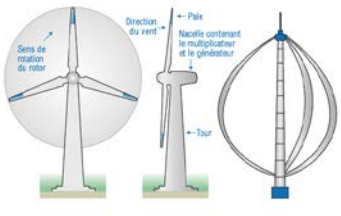


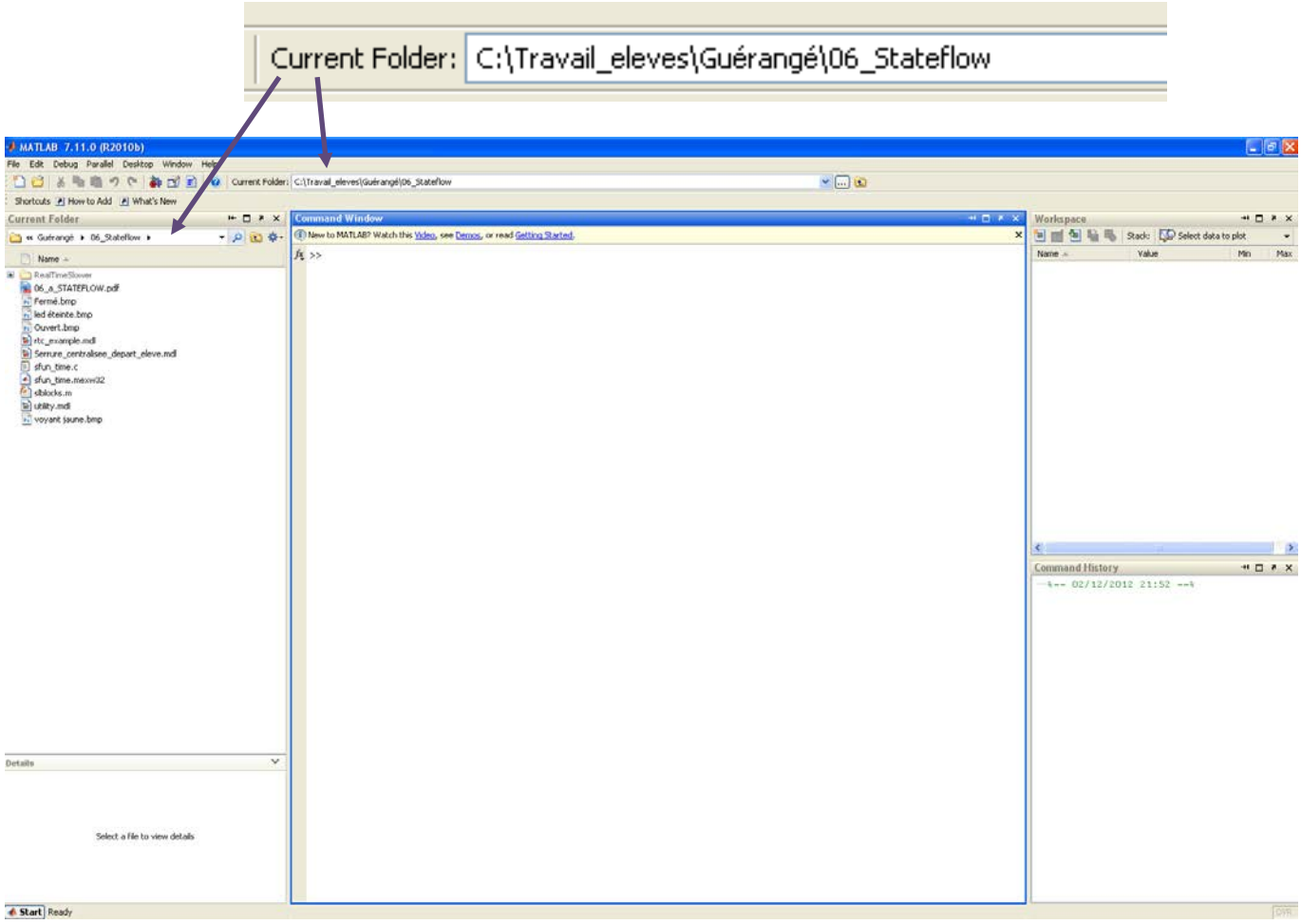
Automate séquentiel : Stateflow

Ouverture du logiciel

Avant tout travail il faut recopier les projets élèves depuis l'endroit signalé par votre professeur et les mettre dans le dossier c:\travail_élèves\ dans un répertoire à votre nom.

Il faut ensuite lancer la version 32 bits de matlab depuis l'icône sur le bureau. Et enfin positionner le dossier de travail dans matlab.

Une fois ces étapes réalisées vous pouvez réaliser la suite du TP Stateflow.



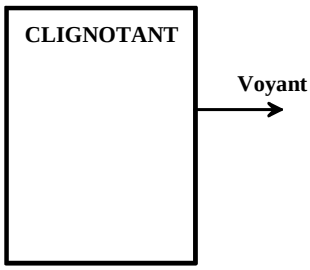
1 Premier exemple un clignotant.

Nous allons réaliser la simulation d'un clignotant avec stateflow. C'est un projet simple qui ne demande pas beaucoup de codage. Pour l'étude nous allons effectuer les étapes précisées ci-dessous :

1) Faire le bilan des entrées sorties de l'automate.
2) Ouvrir matlab puis simulink créer le modèle avec un graphe stateflow : **clignotant.mdl**
3) Ajouter la sortie pour l'automate : **Voyant**.
4) Ouvrir le diagramme stateflow pour en faire la saisie.
5) Compléter le modèle simulink avec la sortie voyant utilisation de **Gauge Blockset**.
6) Ajout de la fonction **Soft Real Time** pour simuler en temps réel.
7) Régler la durée de la simulation.
8) Simuler le clignotant.

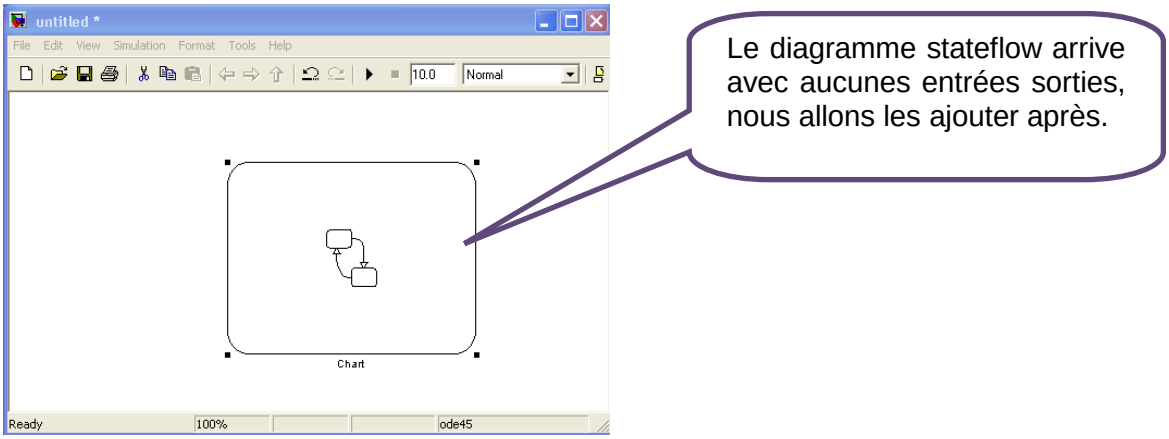
1.1 Bilan des entrées sorties de l'automate

Ici c'est très simple il n'y a qu'une seule sortie :



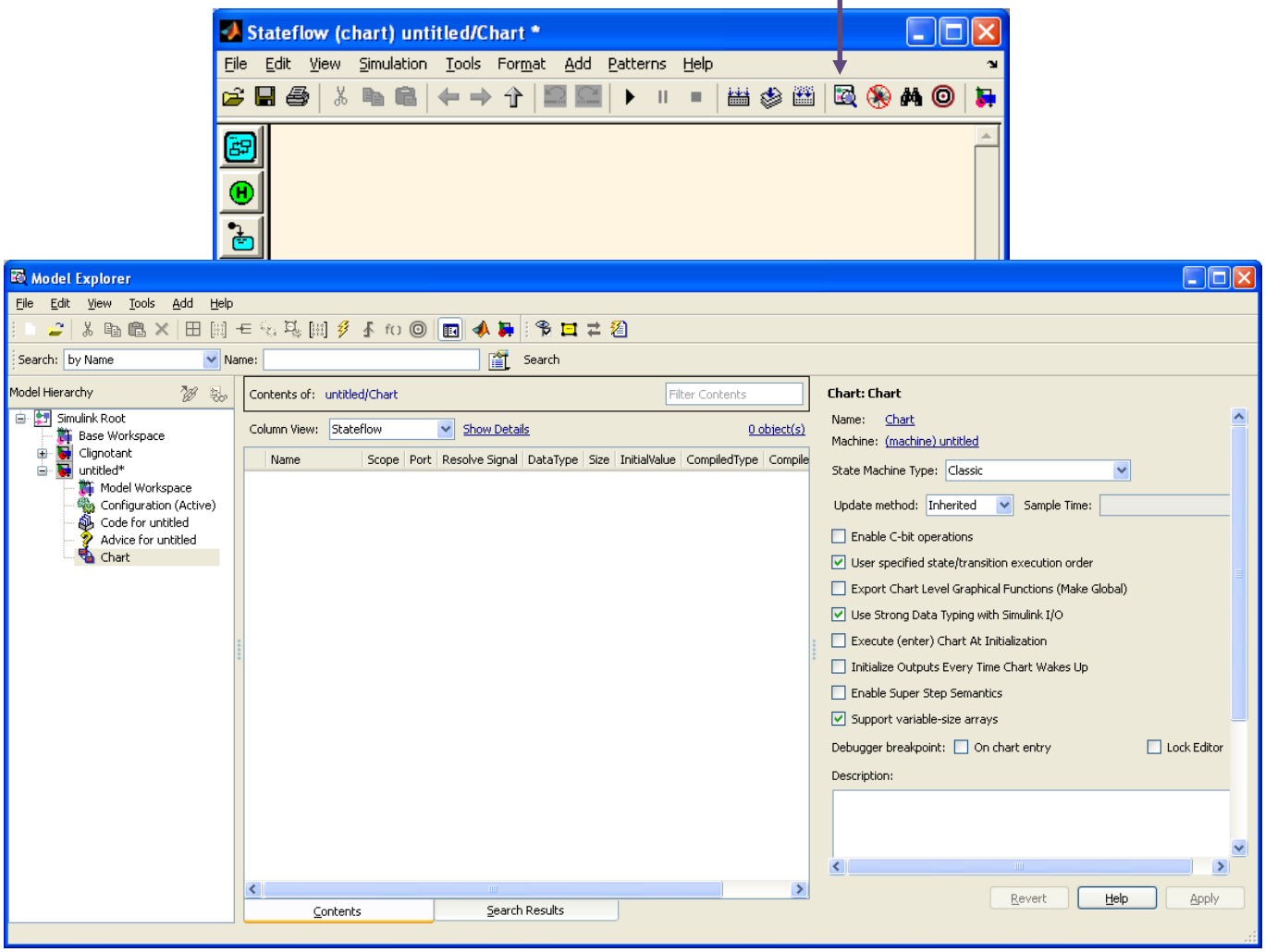
1.2 Création du modèle

Ouvrir matlab, version 32 bits, puis se positionner dans l'espace de travail recommander par votre enseignant. Ouvrir ensuite simulink et y insérer un graph stateflow :



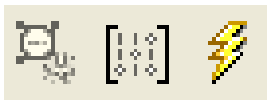
1.3 Ajout de la sortie

Ouvrir l'éditeur stateflow puis l'outil Explore



L'explorateur de modèle est ouvert, nous pouvons ajouter notre sortie :

1) Utiliser l'outil AddData du modèle explorer :

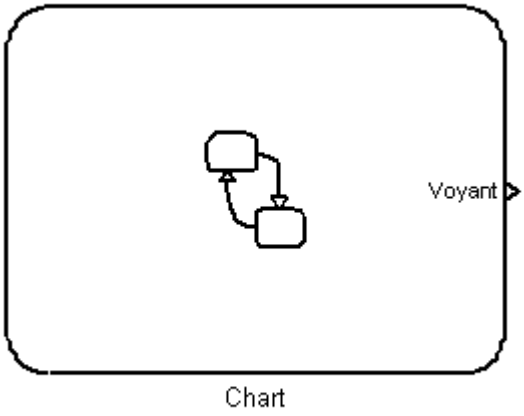


2) Modifier le nom et le scope de la data

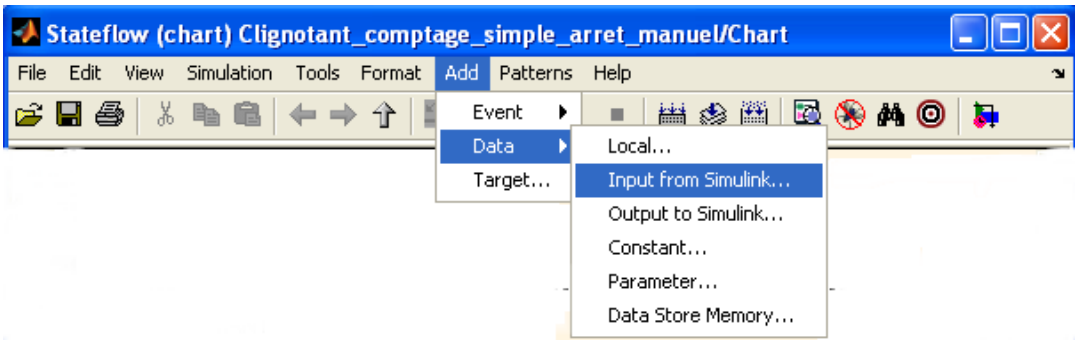
   nom : Voyant
   scope : output
   valider les modifications par apply

