# INTRODUCTION A STATEFLOW® R2019b

*A mettre entre toutes les mains, entre celles des curieux et celles de ceux qui s'y intéressent…*

## Modélisation & Simulation des Systèmes à Evènements Discrets (et Continus).

Ph. Hautcoeur

*Novembre 2019*


« De même qu'il faut apprendre à apprendre individuellement, il faut apprendre à apprendre collectivement, il faut créer les conditions où on peut apprendre les uns des autres… »

François TADDEI présente son livre « Apprendre au XXIe siècle » publié chez Calmann-Lévy.

https://www.youtube.com/watch?v=BhunNczVNaA

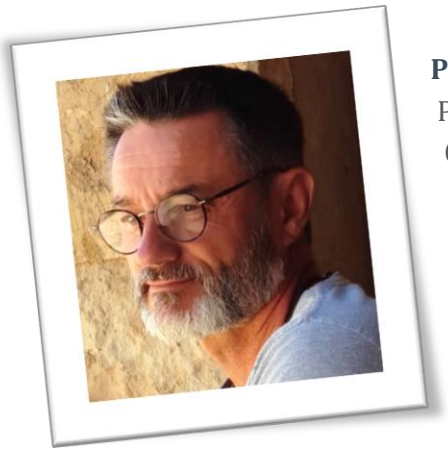
Cet ouvrage revisite entièrement la version précédente publiée en 2014 (Version R2013a de la suite Matlab® Simulink®) et téléchargeable sur la plateforme « File Exchange » de Mathworks ®.

http://bit.ly/IntroStateflow

Cette nouvelle édition s'articule de manière similaire à la précédente et reprend les principales évolutions depuis la version R2013b de la suite Matlab® Simulink®. Ce manuel a donc pour objectif de présenter les fonctionnalités de base de l'outil illustrées par des exemples simples.

Les fichiers associés à la plupart des exemples developpés dans ce cadre sont téléchargeables avec ce document sur la plateforme « File Exchange » de Mathworks ®.

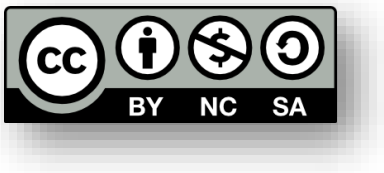
Puisse cet ouvrage contribuer, modestement, à la construction d'une société apprenante…



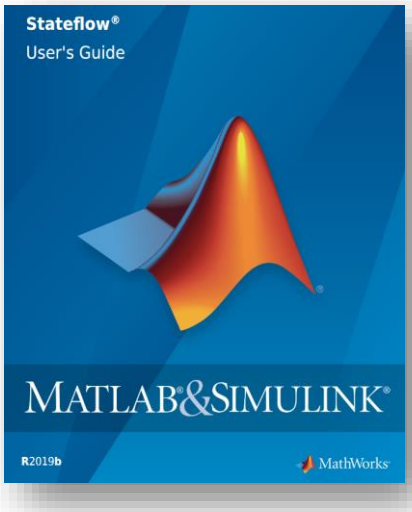
**Philippe Hautcoeur**
Professeur de Sciences Industrielles de l'Ingénieur
Classes Préparatoires aux Grandes Ecoles - PSI
Lycée Clemenceau à Nantes

*Ce document évolue grâce à votre concours.*
*>>>> contact : philippe.hautcoeur@ac-nantes.fr*

*A celles et ceux qui m'ont accompagné dans ce projet.*



# Avant-propos…




**Stateflow®** est un module développé par la société américaine **MathWorks®** qui permet la simulation de machines d'état.

Une machine d'état comporte un nombre fini d'états. Elle modélise le comportement de systèmes qui passent d'un état à un autre en réponse à des évènements. On parle alors de systèmes à évènements discrets.

Comme son nom l'indique, ce module permet de tracer des diagrammes d'état (« State Chart ») et des diagrammes de flux (« Flow Chart »).

**Stateflow®** est intégré à **Matlab® et Simulink®**. Les modèles construits pourront par conséquent comporter des blocs des différentes « toolboxes » de Simulink et/ou appeler des **fonctions Matlab** et/ou des **fonctions Simulink** comme nous le verrons. Ainsi le modèle global d'un système complexe pourra comporter des **modèles linéaires continus** construits avec Simulink sous la forme de schéma-blocs, **des machines à état** construites avec Stateflow® ou encore des **modèles acausaux** réalisés en utilisant **Simscape®**.

