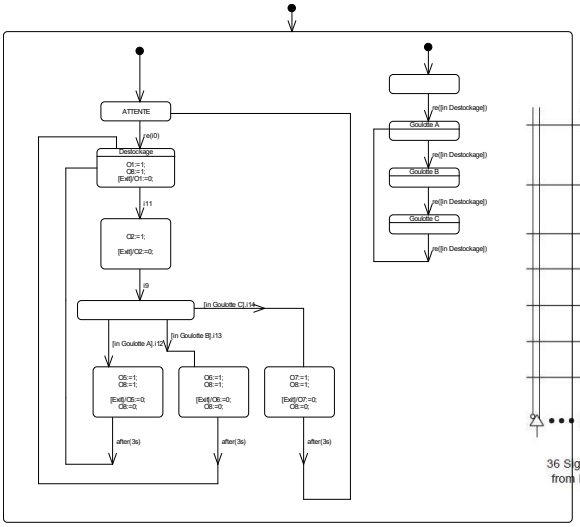
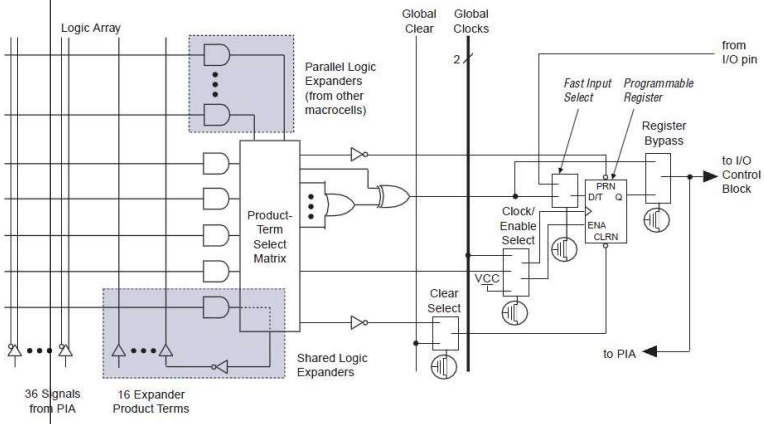
TSI – Lycée Chaptal St Brieuc

COURS – CI2

Système à Evènements Discrets (SED)

Partie 1 – Modéliser le comportement d'un SED à l'aide de diagrammes d'états





Je dois être capable de :

- ***Décrire le comportement d'une machine d'états à l'aide d'un diagramme d'états.***
- ***Connaître les différents types de codage des informations numériques***
- ***Savoir modéliser une condition de garde grâce aux équations logiques et table de vérité***

1. Comportements des systèmes à évènements discrets (SED)

On distingue trois types de systèmes contrôlés :

- **Systèmes asservis**, où la **grandeur de sortie** est : de **même nature** que la grandeur de consigne, **mesurée par un capteur, comparée** à la **consigne** puis **corrigée** si nécessaire ;

- **systèmes à événements discrets** (**SED**)**,** où le signal de sortie est élaboré à partir d'un signal d'entrée logique (ou d'une combinaison de signaux) et des résultats d'opérations logiques déjà réalisées ;

- **systèmes à logique combinatoire,** où le signal d'entrée logique (ou une combinaison des signaux d'entrée) conduit invariablement au même signal de sortie. Exemple : voyant lumineux panne d'un tableau de bord de voiture. Ces systèmes sont utiles pour définir des conditions de garde pour les SED.

Exemple de SED : télécommande de téléviseur

Un appui sur le bouton POWER entraine soit l'allumage ou soit l'extinction du téléviseur.