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType	Compile
Voyant	Out...	1	☐	double			unknown	

La sortie a bien été ajoutée à notre state diagram :

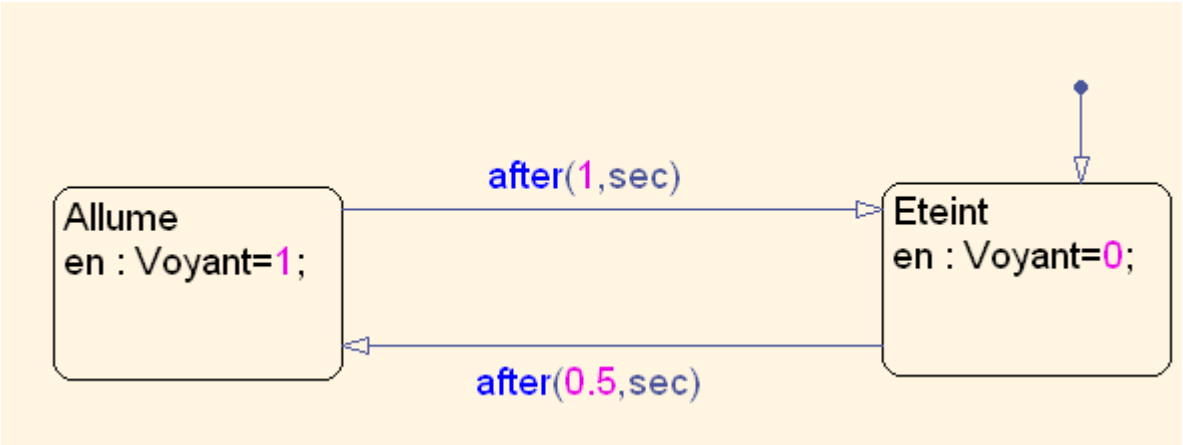


Une autre possibilité pour ajouter une variable, input output ou autre à notre description à partir de stateflow :



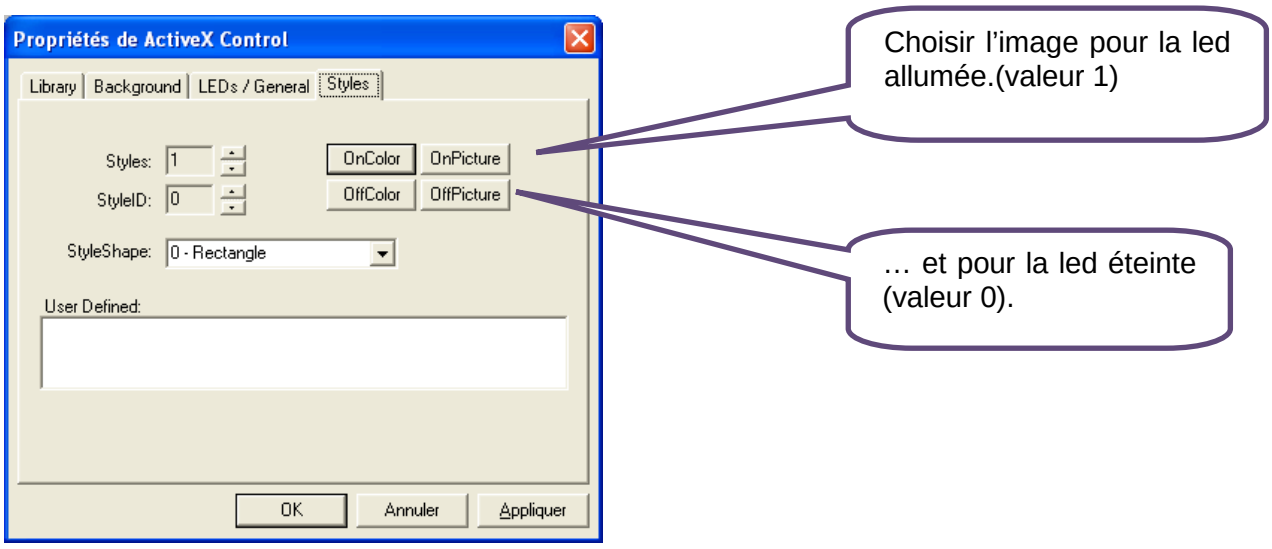
1.4 Ouvrir le diagramme stateflow pour en faire la saisie

Utiliser stateflow pour saisir la machine d'état décrite ci-dessous :



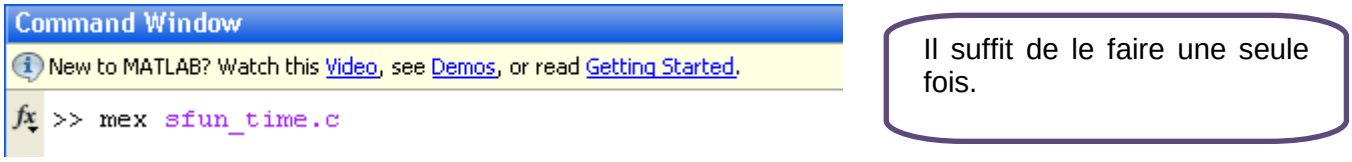
1.5 Compléter le modèle simulink avec la sortie voyant.

Nous allons utiliser la boite à outils Gauge Blockset. Il faut tout d'abord charger le composant Led. Nous allons ensuite le customiser en lui faisant afficher deux images personnalisées quand le voyant est allumé ou éteint.

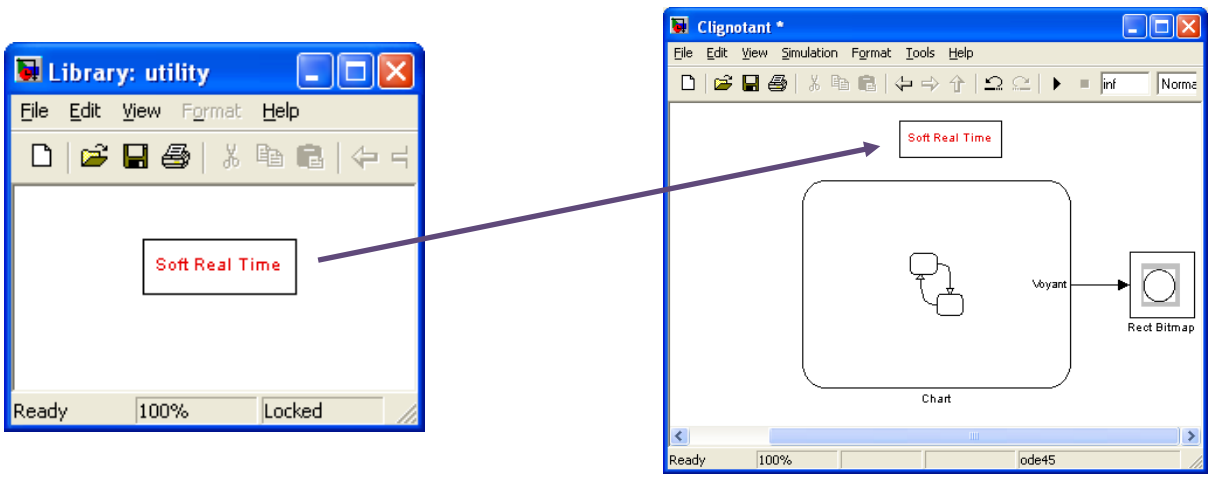


1.6 Simuler en temps réel

Ajout de la fonction Soft Real Time pour simuler en temps réel, cela règle simulink pour que le temps simulink soit identique au temps réel. Il faut pour cela avoir compilé sfun_time.c avec la commande

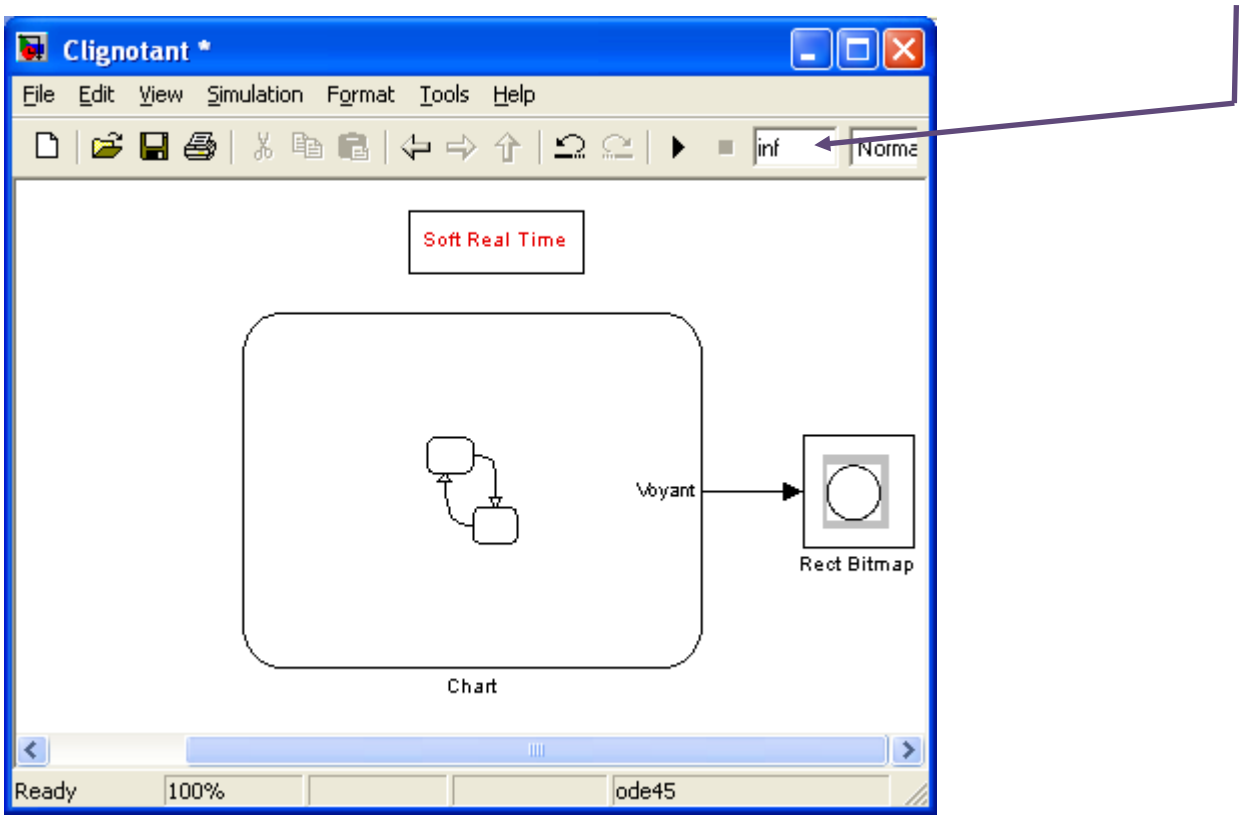


Les fichiers étant positionnés dans le répertoire de travail. La fonction soft real time peut ensuite être recopiée à partir du modèle utility.mdl le module est inséré par glisser déposé dans votre modèle simulink.



1.7 Régler la durée de la simulation

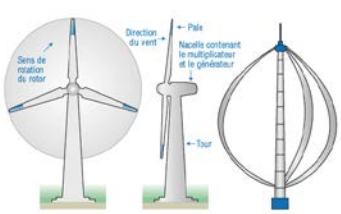
Nous souhaitons une simulation permanente, il faut régler la durée de simulation sur la valeur inf :



1.8 Simulation du clignotant

Vous pouvez maintenant lancer le simulateur, faites constater le fonctionnement par votre enseignant.

Bravo vous êtes prêt pour la deuxième simulation.



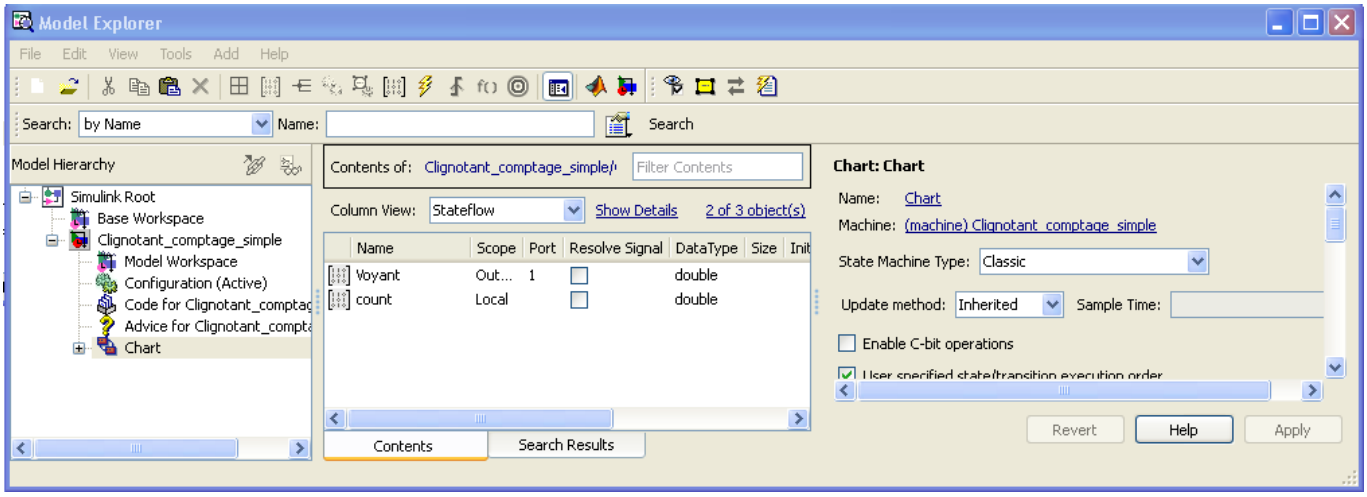
2 Deuxième exemple clignoter x fois

Nous cherchons maintenant à compter nos clignotements et à limiter le nombre de cycle d'allumage et d'extinction.