Aussi Stateflow® permet de simuler le comportement de **systèmes hybrides** c'est-à-dire à évènements **discrets et continus**. C'est par exemple le cas d'une balle qui rebondit sur le sol. En effet son déplacement dans l'air est continu alors qu'à chaque rebond, considéré comme un évènement, sa trajectoire est modifiée. C'est encore le cas si un robot doit éviter un obstacle présent sur sa trajectoire.

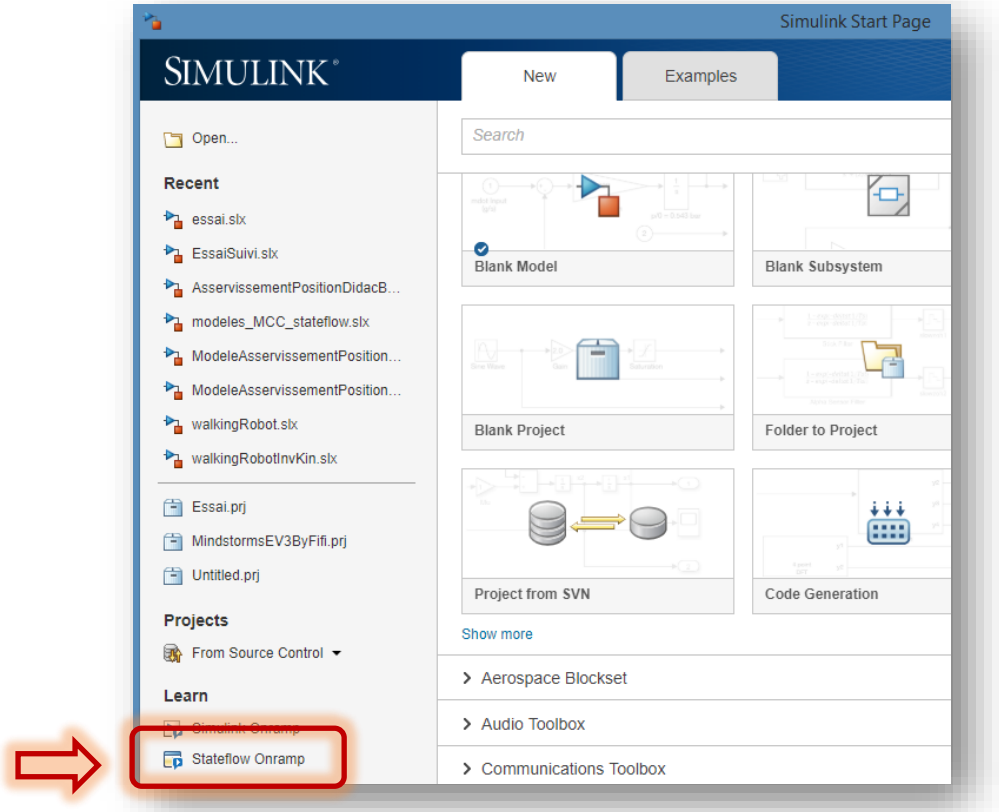
Une connaissance approfondie de Matlab® et Simulink® n'est pas indispensable pour commencer à travailler avec Stateflow®. Dans la plupart des exemples traités avec la version **R2019b**, les chemins menant aux composants des bibliothèques Simulink® utilisées sont précisés.

Après avoir présenté les différents outils et quelques applications, nous verrons comment, dans le cadre d'une démarche Model Based Design, implémenter un programme réalisé avec Stateflow® vers une cible telle que la brique **LEGO® Mindstorms EV3.**

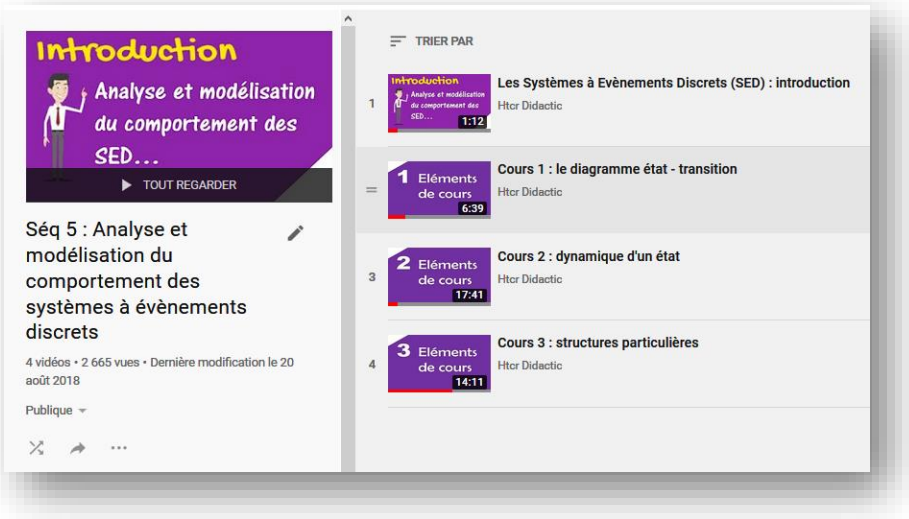
Le guide de l'utilisateur complet de Stateflow® est téléchargeable sur le site de MathWorks® :