Ainsi, dans le cas d'un SED séquentiel :

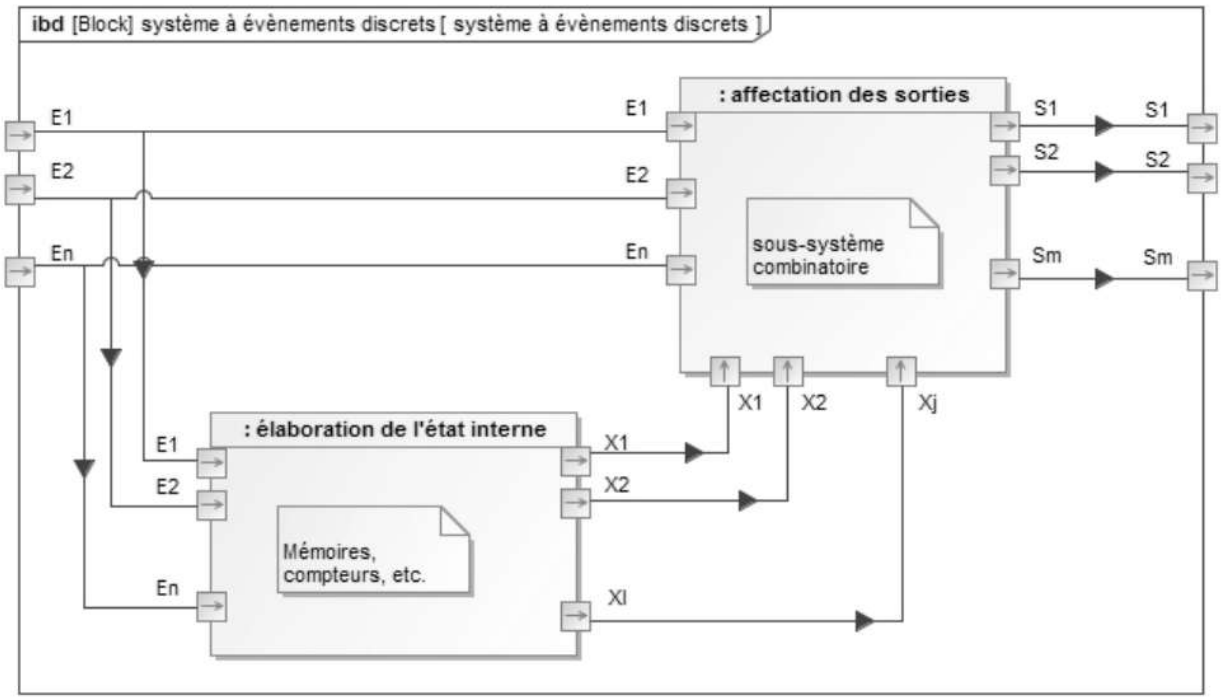
- une même cause (même combinaison des entrées) peut produire des effets différents (ex : bouton power qui allume ou éteint);

- le temps peut être une cause déclenchante (ex : arrêt du téléviseur après 1h d'inactivité);

- l'effet peut persister si la cause disparaît.

Pour décrire le comportement des SED, il est alors nécessaire de définir :

- Un vecteur d'état des variables de sortie $\mathbf{S = (S_1, S_2, \dots, S_m)}$,
- Un vecteur d'état des entrées $\mathbf{E = (E_1, E_2, \dots, E_n)}$,
- Vecteur d'état des variables internes $\mathbf{X = (X_1, X_2, \dots, X_i)}$.



2. Les machines d'états et graphes d'états.

La **Machine à nombre fini d'états** (FSM : Finite State Machine) représente le fonctionnement d'un système à évènements discrets. Elle comporte un nombre fini d'états et pourra se traduire par un programme dans un microprocesseur ou un automate, mais peut également rester au stade de concept (algorithme).

La machine d'états est constituée de différents éléments :

- un état initial,
- un ensemble d'évènements,
- un ensemble d'états,
- un ensemble d'activités,
- un état final.

Un **état** implique une activité ou une attente. Les états d'un système se succèdent en fonction d'évènements. Un **événement** peut conduire à une évolution du comportement du système. On l'appelle donc aussi déclencheur (**trigger**).

Le graphe d'état permet de décrire le comportement d'une machine d'état. Il est composé par :

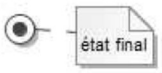
- Les nœuds, un état interne (action en cours),
- Les transitions, arc qui relie deux nœuds (conditions pour passer d'un nœud à un autre).