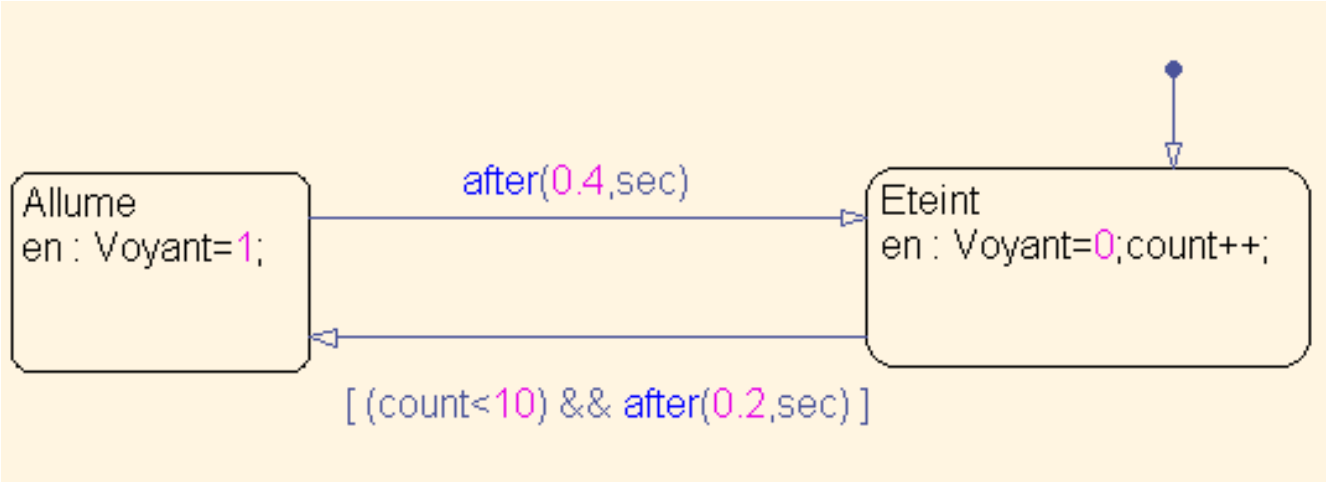
Pour créer cet automate suivre les étapes ci-après :

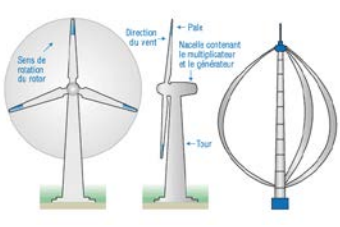
1) Faire le bilan des entrées sorties de l'automate. Y a-t-il une modification de l'automate précédent ?
2) Ajouter une variable locale appelée count qui va nous permettre de comptabiliser les cycles.
3) Modifier le diagramme stateflow précédent en prenant soin de le sauvegarder avec un nouveau nom.
4) Simuler le fonctionnement clignotant.

2.2 Ajout de la variable locale :



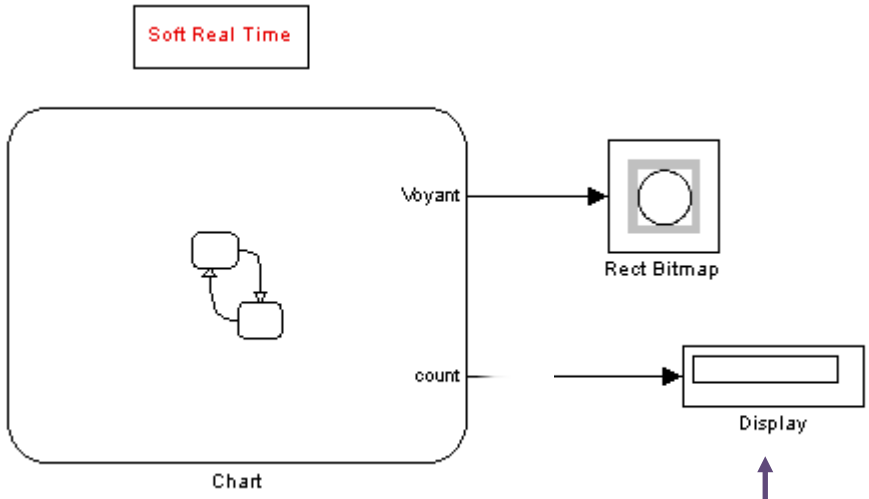
2.3 Saisie du nouveau diagramme stateflow





3 Modification de l'automate pour afficher le nombre de cycle comptabilisé

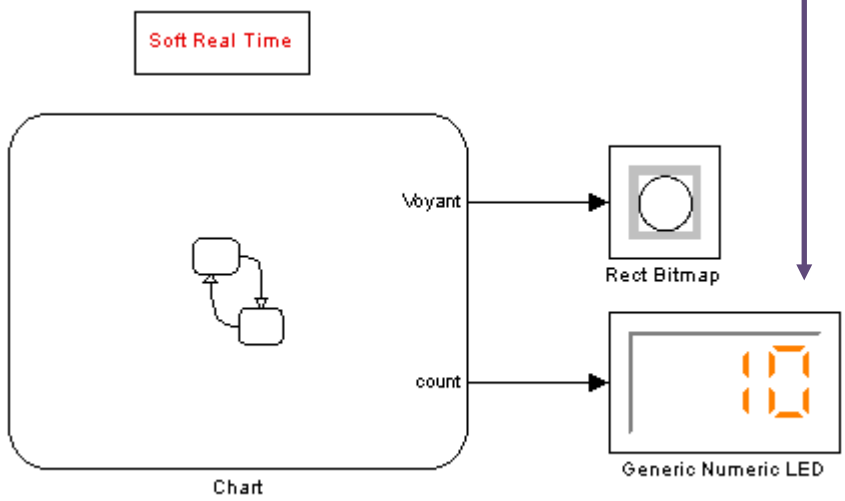
Modifier le scope de la variable count pour la passer de la valeur local à la valeur sortie. Une nouvelle sortie apparaît sur l'automate :



Pour afficher la valeur mettre un display

Simuler le fonctionnement.

Le display est assez impersonnel nous allons le remplacer par un outil de Gauge Blockset à savoir un Generic Numeric LED, une fois connecté à la place du display il faut le configurer pour afficher trois digits comme sur le schéma ci-dessous :



Tester votre simulation et faites constater le bon fonctionnement à votre professeur.

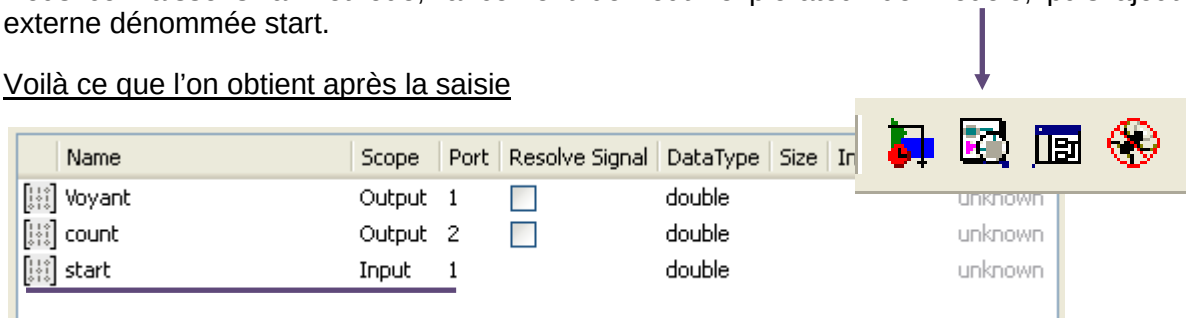
4 Ajout d'entrées externes

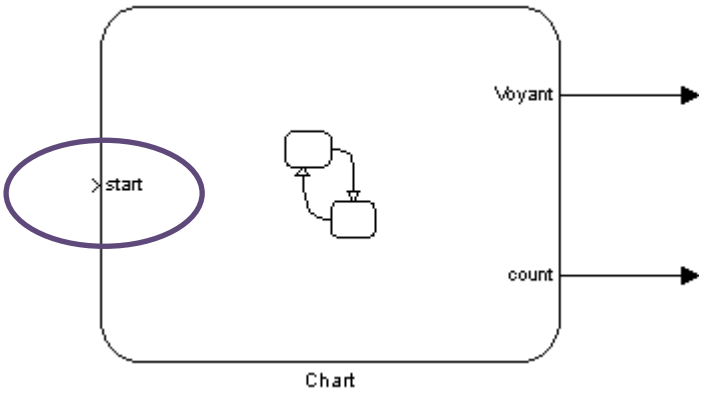
Nous voulons maintenant déclencher à nouveau le clignotement. Pour cela il nous faut disposer d'une entrée externe à notre state diagram. Puis modifier la machine d'états en conséquence.

4.1 Ajout d'une entrée

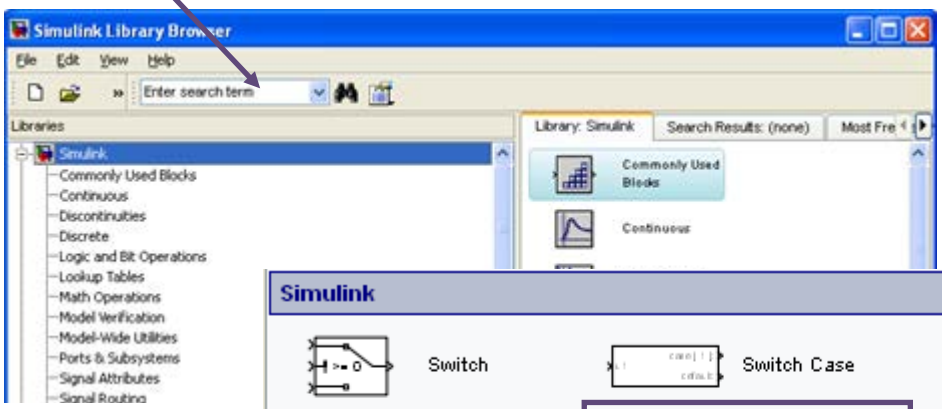
Nous connaissons la méthode, lancement de l'outil explorateur de modèle, puis ajout d'une entrée externe dénommée start.

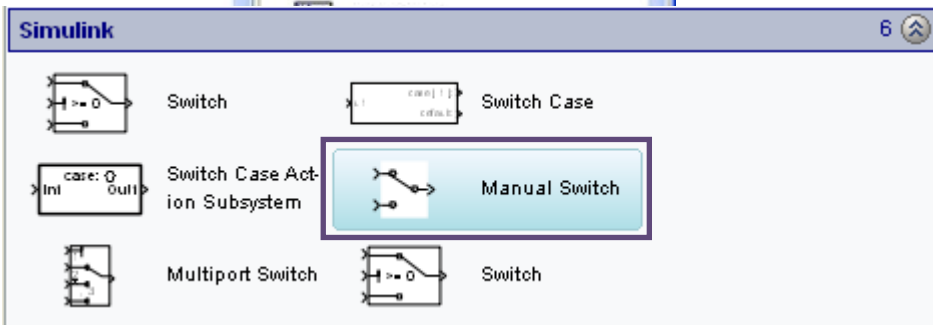
Voilà ce que l'on obtient après la saisie