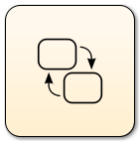
http://www.mathworks.com/help/pdf_doc/stateflow/sf_ug.pdf

Aussi, ce document peut être un outil complémentaire à la formation en ligne (en anglais) proposée par **MathWorks® : Stateflow Onramp** à laquelle on accède à partir de la page de démarrage de **Simulink®** (Simulink Start Page) en lançant d'abord **Matlab®** puis **Simulink® :**



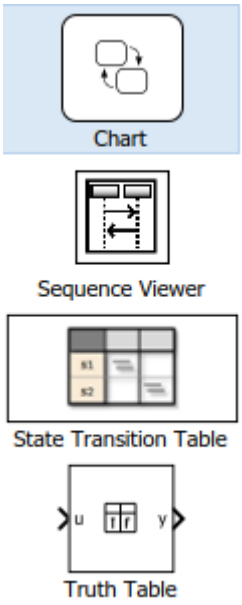

Enfin, pour appréhender le fonctionnement d'une machine d'état, vous pouvez vous rendre sur ma chaîne YouTube et visualiser la playlist consacrée à l'analyse et la modélisation du comportement des systèmes à évènements discrets en général, et au diagramme d'état SysML en particulier : http://bit.ly/StmDiagram