3. Eléments de lecture des diagrammes d'états.

Le **diagramme d'états** (state machine diagram ou stm) est un **diagramme normalisé SysML**. Il permet de décrire le comportement d'un système ou d'une de ses parties. Le diagramme d'états décrit la structure d'un bloc ou **instance** (objet du type du bloc).

3.1. Etats.

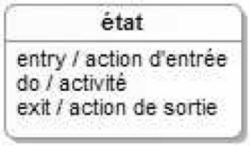
Etat initial : création de l'instance du bloc pour lequel le diagramme d'état est spécifié.



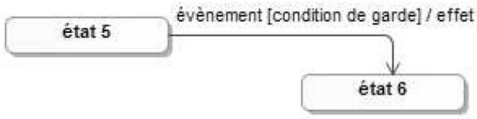
Etat final : destruction de l'instance de bloc (plusieurs fins possibles dans un même diagramme d'états).

Nœud de diagramme ou « état » : consiste à réaliser une activité, une action d'entrée et/ou une action de sortie.

- **Actions d'entrée ou de sortie** (mots clés « entry » ou « exit »). Elles provoquent un changement d'état, par exemple d'un préactionneur. Une action ne prend pas de temps et ne peut pas être interrompue.
- **Activités** (mot clé « do »). Elles permettent de spécifier le comportement du nœud. Une activité prend du temps et peut être interrompue.



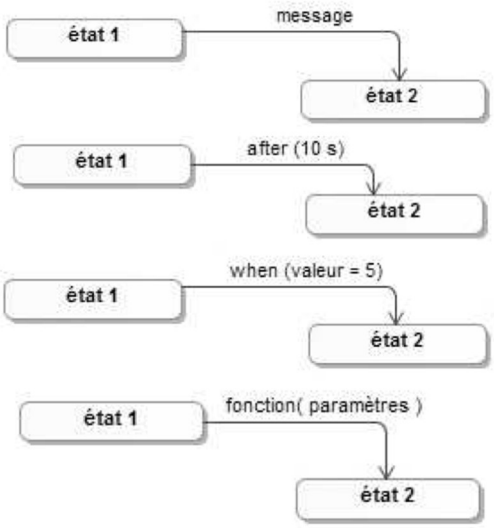
3.2. Transitions.

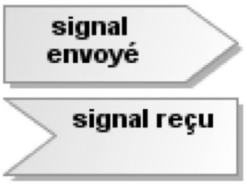


Transition : permet de passer d'un état à un autre. Une transition peut être associée à un **événement**, à une **condition de garde** et/ou à un **effet** (d'une action).

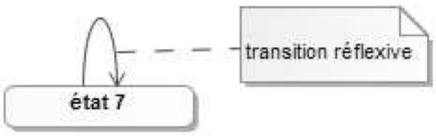
Il existe quatre types d'évènements associés à une transition :

- le **message** (signal event) : un message asynchrone est arrivé,
- l'**événement temporel** (time event) : un intervalle de temps s'est écoulé depuis l'entrée dans un état (mot clé « after ») ou un temps absolu a été atteint (mot clé « at »),
- l'**événement de changement** (change event) : une valeur a changé de telle sorte que la transition est franchie (mot clé « when »),
- l'**événement d'appel** (call event) : une requête de fonction (operation) du bloc a été effectuée. Un retour est attendu. Des arguments (paramètres) de fonction peuvent être nécessaires.





Les **événements** peuvent être utilisés pour décrire les interactions entre les différents blocs d'un système. En effet, un événement peut être émis par un autre bloc (**send signal action**) que celui pour lequel le diagramme d'états est spécifié. Il est alors par défaut destiné à tous les blocs du système (**diffusion « broadcast »)**. De même, un événement peut être reçu (**receive signal action**).

