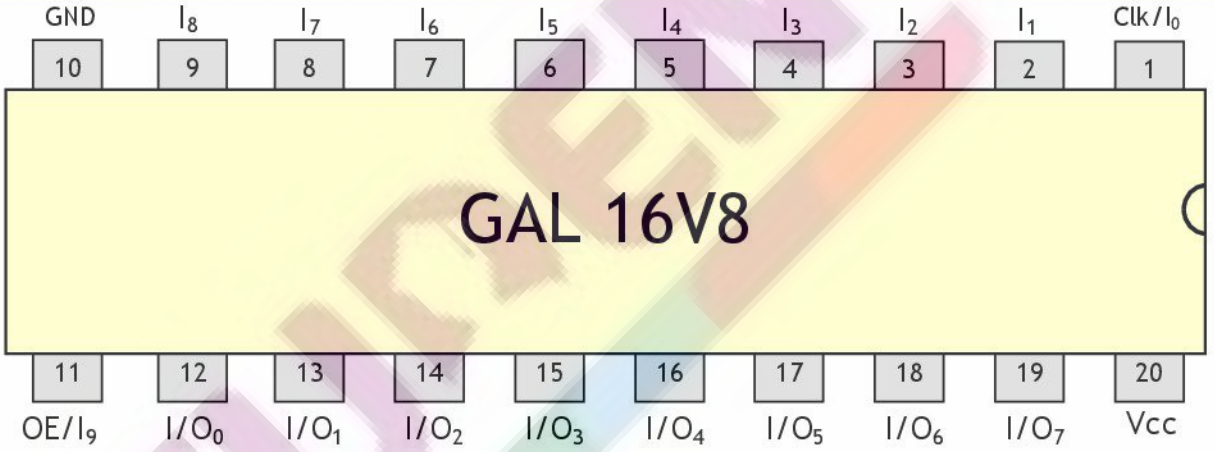
Programmation des PLD

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 circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmable.

3- Initiation au langage ABEL :

3.1- Structure :

Section d'en-tête

Section des déclarations

Section de description

Section de test

Déclaration de fin de fichier

```
MODULE logique1
TITLE 'alarme de phares'
DECLARATIONS
        C, L, P pin 18, 19, 20; // Variables d'entrée
        S1, S2 pin 16, 17 istype 'com'; // Variables de sortie
EQUATIONS
        S1 = !C & L & P;
        S2 = P;
TEST_VECTORS
        ([C, L, P ] -> [S1, S2]);
        [ 0, 0, 0 ] -> [.X.,.X.];
        [ 0, 1, 1 ] -> [.X.,.X.];
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 reconnue par le mot clé **MODULE** suivi du nom du fichier ; on peut aussi ajouter au programme un titre précédé du mot **TITLE** ; la section du 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 : S1, S2 pin 16, 17 istype 'com, buffer';

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

Exemple :
N = [E3, E2, E1, E0] ;
S = [Q7, Q6, Q5, Q4, Q3, Q2, Q1, Q0] ;

- ✓ Le mot N est constitué des bits E3, E2, E1 et E0.
- ✓ E3 constitue le MSB (bit de poids fort) et E0 le LSB (bit de poids faible).
- ✓ Le mot S est constitué des bits Q7, Q6, Q5, Q4, Q3, Q2, Q1 et Q0.
- ✓ Q7 constitue le MSB et Q0 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 zéro 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)

A- 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;
```

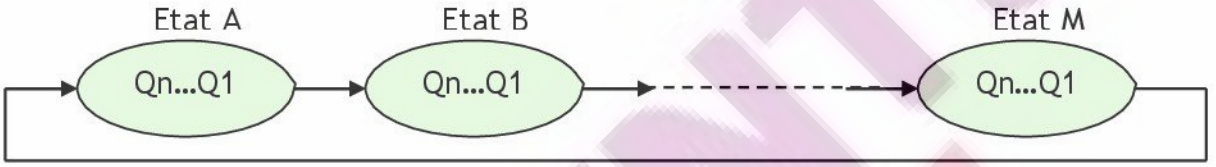
B- Description par table de vérité :