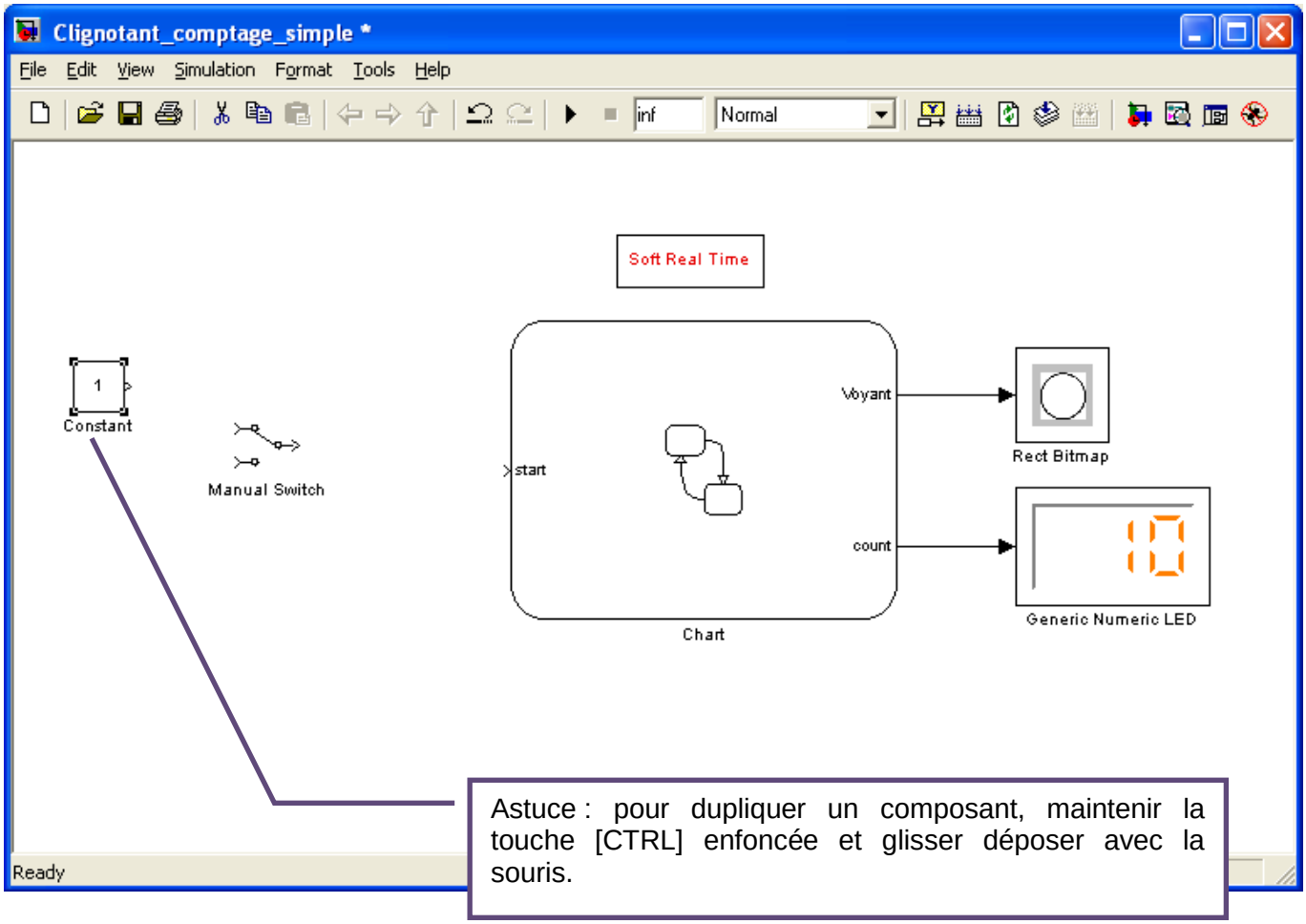
Ensuite nous ajoutons le commutateur manuel pour fixer les valeurs des entrées manual switch et constante. Si nous ne connaissons pas où trouver ces composants nous utilisons la recherche du Simulink Library Browser :



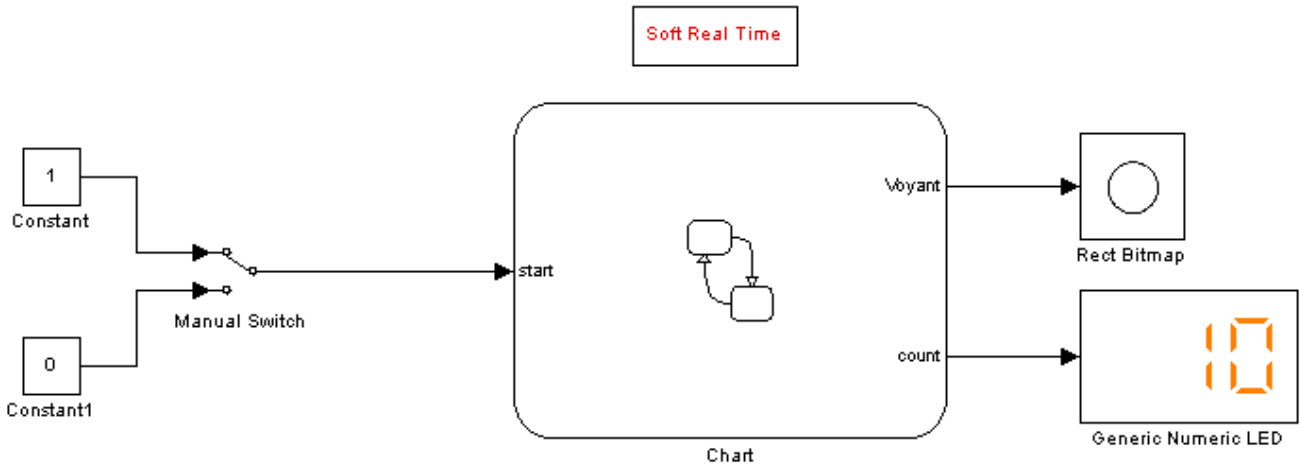


Nous trouvons alors l'emplacement du composant. Ici dans la toolBox Simulink.

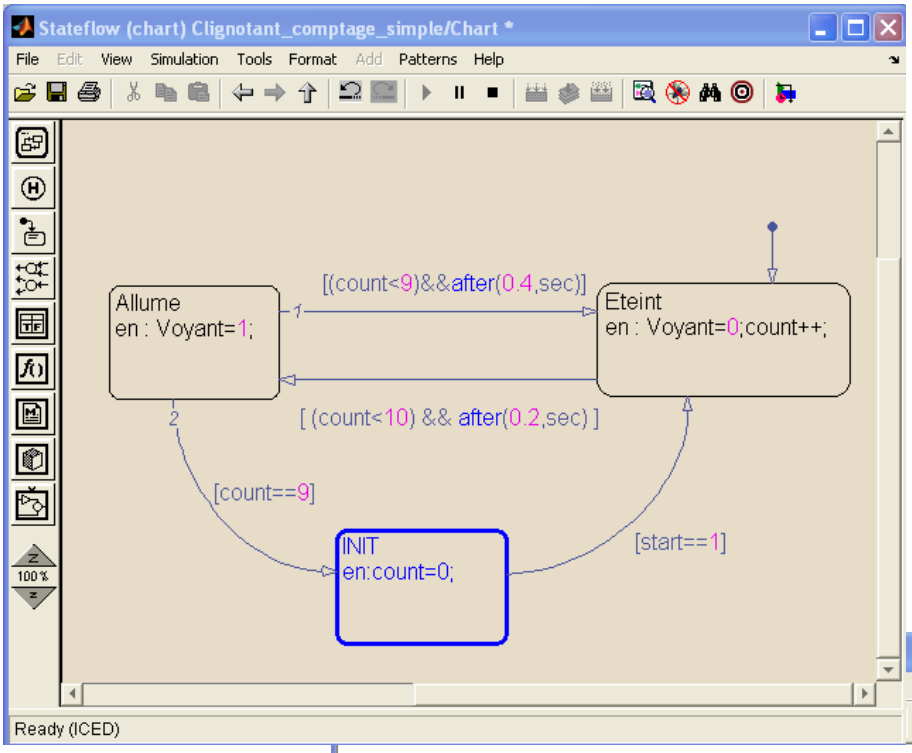
Procéder de même pour localiser le composant constant.

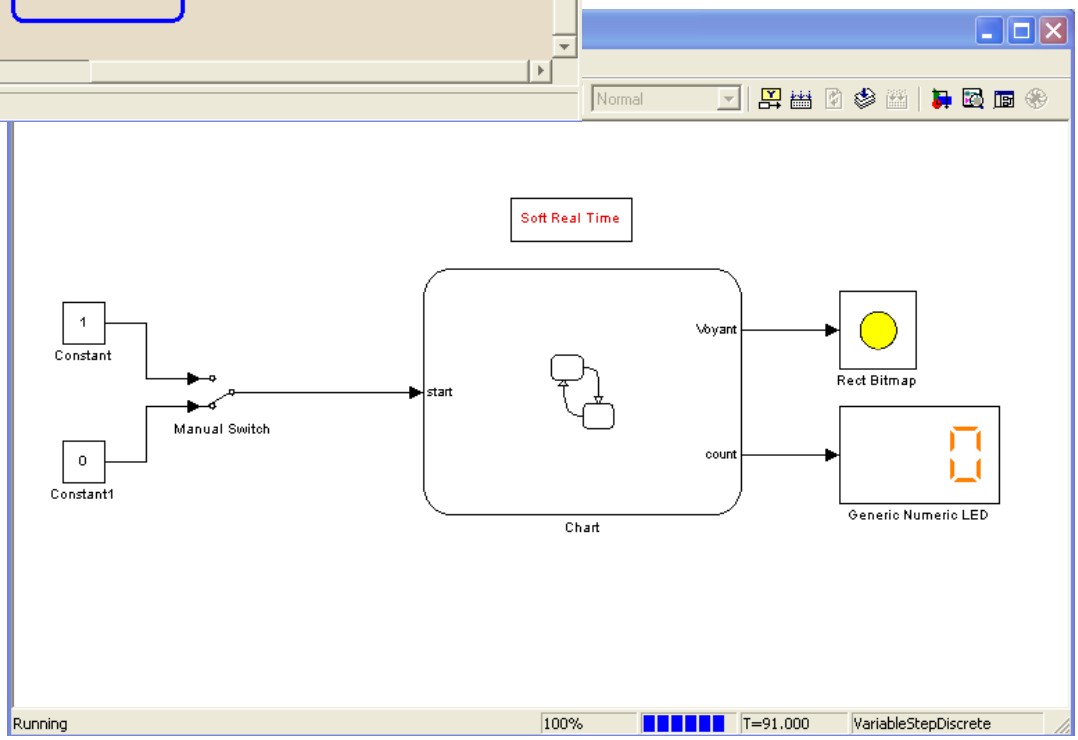


Voilà le modèle terminé :



Modifions l'automate pour prendre en compte la réinitialisation du clignotement. Voilà une solution :

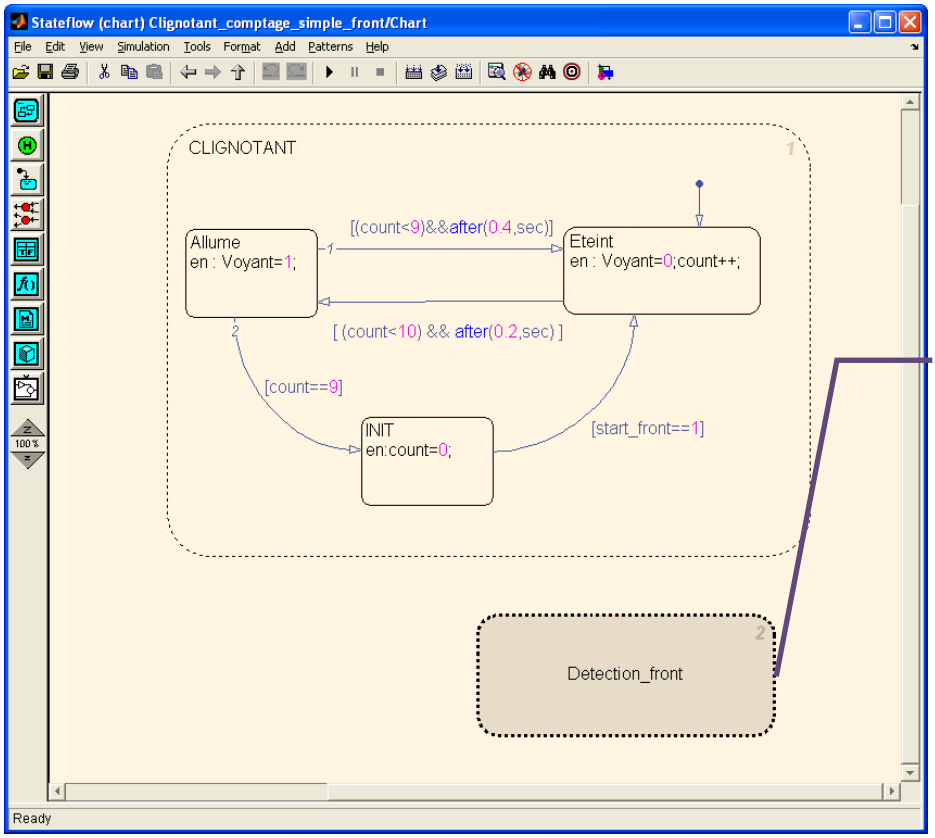




5 Prise en compte des fronts

Parfois, et même souvent, il faut détecter le changement d'une variable et non pas son niveau. Cette détection des fronts nécessite une mémorisation de la valeur n-1 pour la comparer à la valeur n. Si les deux valeurs sont différentes alors un front est détecté.

La détection des fronts peut se réaliser de plusieurs manières. Ici dans notre exemple nous allons utiliser une fonction script placée dans un état Détection_front un peu particulier puisqu'il s'agit d'un subchart.



Un subchart est un super état qui contient une description stateflow complète.

Utiliser les subchart permet de simplifier la représentation des automates.