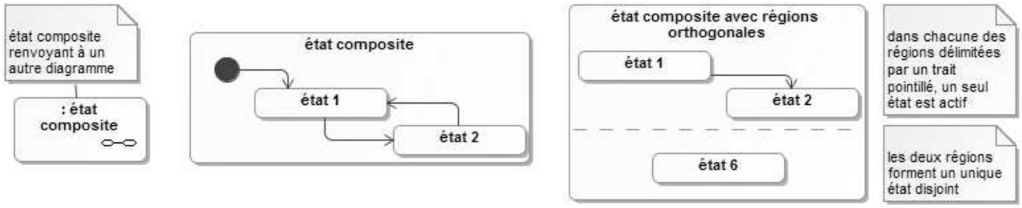


Une **transition réflexive** entraîne une sortie d'état puis un retour dans ce même état. Cela n'est donc pas sans conséquences selon les cas.

La **condition de garde** est une expression booléenne faisant intervenir des entrées et/ou des variables internes. Elle autorise le passage d'un état à un autre. Il est possible d'utiliser les notations non booléennes de front montant (🡅) et front descendant (🡇). La condition de garde est évaluée uniquement lorsque l'événement déclencheur se produit.

3.3. Notion d'état composite.

Un état composite est constitué de sous-états liés par des transitions. Cela permet d'introduire la notion d'état de niveau hiérarchique inférieur et supérieur.



4. Notion de pseudo-état.

Un **pseudo-état** est un élément de commande qui influence le comportement d'une machine d'état. Ils peuvent être utilisés dans un diagramme d'états ou dans un diagramme d'activité.

Le formalisme SysML admet **neuf pseudo-états** :

«shallow history» : permet à un état de niveau hiérarchique supérieur (état composite) de se souvenir du dernier sous-état, avant qu'il n'évolue vers un autre état. Lors de la réactivation de l'état composite, la machine d'état se réactive au niveau de l'état mémorisé.

«deep history» : idem que précédemment mais avec la propagation de « l'histoire » à tous les sous-états composites de niveaux hiérarchiques inférieurs.

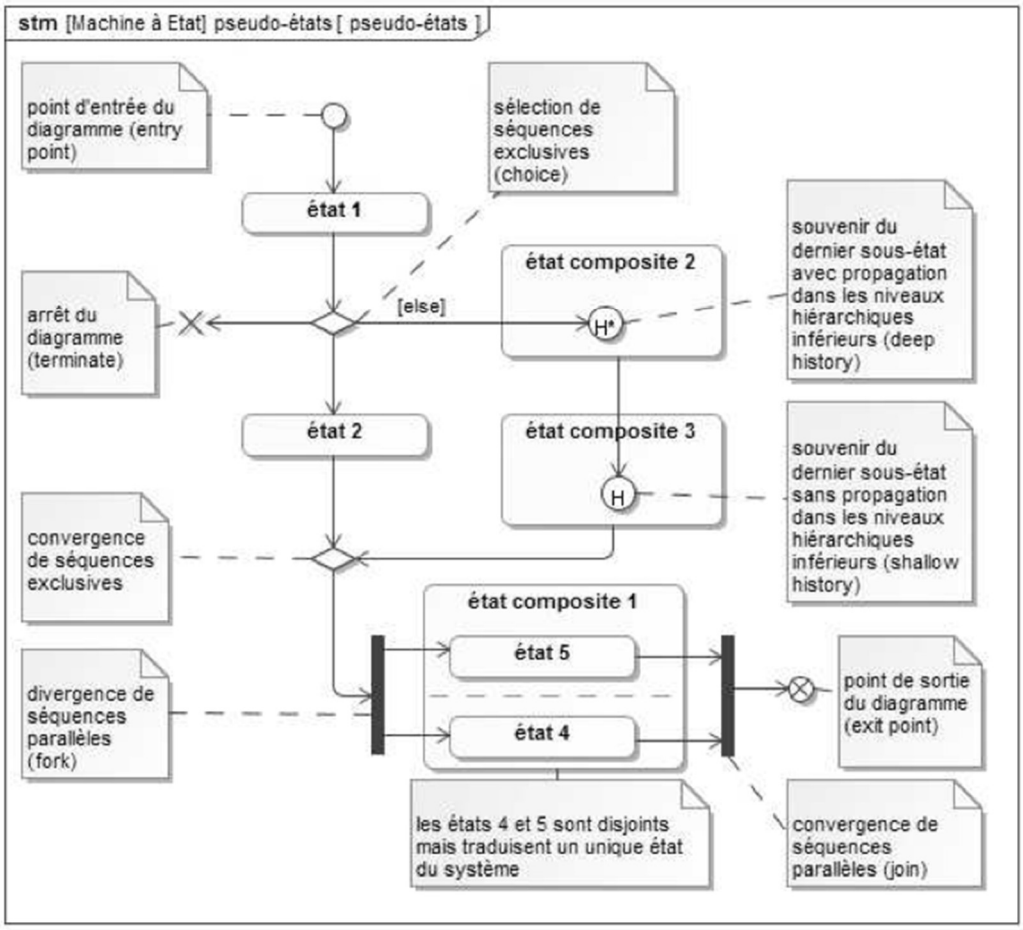
«fork» et**«join»** : divergence et convergence de séquences parallèles.