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] -> [S1, S2, S3])
            [0, 0] -> [0, 1, 0];
            [0, 1] -> [0, 0, 1];
            [1, 0] -> [1, 0, 0];
            [1, 1] -> [1, 1, 1];
```

C- 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 transition 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;
```

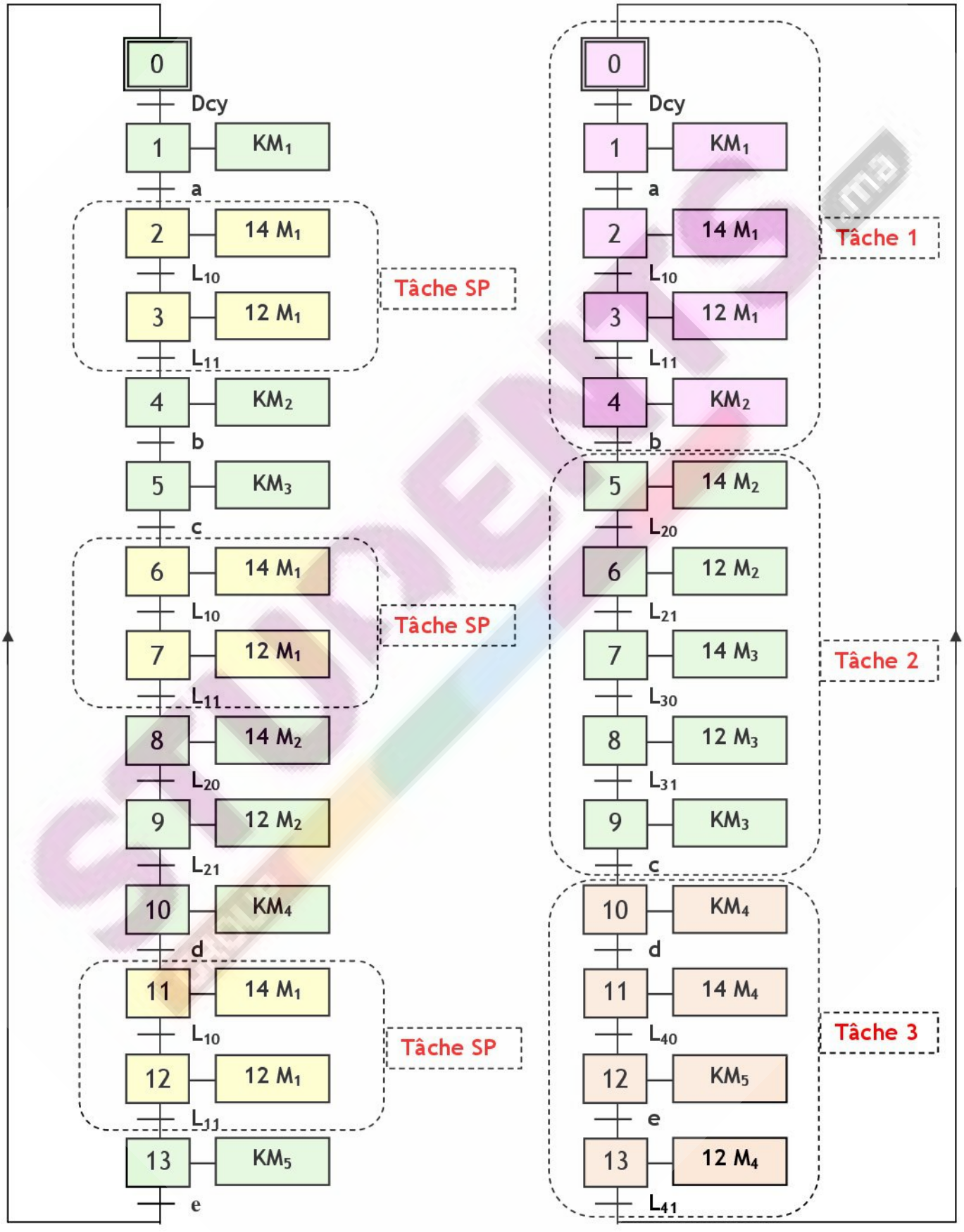
3.6- Constantes spéciales et extensions :