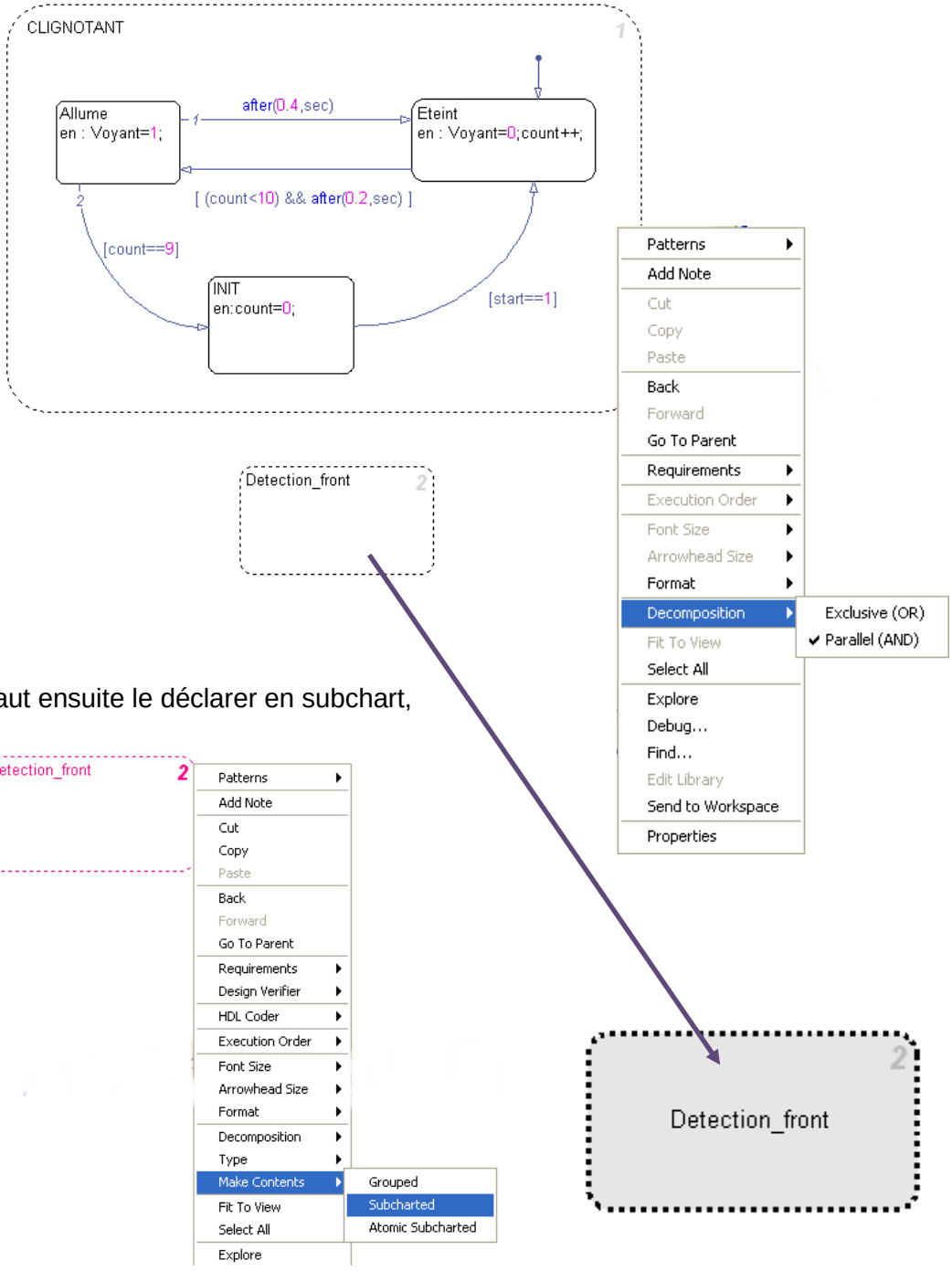
Ici nous allons la spécialiser dans la détection du front de l'entrée start.

Les étapes du projet :

1) Ajouter un état dénommé Détection_front.
2) Renseigner son contenu, ici la détection des fronts.
3) Modifier la modèlisation simulink pour intégrer la détection du front de start.
4) Tester la simulation.

5.1 Ajouter l'état subchart

Nous ajoutons le nouvel état, le nommer **Detection_front** puis vérifier qu'il est bien en décomposition parallèle (AND) ce qui signifie que les états peuvent être actifs simultannément. Le contour de ces états est tracé en pointillé.



Il faut ensuite le déclarer en subchart,

Le fond est coloré pour indiquer la nature de l'état en tant que diagram stateflow complet. Il nous faut maintenant définir son contenu.

5.2 Renseigner le contenu du subchart.

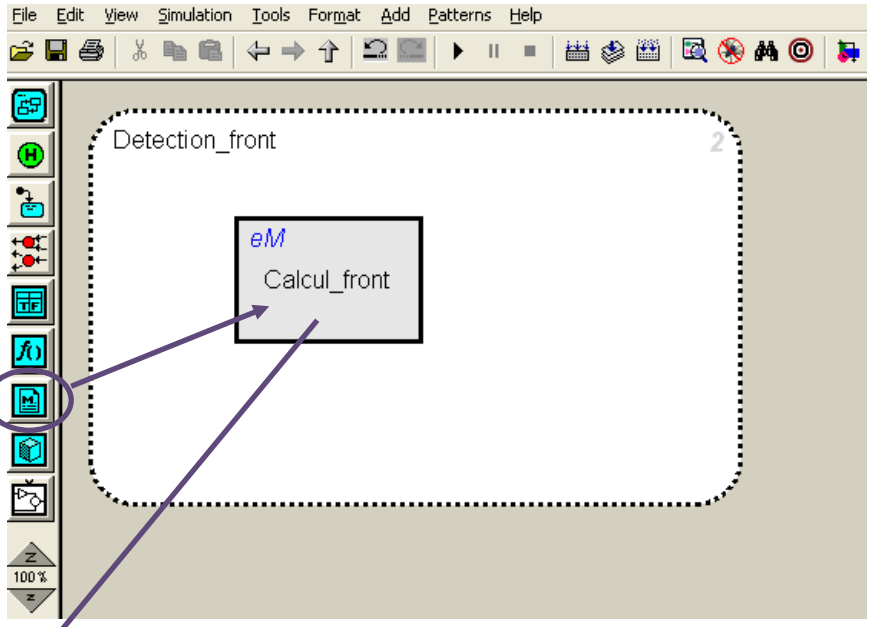
Une fois l'état déclaré en subchart il faut définir son contenu. Pour cela double cliquer dedans pour accéder à la description interne.

Cet outil permet de remonter au niveau supérieur

Dans notre projet il faut surveiller une entrée pour pouvoir détecter les changements de niveaux.

Nous allons utiliser une fonction décrite dans le langage script m de matlab.

Tout d'abord insérer la boite fonctionnelle dans le subchart et la nommée Calcul_front.

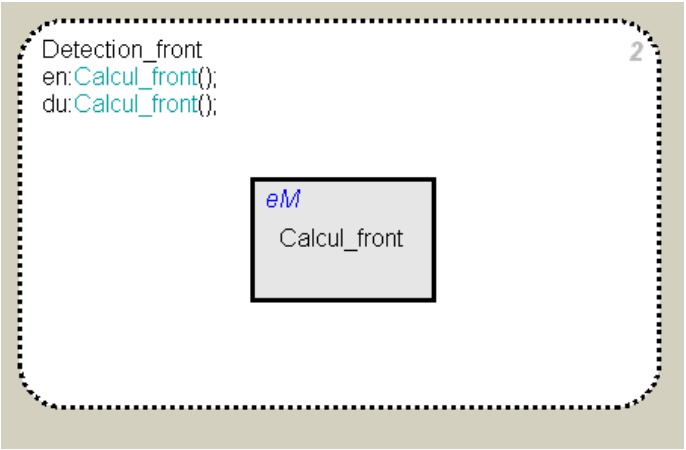


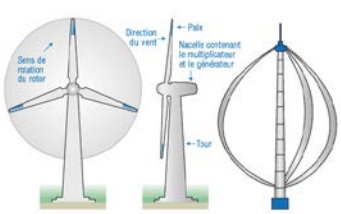
Pour définir le contenu du script de nouveau on double clique dans la boite eM, et on y insère ce code :

```
function Calcul_front
if (start~=start_old) start_front=1; else start_front=0; end;
start_old=start;
```

Ne pas oublier au préalable de définir les deux variables start_front et start_old en variables locales pour notre modèle stateflow. Utiliser l'explorateur de modèles. **Voir sur la page suivante.**

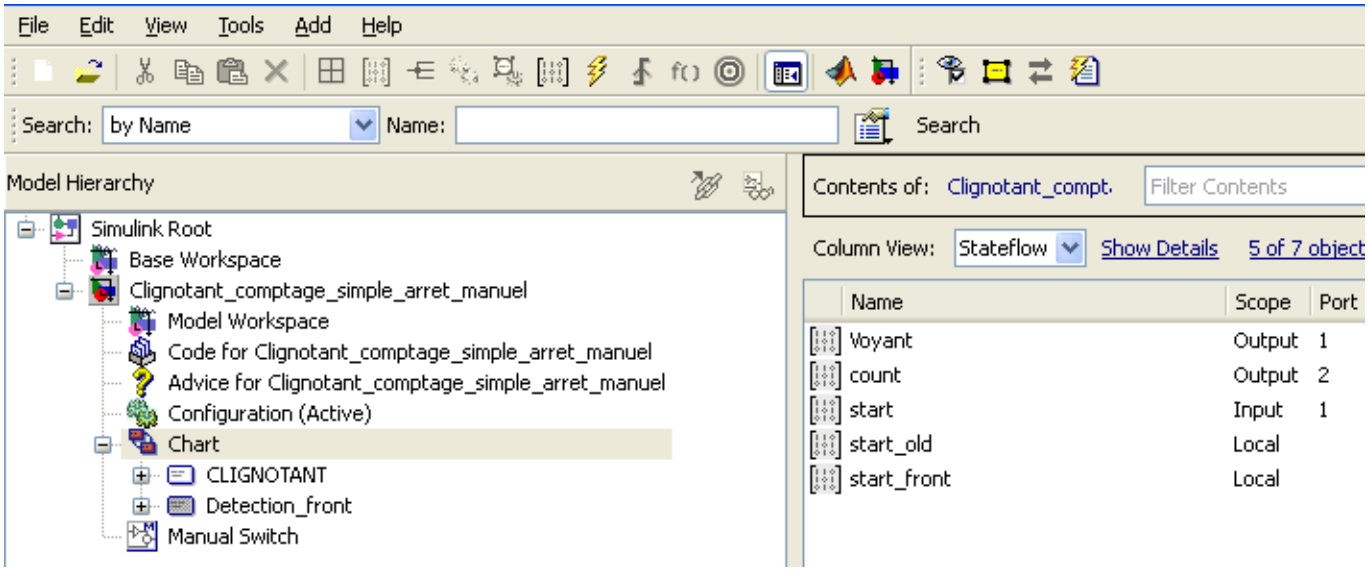
Puis appeler la fonction lors de l'activation de l'état Detection_front en : et pendant la durée de son activation dur : , (ici tout le temps puisque l'état n'est jamais désactivé.)





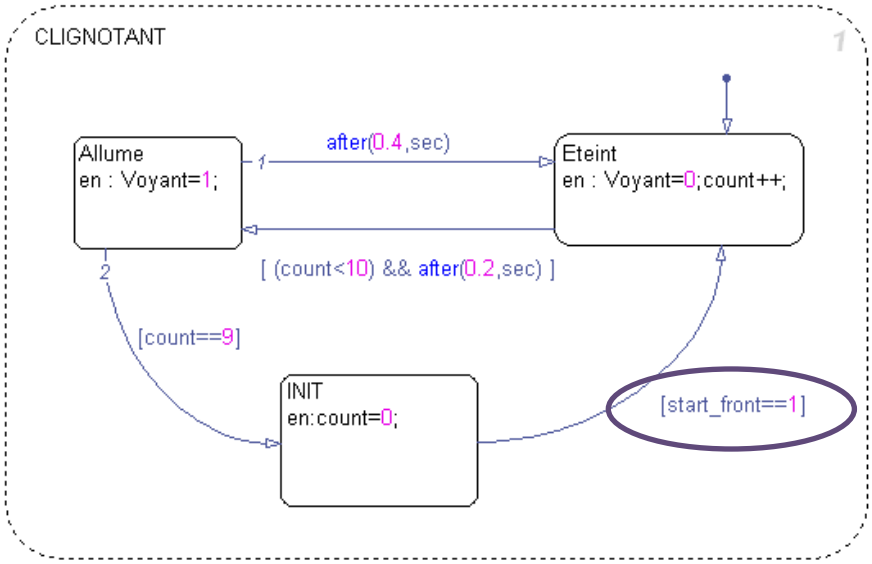
Ajout des variables dans le projet avec l'explorateur de modèle:

On ajoute start_old puis start_front avec le scope local **au niveau du chart** :



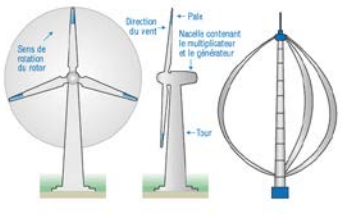
5.3 Modification du diagramme stateflow.