« choice » : sélection et convergence de séquences exclusives. Il est nécessaire qu'une condition située en aval soit vraie pour que l'évolution du système se poursuive. Les conditions de gardes doivent être exclusives. Le mot clé « else » peut-être utilisé pour englober tout ce qui n'est pas décrit dans les autres expressions booléennes. Les conditions de garde situées en aval sont toutes évaluées une fois le pseudo-état atteint.

« junction » : idem au pseudo-état **« choice »**, à la différence que pour qu'un chemin soit emprunté, toutes les conditions de garde situées en aval et en amont, doivent être vraies. L'évaluation des conditions avales est réalisée avant que le pseudo-état soit atteint.

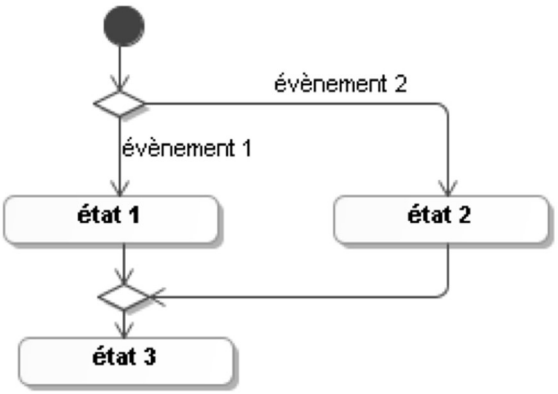
« entry point » et **« exit point »** : permet de créer un point d'entrée du diagramme et un point de sortie vers un autre diagramme.

«terminate» : permet de terminer une séquence sans destruction de l'instance de bloc.