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

PARTITION D'UN GRAFCET

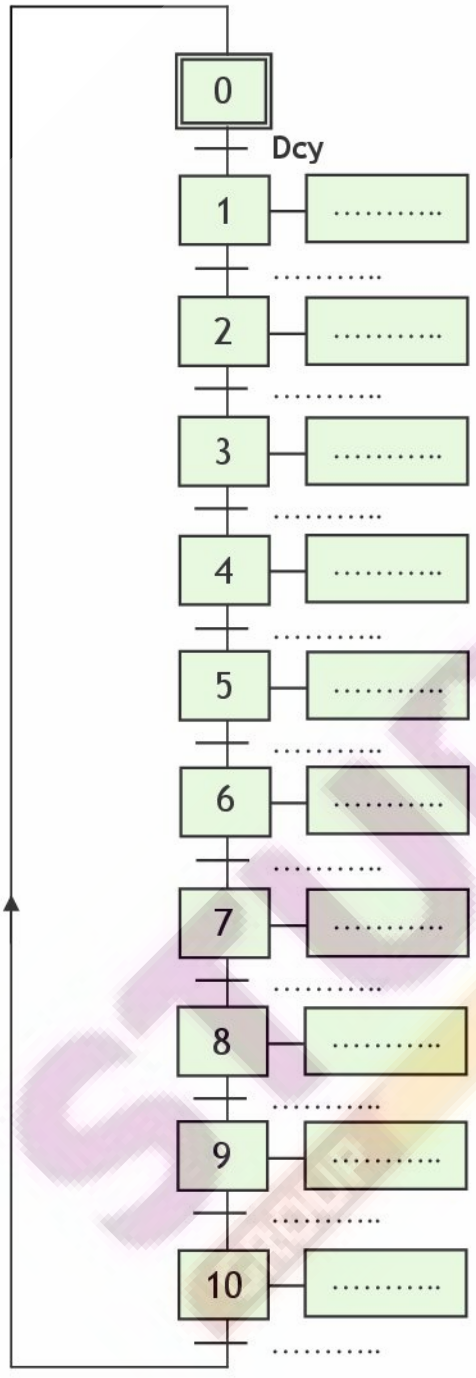
Soient les deux GRAFCETS ci-dessous :



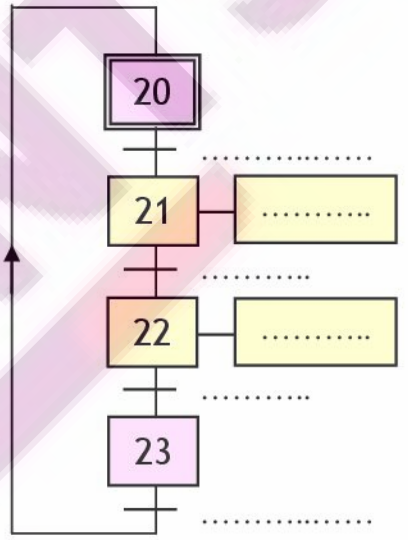
Coordination Verticale asynchrone :

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**.