# Chapitre 1

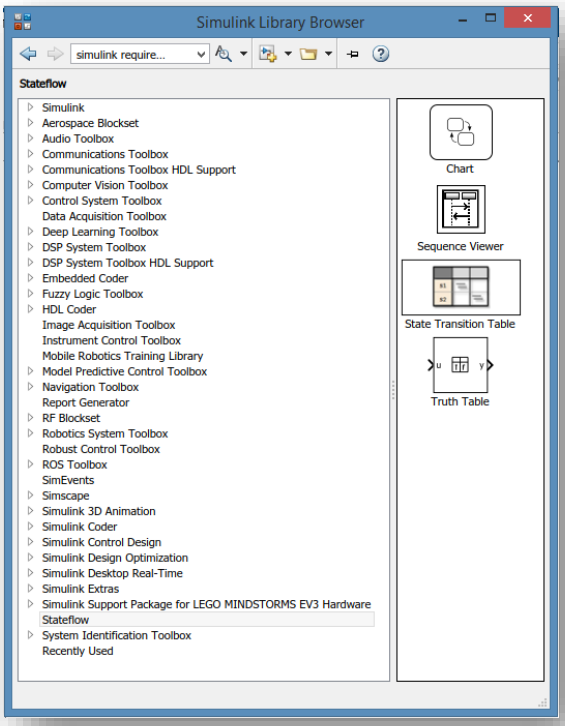
# *Pour bien commencer*



# Chapitre 1 - Pour bien commencer…

## 1.1- PREPARATION ET PRESENTATION DE L'INTERFACE

### *A propos de la préparation…*

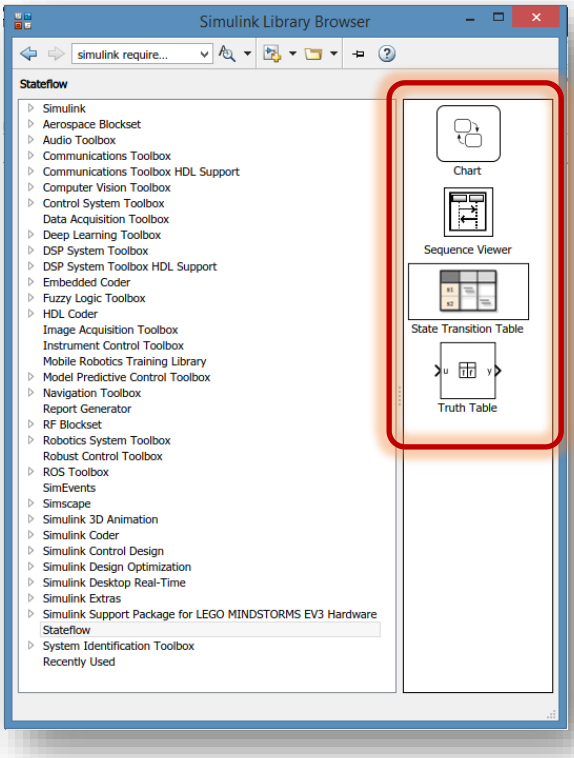



Pour fonctionner, Stateflow® nécessite l'installation d'un compilateur C.