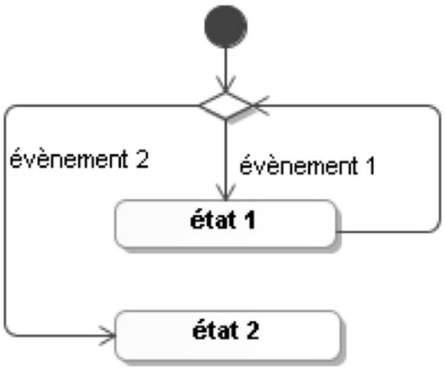


5. Exemples de structures conditionnelles avec le diagramme d'état.

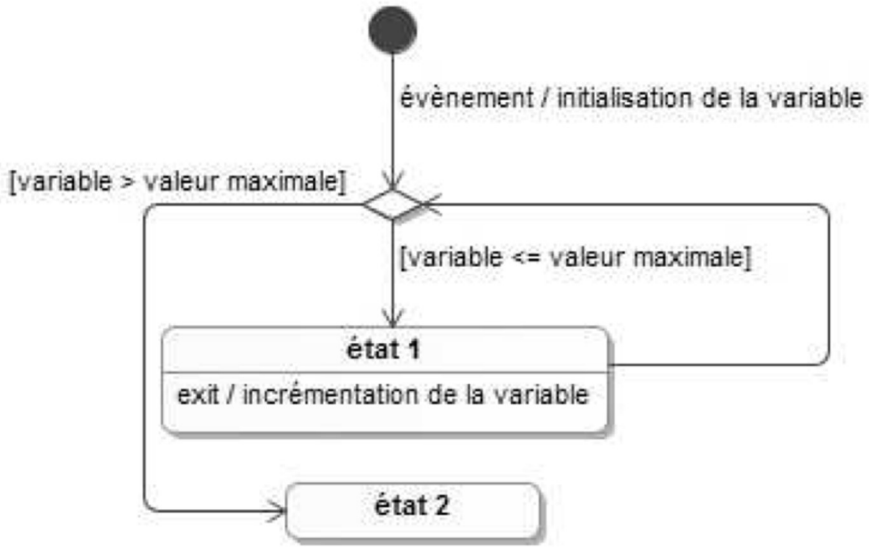
Si… alors… sinon…



Tant que (condition vraie) faire…



Pour (variable de 0 à Vmax) faire…

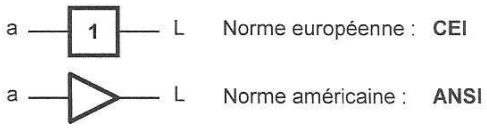


6. Définition des conditions de garde par des opérateurs logiques

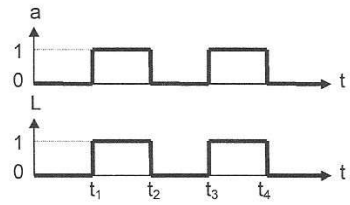
Les conditions de garde correspondent souvent à des états logiques « 0 » ou « 1 ». Afin de créer ces états logiques, la logique combinatoire est très utile.

a. Les différentes représentations

Portes logiques et fonction logiques : les symboles utilisés respectent la norme CEI (ou européenne), ou la norme ANSI (américaine). Un ensemble de symboles logiques associés est un **logigramme**.



Les **chronogrammes** permettent de représenter l'évolution des entrées et des sorties en fonction du temps : état **"0",** état **"1",** état indéterminé Ø **ou X**, état **Hiz** ou "haute impédance".



Pour générer une condition de garde (ou un état logique), il est nécessaire d'avoir un signal logique associé. En **BASE 2**, on utilise classiquement 5V pour **"1"** logique, et 0V pour **"0"** logique.

b. Les opérations logiques

Les **tables de vérité** permettent de représenter les différentes combinaisons possibles relatives aux systèmes logiques.

a	L
0	0
1	1

Les différentes **fonctions logiques de base** :

- **OUI**,
  L est à "1" si a est à "1"
- **NON**,
  L est à "1" si a est à "0"
- **ET** (AND),
  L est à "1" si a ET b sont à "1"
- **OU** (OR),
  L est à "1" si a OU b est à "1"

Table de vérité	Equation logique	Symbole logique
Opérateur identité : **OUI**		
a | L 0 | 0 1 | 1	L = a	CEI / ANSI
Opérateur complément : **NON**		
a | L 0 | 1 1 | 0	$L = \overline{a}$	CEI / ANSI
Opérateur **ET** (AND)		
a | b | L 0 | 0 | 0 0 | 1 | 0 1 | 0 | 0 1 | 1 | 1	L = a · b	CEI / ANSI
Opérateur **OU** inclusif (OR)		
a | b | L 0 | 0 | 0 0 | 1 | 1 1 | 0 | 1 1 | 1 | 1	L = a + b	CEI / ANSI

c. Table de vérité et équation logique

Le fonctionnement de la structure logique peut aussi être représenté par un ensemble de tables de vérité (une table par sortie de la structure). La taille des tables de vérité est définie par le nombre d'entrées qui influencent chaque sortie :