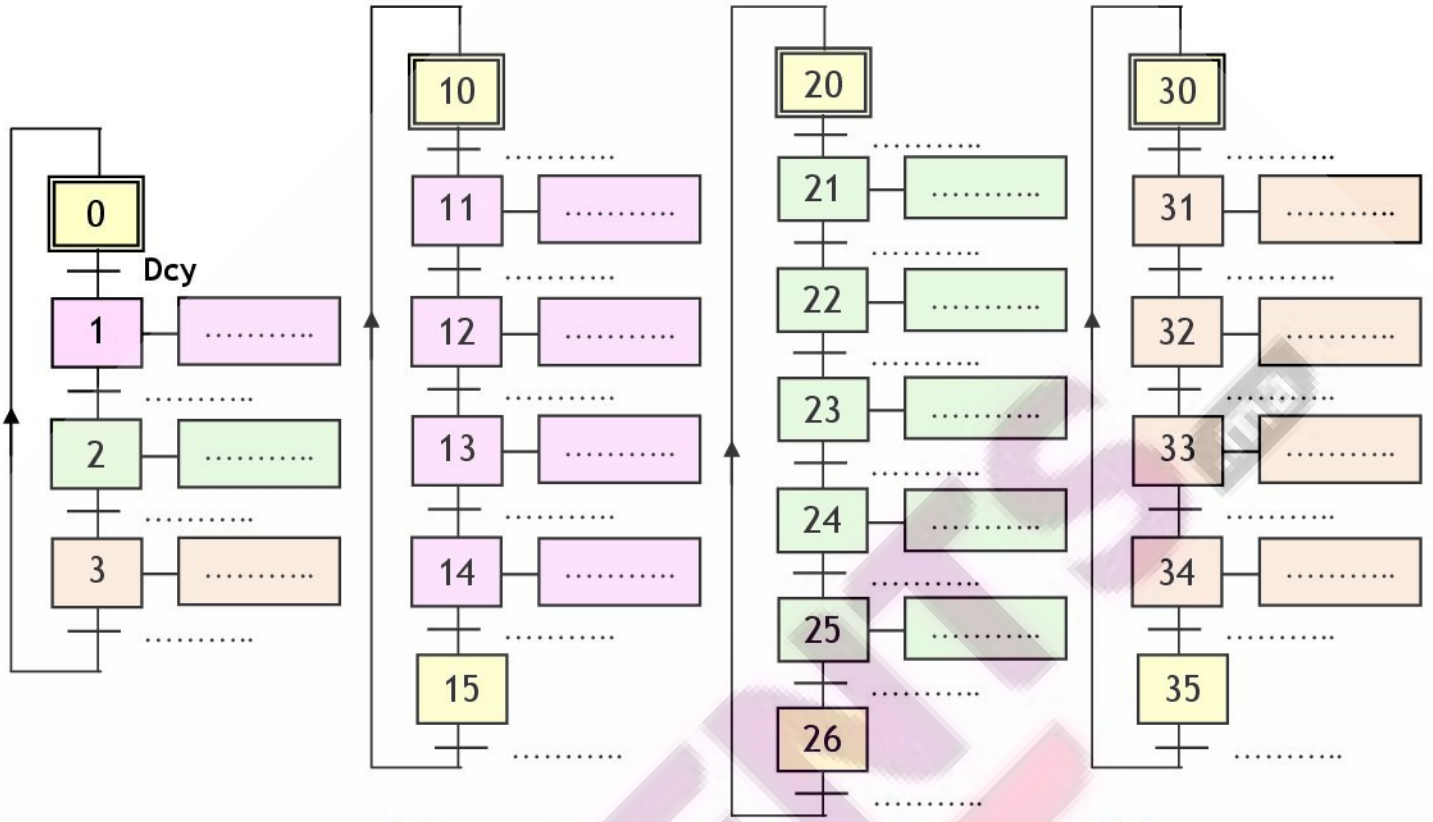
GRAFCET principal



GRAFCET de tâche SP

GRAFCET n°2 :