Pour tenir compte de la détection des fronts il faut modifier la modélisation stateflow pour réagir sur start_front au lieu de start. Modifions le diagramme en conséquence :

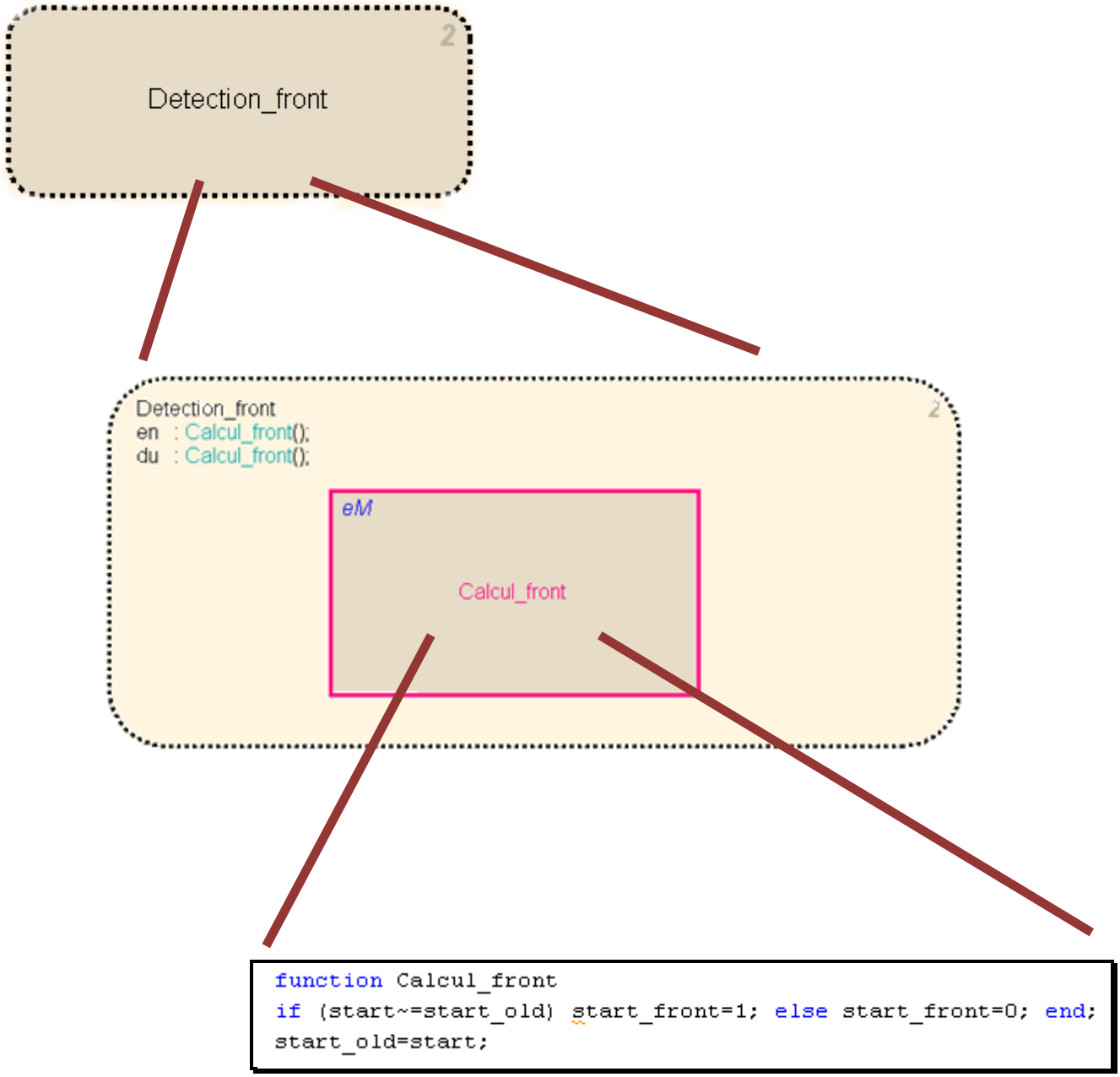


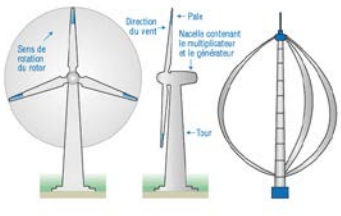
5.4 Tests finaux.

Tester le comportement de la nouvelle simulation. Que se passe t-il par rapport au comportement précédent ?

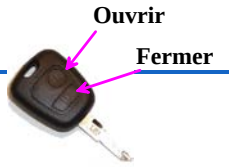


Une vue générale de l'organisation de notre subchart Detection_front





Fermeture centralisé des portières



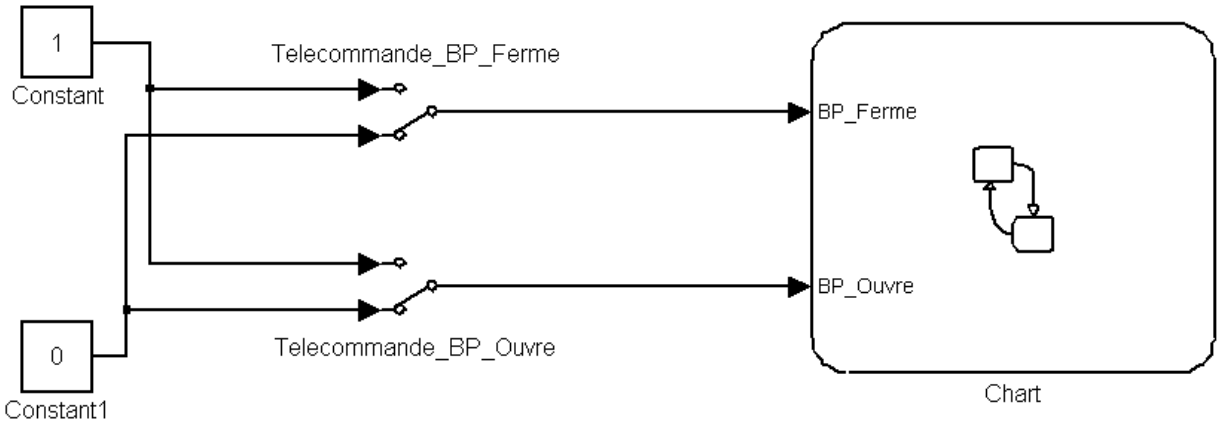
Nous pouvons mettre en pratique nos connaissances pour décrire un automate de gestion d'ouverture de portières d'un véhicule.

6 Description des entrées sorties du projet

Les entrées de notre automate sont les deux boutons de la télécommande BP_Ouvre et BP_Ferme. Nous utiliserons sur ces entrées la technique de détection des fronts vue précédemment. L'automate agit sur deux sorties, le verrouillage des portières et le clignotant.

6.1 Préparation des entrées

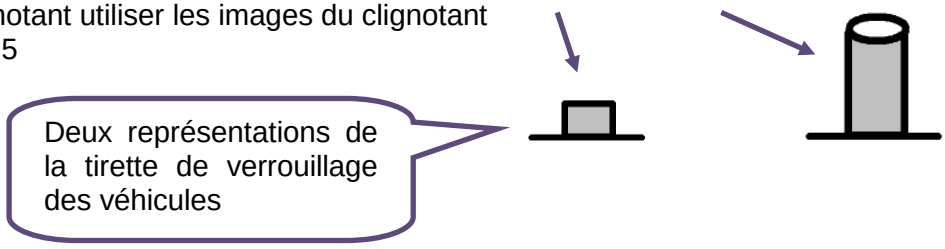
Définir une modélisation simulink avec les deux entrées BP_Ouvre et BP_Ferme selon le schéma ci-dessous :



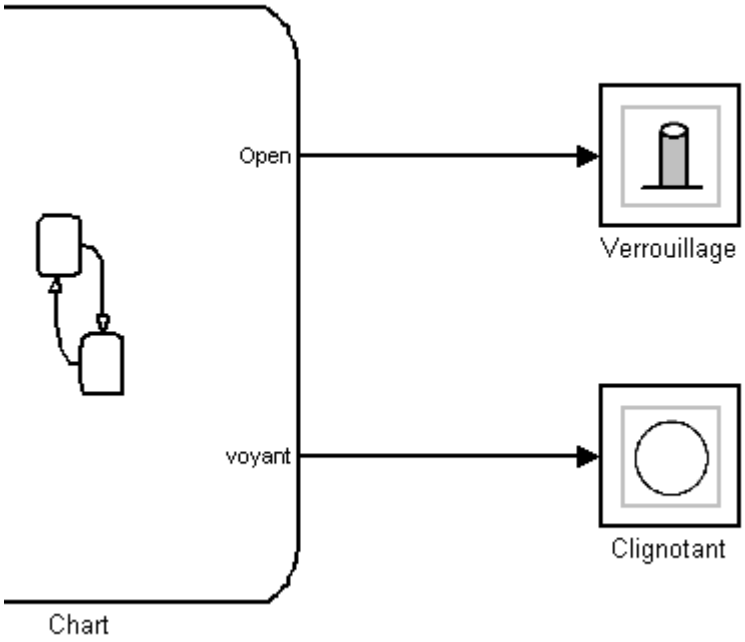
6.2 Préparation des sorties

Nous avons deux sorties dans notre automate, une dénommée Open pour le verrouillage des portières et la seconde dénommée voyant pour le clignotant. Pour ces deux sorties nous utilisons la led issue de la boite à outils gauge_blockset, que nous allons pouvoir customiser selon la technique vue en 1.5.

- Pour le verrouillage des portières mettre les images Fermé.bmp et Ouvert.bmp
- Pour le clignotant utiliser les images du clignotant simulé en 1.5



Voilà les sorties positionnées :



6.3 Analyse du fonctionnement de l'automate

A partir de la situation sur la page suivante répondre aux questions ci-dessous :

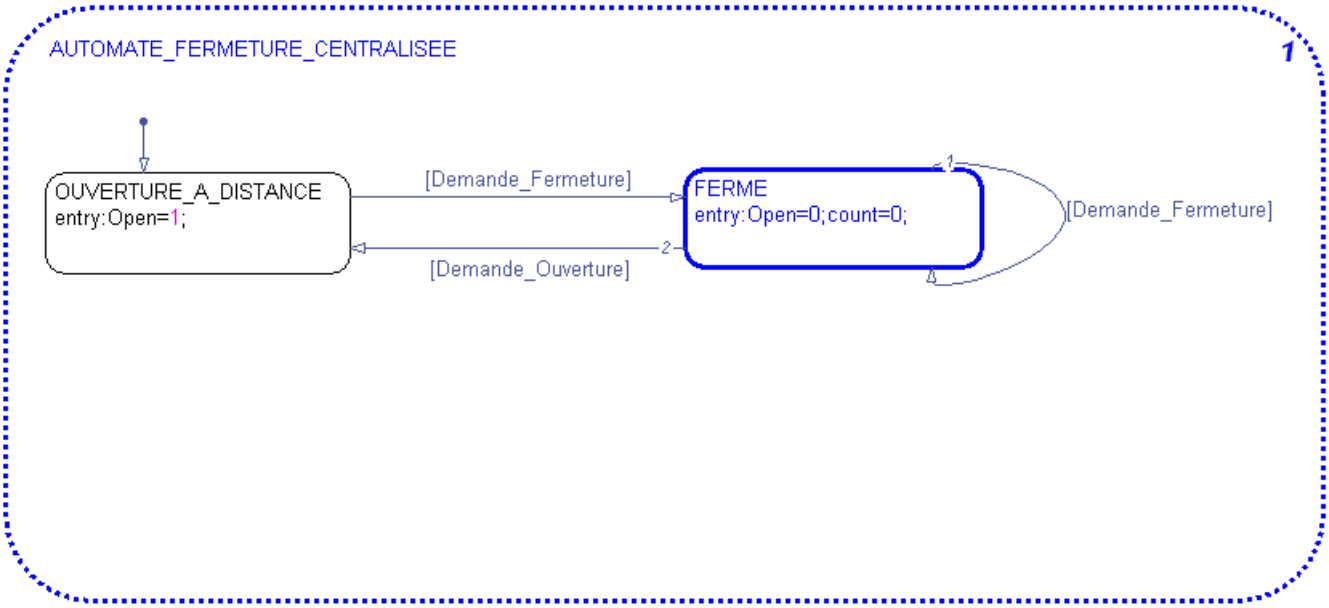
1. Dans quel état est la serrure du véhicule ?

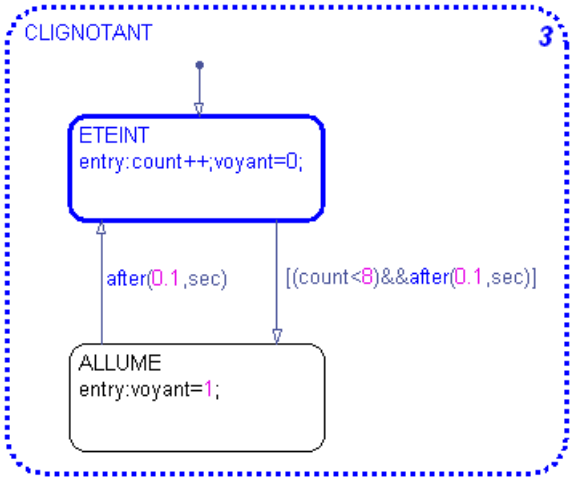
2. Quels sont les états actifs ?