Pour vérifier si vous disposez d'un compilateur sur votre machine ou pour l'installer, il faut lancer Matlab® puis, dans la fenêtre de commande, saisir la ligne de commande suivante et suivre les instructions proposées :

```
Trial>> mex -setup

Welcome to mex -setup.  This utility will help you set up
a default compiler.  For a list of supported compilers, see
http://www.mathworks.com/support/compilers/R2013b/win64.html

Please choose your compiler for building MEX-files:

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

Select a compiler:
[1] Microsoft Software Development Kit (SDK) 7.1 in C:\Program Files (x86)\Microsoft Visual Studio 10.0

[0] None

Compiler: 1

Please verify your choices:

Compiler: Microsoft Software Development Kit (SDK) 7.1
Location: C:\Program Files (x86)\Microsoft Visual Studio 10.0

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

Compilateur installé sur la machine




La bibliothèque Stateflow® comporte quatre éléments : « **Chart** », « **Sequence Viewer** », « **State Transition Table** » et enfin « **Truth Table** ».

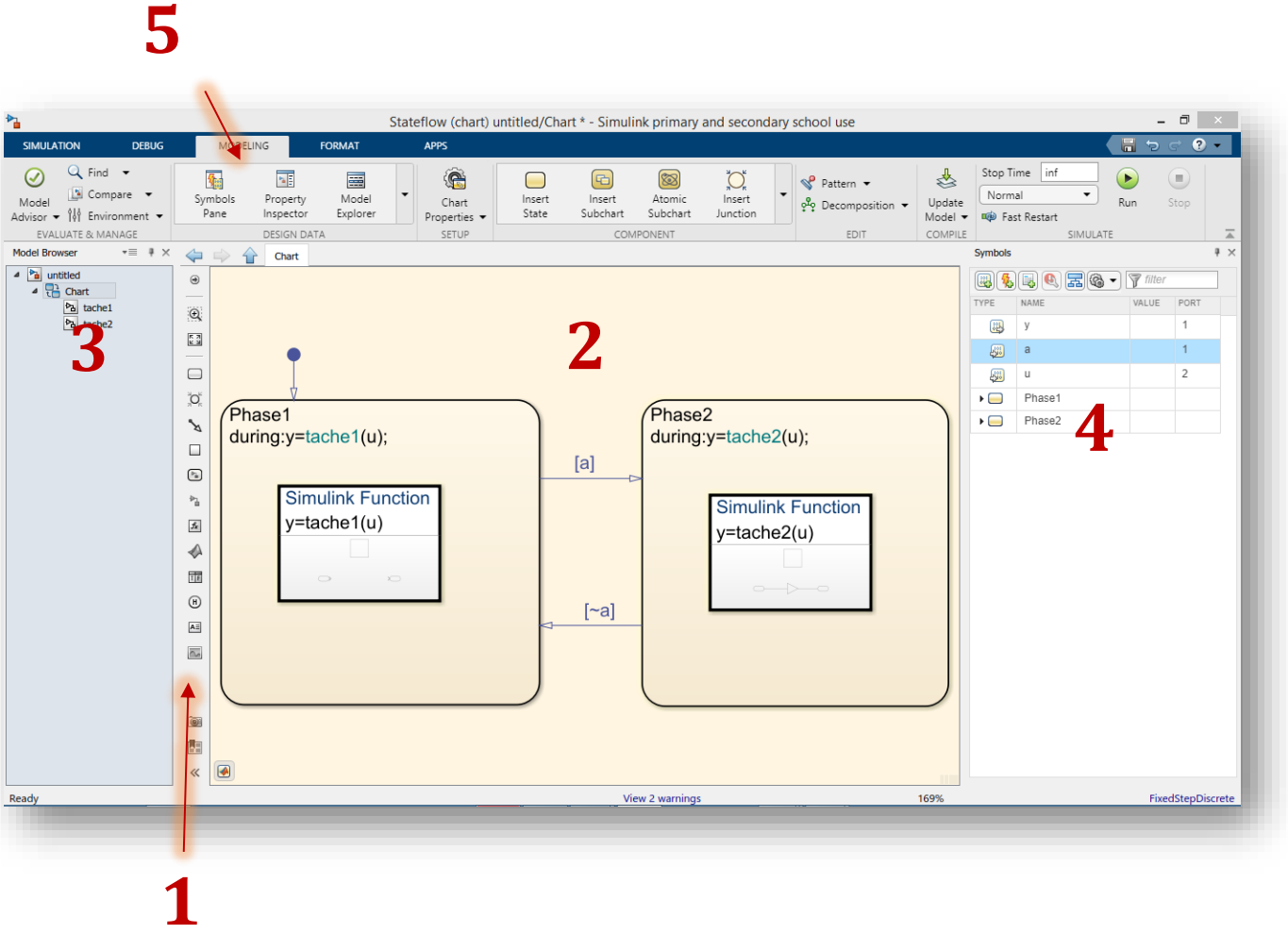
- « **Chart** » permet de construire un diagramme état– transition.
- « **Sequence Viewer** » permet la construction d'un diagramme de séquence.
- « **State Transition Table** » génère automatiquement un diagramme d'état-transition à partir d'une table qui précise tous les états et toutes les transitions.
- Enfin « **Truth Table** » permet la création de table de vérité qui décrira le comportement combinatoire d'un système.

Dans ce document nous passerons en revue ces quatre éléments de la bibliothèque de Stateflow® pour en présenter les principes à partir d'exemples.

Cependant nous nous intéresserons de manière plus approfondie à l'utilisation du bloc « **Chart** » pour la création d'un modèle. Les savoir-faire alors apportés pourront faciliter la prise en main des trois autres éléments mis à disposition dans la bibliothèque Stateflow®.