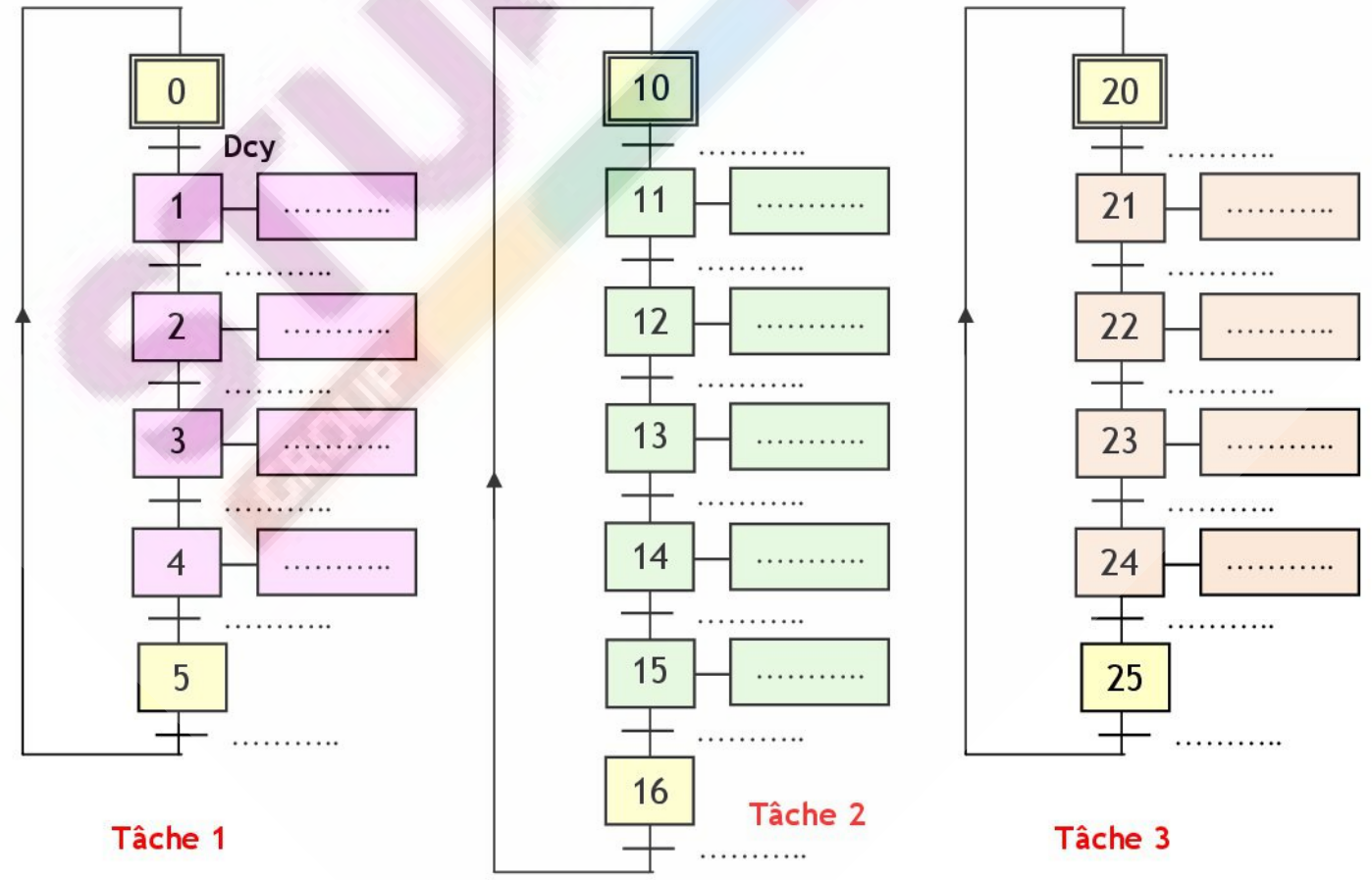
On veut réécrire le GRAFCET n°2 d'une manière simplifiée. Compléter le GRAFCET **principal** et les GRAFCETS de tâche **1, 2 et 3**.



GRAFCET principal Tâche 1 Tâche 2 Tâche 3

Coordination Horizontale :