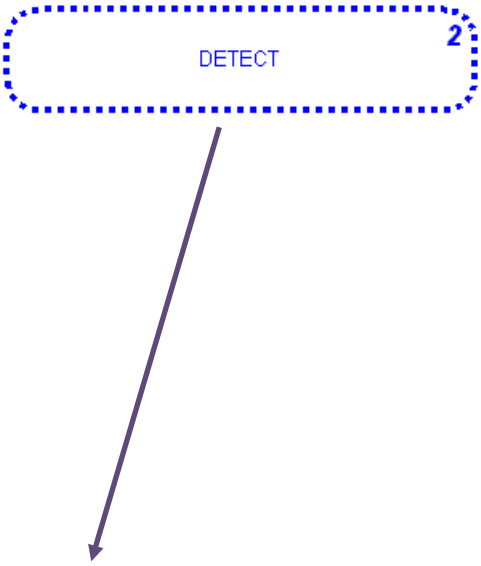
3. Que se passera t-il si le propriétaire du véhicule appui sur la fonction Fermeture de sa télécommande ?

4. Et si le propriétaire appui sur la fonction ouverture de sa télécommande ?



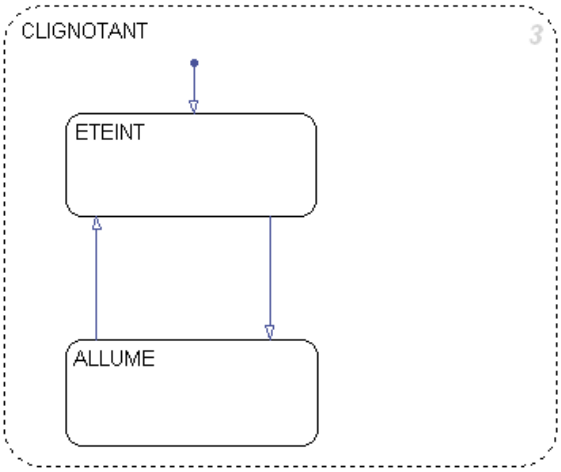




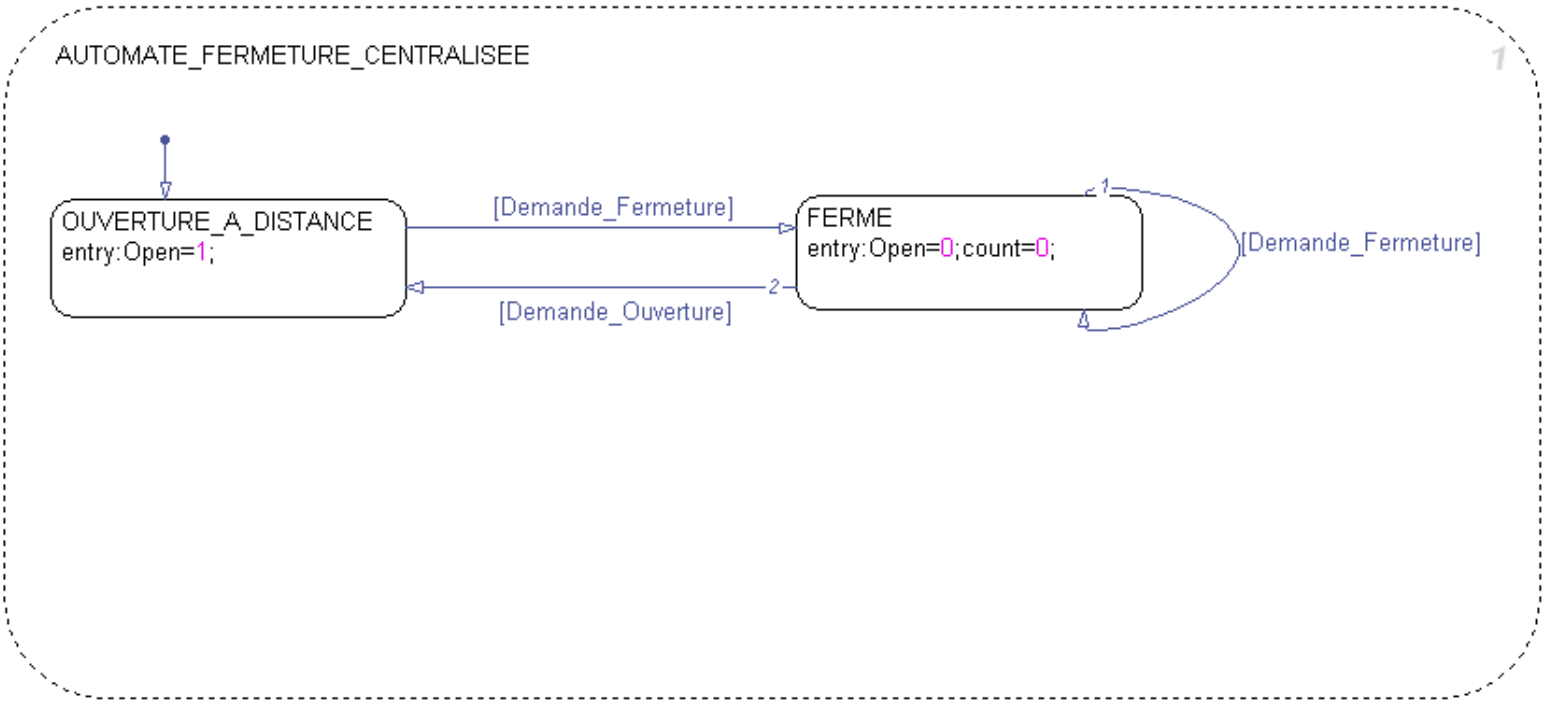

```
function buffer_data
if (BP_Ouvre~=BP_Ouvre_old) Demande_Ouverture=1; else Demande_Ouverture=0; end;
BP_Ouvre_old=BP_Ouvre;
if (BP_Ferme~=BP_Ferme_old) Demande_Fermeture=1; else Demande_Fermeture=0; end;
BP_Ferme_old=BP_Ferme;
```

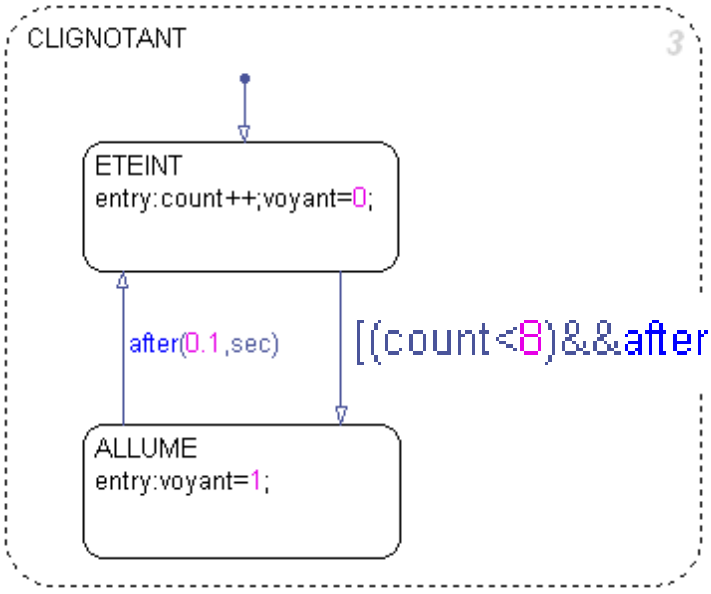
6.4 Saisie du graphe de la machine d'états

La machine d'état est incomplète, à vous de jouez !



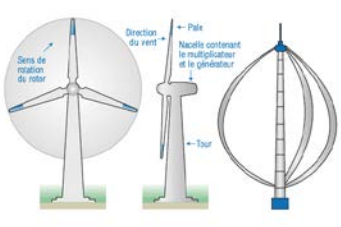
Pour vous aider voilà la vue de la machine d'état correctement saisie sur la page suivante :





Que signifie cette condition associée à la transition ETEINT / ALLUME ?

Tester votre machine d'état une fois la saisie terminée, faire valider par le professeur.



Amélioration de l'automate de serrure de véhicule

7 Amélioration de l'automate de serrure

Sur les véhicules l'automate est un peu plus sophistiqué en effet si nous ouvrons le véhicule mais que nous n'ouvrons pas de portière alors le véhicule se referme automatiquement au bout d'un certain temps. Vous aller ajouter cette fonctionnalité au projet précédent.

Pour cela suivre les étapes décrites ci-après :

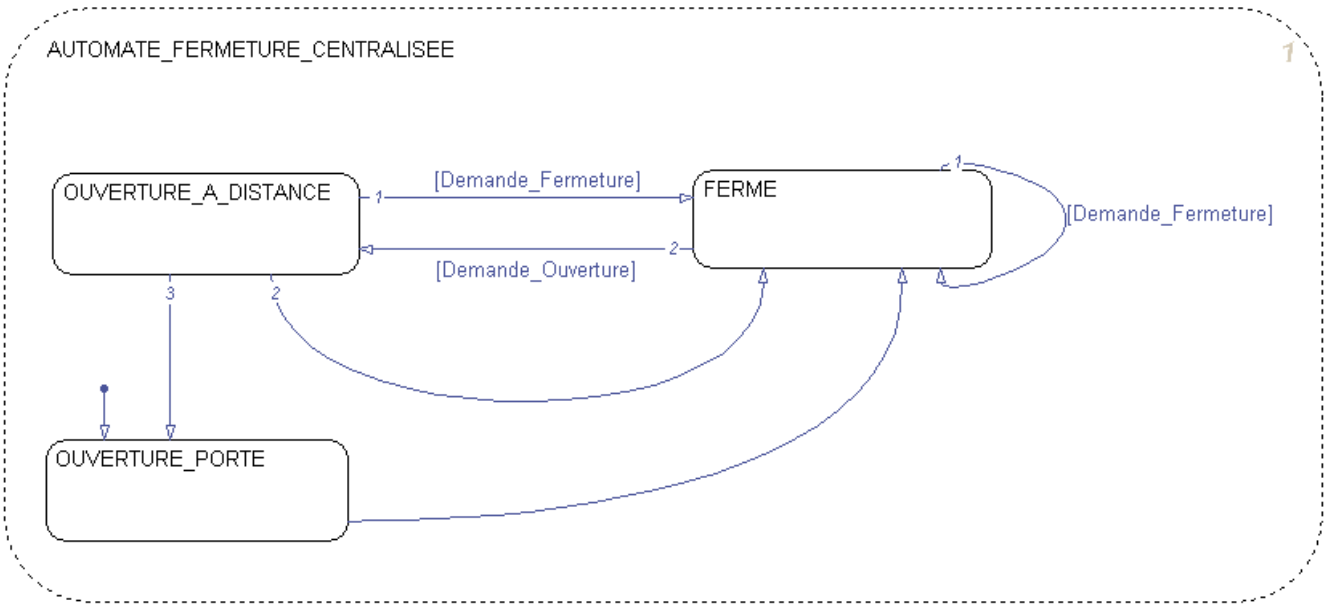
1. Sauvegarder votre projet précédent sous un nouveau nom, pour conserver la solution du travil précédent.

2. Ajouter une nouvelle entrée Capt_Porte qui détectera l'ouverture physique d'une portière.

3. Compléter la fonction Buffer_data pour la surveillance d'un front sur la nouvelle entrée, il faudra ajouter les variables internes : `CAPT_Porte_old et Ouverture_Porte` puis saisir le code :

```
if (CAPT_Porte~=CAPT_Porte_old)
      Ouverture_Porte=1;
else
      Ouverture_Porte=0;
end;
CAPT_Porte_old=CAPT_Porte;
```

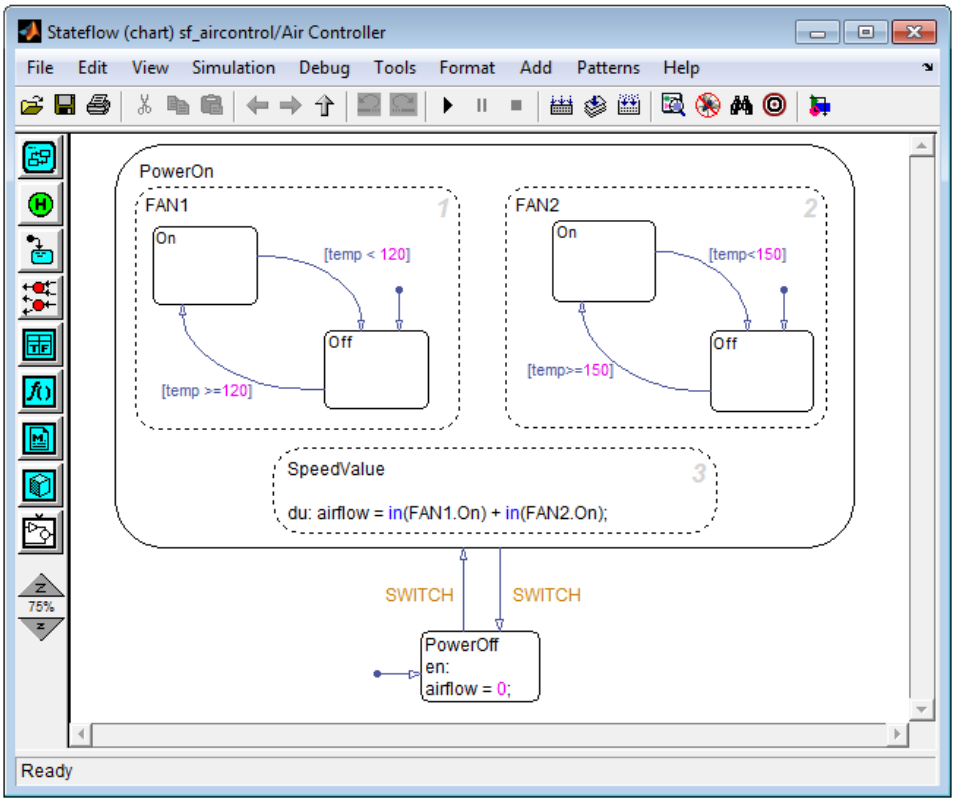
4. Modifier l'automate pour ajouter l'état OUVERTURE_A_DISTANCE et la gestion correspondante, le début de la solution est donnée ci-dessous :