- 2 entrées = 4 combinaisons possibles,
- 3 entrées = 8 combinaisons possibles,
- 4 entrées = 16 combinaisons possibles,
- *on généralise* avec **n entrées = 2^n combinaisons possibles.**

On peut ainsi donner les tables de vérité de la structure étudiée et ensuite en déduire les **équations logiques** de la logique combinatoire :

S1	WDEWIE	EWI
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

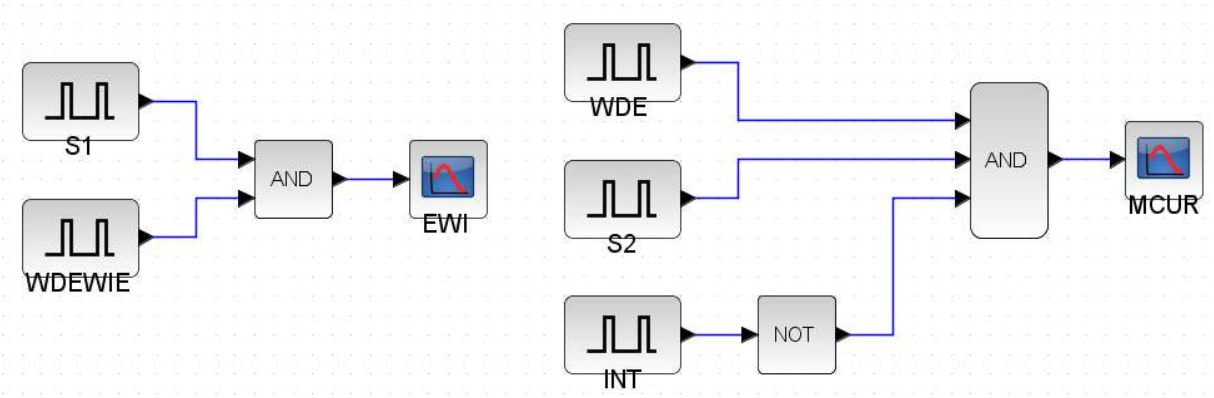
$$EWI = S1.WDEWIE$$

WDE	S2	INT	MCUR
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

$$MCUR = WDE.S2.\overline{INT}$$

d. Logigramme

A partir des **équations logiques**, on peut réaliser des **logigrammes.** Ces logigrammes qui pourront être **directement implémentés sur un circuit logique** en utilisant un logiciel dédié à la réalisation de circuits électriques comme Proteus. On donne ci-dessous le logigramme des équations logiques déterminées ci-dessus :



e. Les principaux codages

On utilise le **binaire naturel** pour présenter les différentes combinaisons d'entrées des tables de vérité. Le **code GRAY** (ou **binaire réfléchi**) s'utilise dans quelques cas particuliers : particularité, seul un état change d'une ligne à une autre.

Code décimal	Code binaire naturel				Code héxa	Code Gray			
0	**0**	**0**	**0**	**0**	0	0	**0**	**0**	**0**
1	**0**	**0**	**0**	**1**	1	0	**0**	**0**	**1**
2	**0**	**0**	**1**	0	2	0	**0**	**1**	1
3	**0**	**0**	**1**	1	3	0	**0**	**1**	0
4	**0**	**1**	0	0	4	0	**1**	1	**0**
5	**0**	**1**	0	1	5	0	**1**	1	**1**
6	**0**	**1**	1	0	6	0	**1**	0	1
7	**0**	**1**	1	1	7	0	**1**	0	0
8	**1**	0	0	0	8	1	1	**0**	**0**
9	**1**	0	0	1	9	1	1	**0**	**1**
10	**1**	0	1	0	A	1	1	**1**	1
11	**1**	0	1	1	B	1	1	**1**	0
12	**1**	1	0	0	C	1	0	1	**0**
13	**1**	1	0	1	D	1	0	1	**1**
14	**1**	1	1	0	E	1	0	0	1
15	**1**	1	1	1	F	1	0	0	0