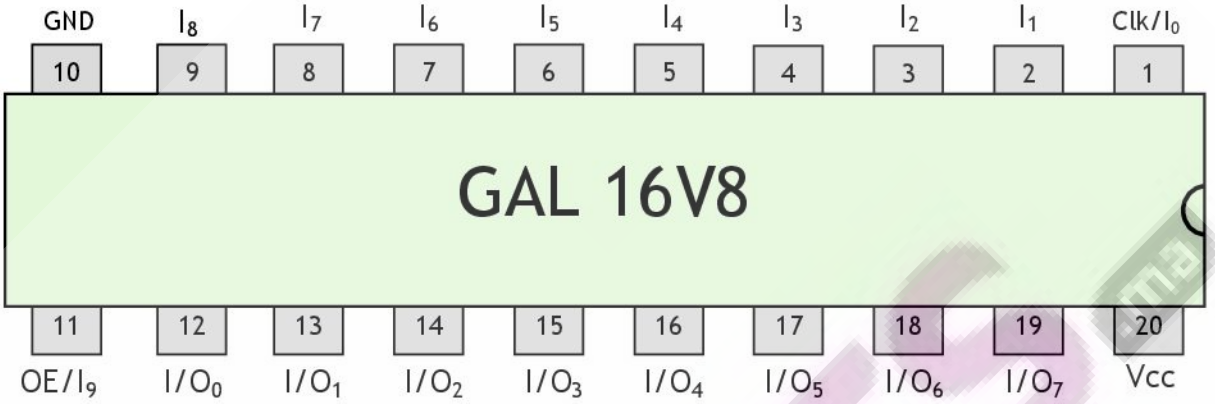
On veut réécrire ce GRAFCET n°2 d'une manière simplifiée. Compléter les GRAFCETS **Tâche 1, 2 et 3.**



Tâche 1 Tâche 2 Tâche 3

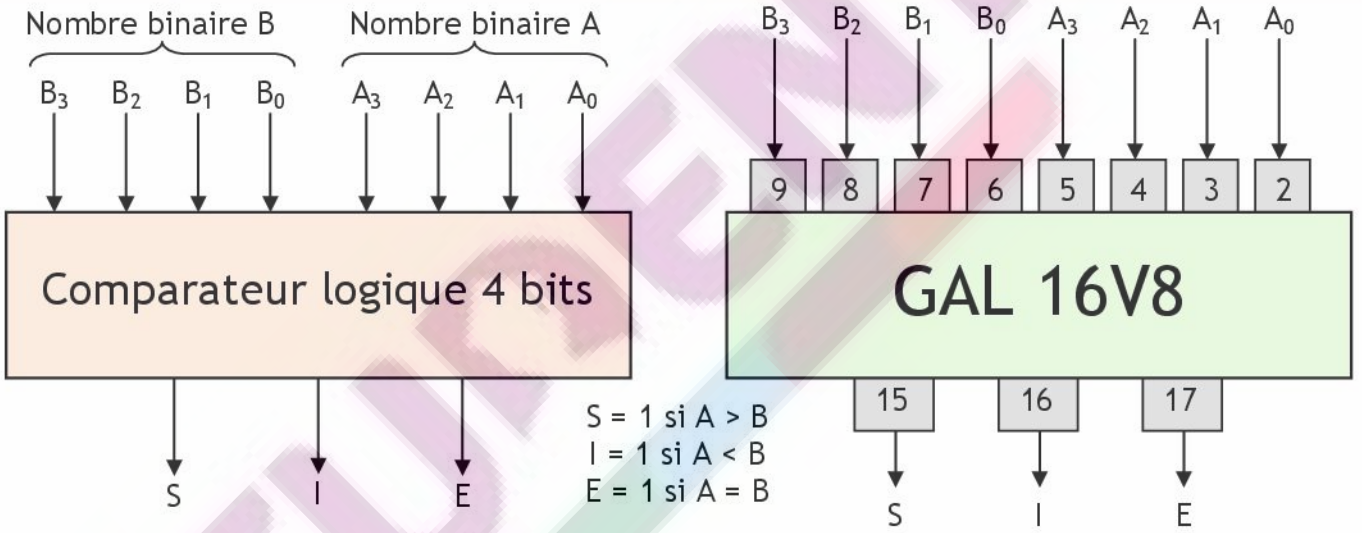
PROGRAMMATION DES PLD

GAL à programmer :



C'est circuit logique programmable et effaçable électriquement.
Le GAL 16V8 possède 16 Entrées et 8 sorties programmables.

1- Comparateur binaire 4 bits :



Description par équations :

MODULE comparateur
TITLE 'comparateur logique 4 bits'
DECLARATIONS
Comparateur **device** 'P16V8'; // 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 ....**

EQUATIONS
.....
.....
.....