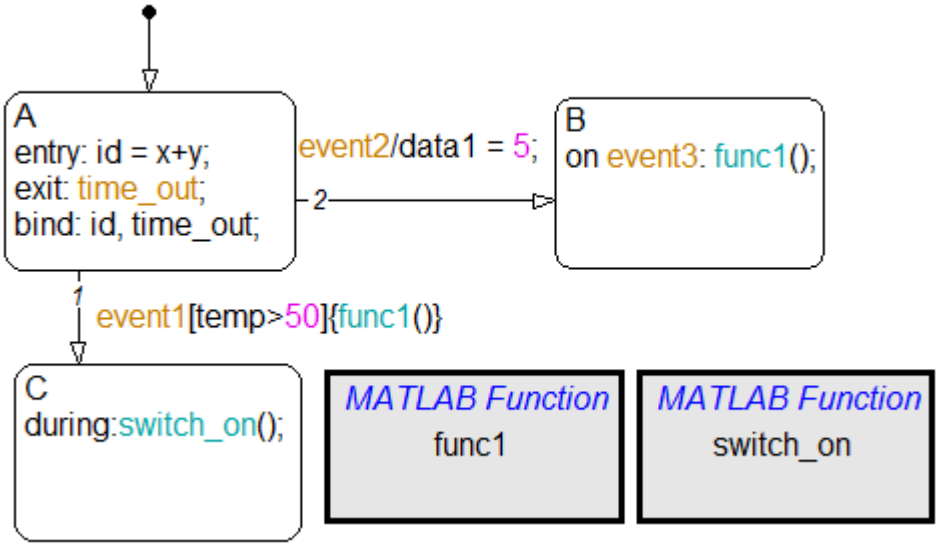
Faites vérifier par le professeur.

Résumé stateflow

1 Premier aperçu d'un chart stateflow



2 Les différentes actions possibles



State Action	Abbreviation	Description
`entry`	`en`	Executes when the state becomes active
`exit`	`ex`	Executes when the state is active and a transition out of the state occurs
`during`	`du`	Executes when the state is active and a specific event occurs
`bind`	none	Binds an event or data object so that only that state and its children can broadcast the event or change the data value
`on`*`event_name`*	none	Executes when the state is active and it receives a broadcast of *`event_name`*
`on after(`n, *`event_name`*`)`	none	Executes when the state is active and after it receives n broadcasts of *`event_name`*
`on before(`n, *`event_name`*`)`	none	Executes when the state is active and before it receives n broadcasts of *`event_name`*
`on at(`n, *`event_name`*`)`	none	Executes when the state is active and it receives exactly n broadcasts of *`event_name`*
`on every(`n, *`event_name`*`)`	none	Executes when the state is active and upon receipt of every n broadcasts of *`event_name`*

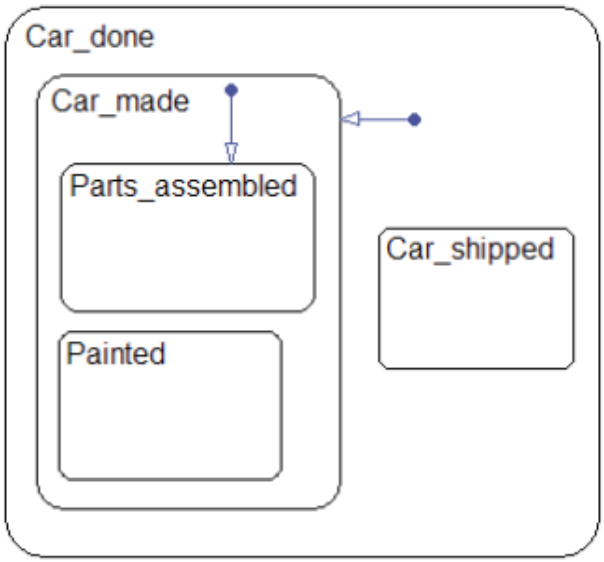
A l'activation du state A : id = x + y ;

D'après StateFlow user guide.pdf

3 Hiérarchie des états

3.1 Exemple de hiérarchie et identification des états

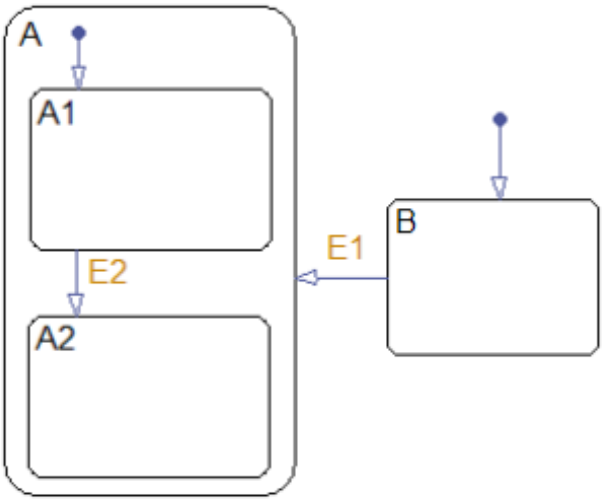
Un état est symbolisé par un rectangle aux coins arrondis, le trait est continu (exclusif OR) ou pointillé (parallel AND).



- /Car_done
- /Car_done.Car_made
- /Car_done.Car_shipped
- /Car_done.Car_made.Parts_assembled
- /Car_done.Car_made.Painted

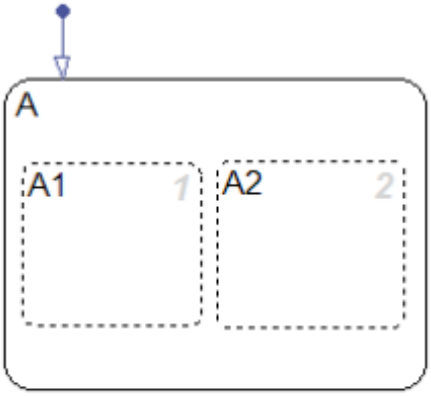
3.2 Décomposition avec des états exclusifs

(OR exclusif state decomposition)

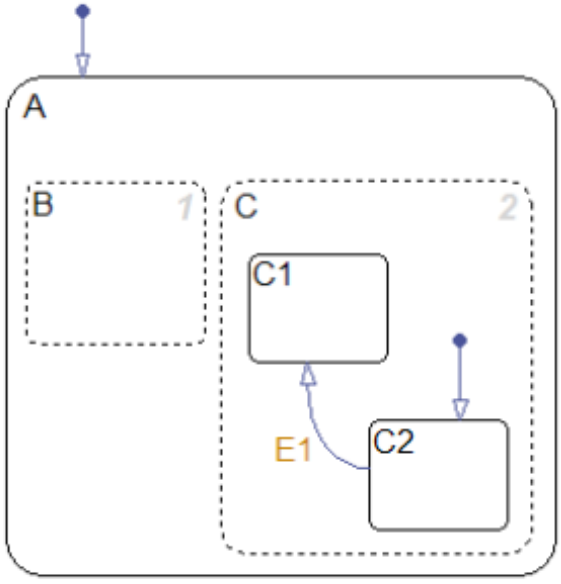


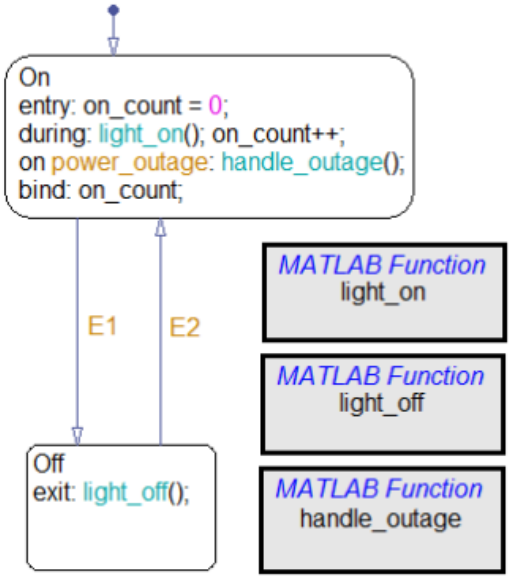
3.3 Décomposition parallèle

(Parallel AND state decomposition)



3.4 Exemple complet :





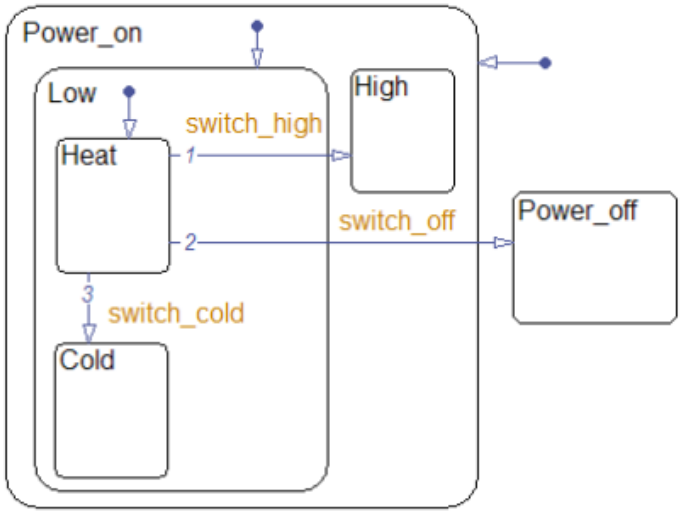
3.5 Identification des états : state label

```
name/
entry:entry actions
during:during actions
exit:exit actions
on event_name:on event_name actions
bind:events, data
```

Bind : lier, attacher

4 Les transitions

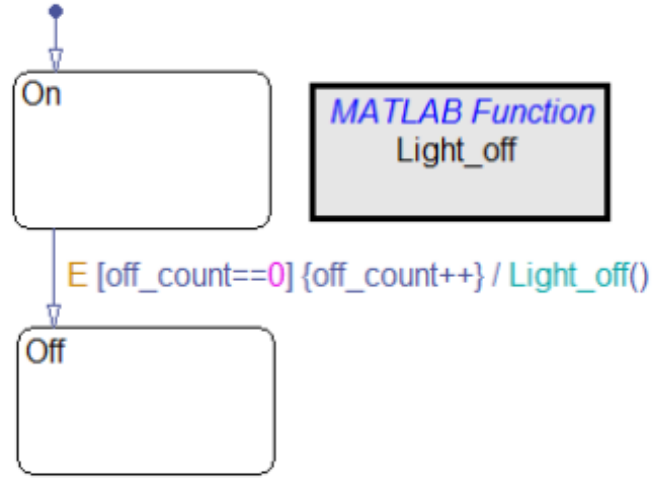
Le passage d'un état d'un un autre soumis à une condition logique. Cette condition est écrite dans le label associé à la transition.

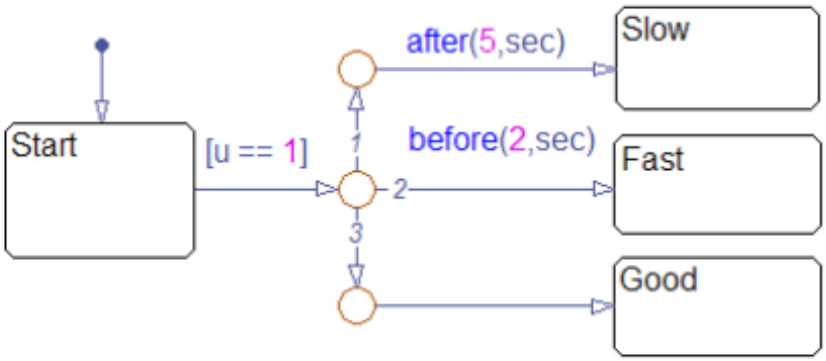


Transition Label	Transition Parent	Transition Source	Transition Destination
switch_off	/	/Power_on.Low.Heat	/Power_off
switch_high	/Power_on	/Power_on.Low.Heat	/Power_on.High
switch_cold	/Power_on.Low	/Power_on.Low.Heat	/Power_on.Low.Cold

event[condition]{condition_action}/transition_action

4.1 Exemple :





If the input equals 1...	A transition occurs from...
Before t = 2	Start to Fast
Between t = 2 and t = 5	Start to Good
After t = 5	Start to Slow

Logique temporelle associée aux transitions (10-63)

En cas de problème de compilation

Pour redéfinir le compilateur faire la commande mex –setup dans la console.

Matlab répond par la détection des compilateurs installés dans l'ordinateur. Choisir le compilateur Microsoft visual c++ installé puis confirmer le choix. Voir l'exemple ci-dessous.

Command Window

New to MATLAB? Watch this Video, see Demos, or read Getting Started.

```
>> mex -setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~2\STI2D\MATLAB\R2010b\sys\lcc
[2] Microsoft Visual C++ 2008 Express in C:\Programmes\CAO\Visual c++

[0] None

Compiler: 2

Please verify your choices:

Compiler: Microsoft Visual C++ 2008 Express
Location: C:\Programmes\CAO\Visual c++

Are these correct [y]/n? y
```