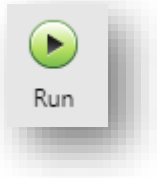
## *A propos de l'interface Simulink®…*

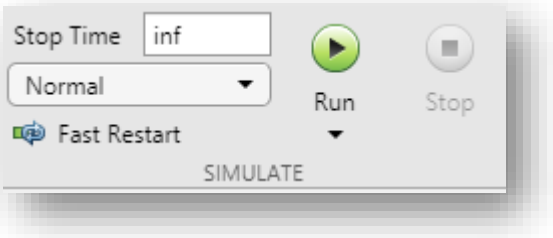
L'interface Simulink® présente cinq zones dont nous allons donner une description sommaire :




- La **zone 1** donne accès à l'ensemble des composants graphiques utiles à la réalisation d'une machine à état.
- La **zone 2** est la zone dans laquelle le diagramme est construit,
- La **zone 3** est un navigateur qui permet d'afficher le contenu des différents éléments du modèle, ici les Fonctions Simulink,
- La **zone 4** permet de gérer les données et évènements de l'ensemble du modèle,
- La **zone 5**, le bandeau supérieur, possède différents onglets et menus associés permettant la construction du modèle et sa simulation. Dans le cadre de cet ouvrage nous utiliserons essentiellement les deux onglets « **SIMULATION** » et « **MODELING** » Je vous invite vivement à naviguer dans le bandeau supérieur pour vous approprier sa logique.

La simulation peut-être lancée à partir des onglets « **SIMULATION** » et « **MODELING** » du bandeau supérieur en cliquant sur l'icône « Run » .



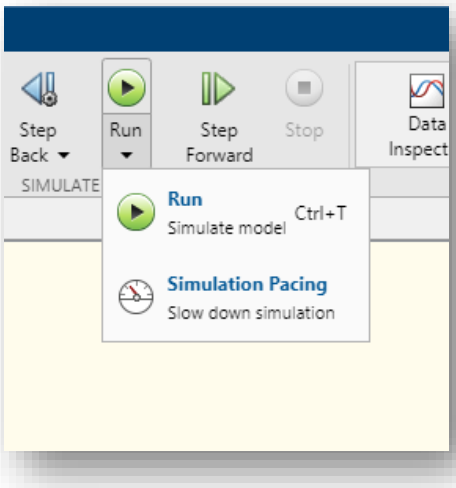



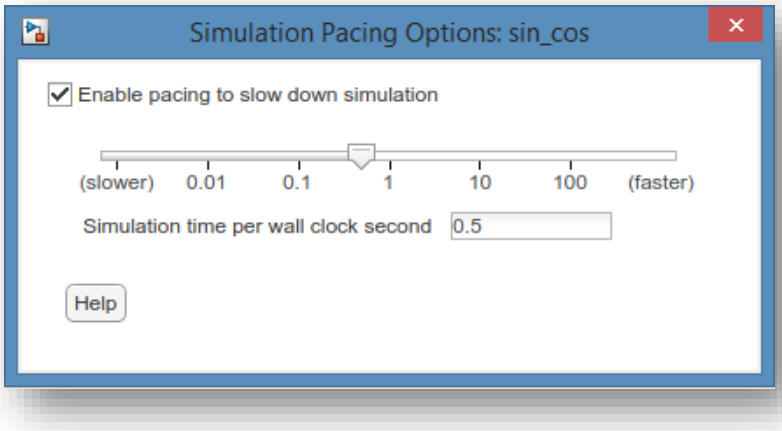

La durée de la simulation est saisie dans le champ « **Stop Time** » du menu « **SIMULATE** » des l'onglets « **SIMULATION** » ou « **MODELING** ». Ici, elle est infinie.

Il est parfois nécessaire de ralentir la simulation pour une bonne visualisation de l'évolution d'un modèle.

Cela est possible à partir de l'onglet « SIMULATION ». En cliquant sur la petite flèche noire sous l'icône « Run ».

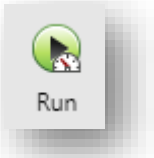

Une fenêtre s'ouvre alors :



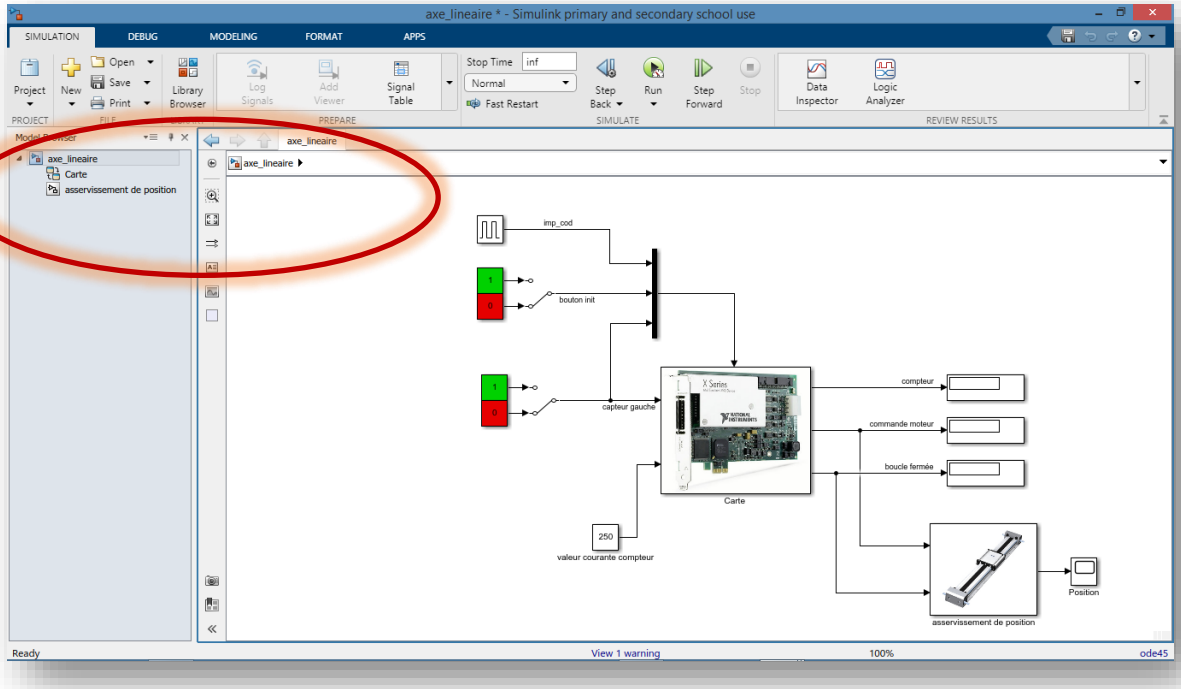



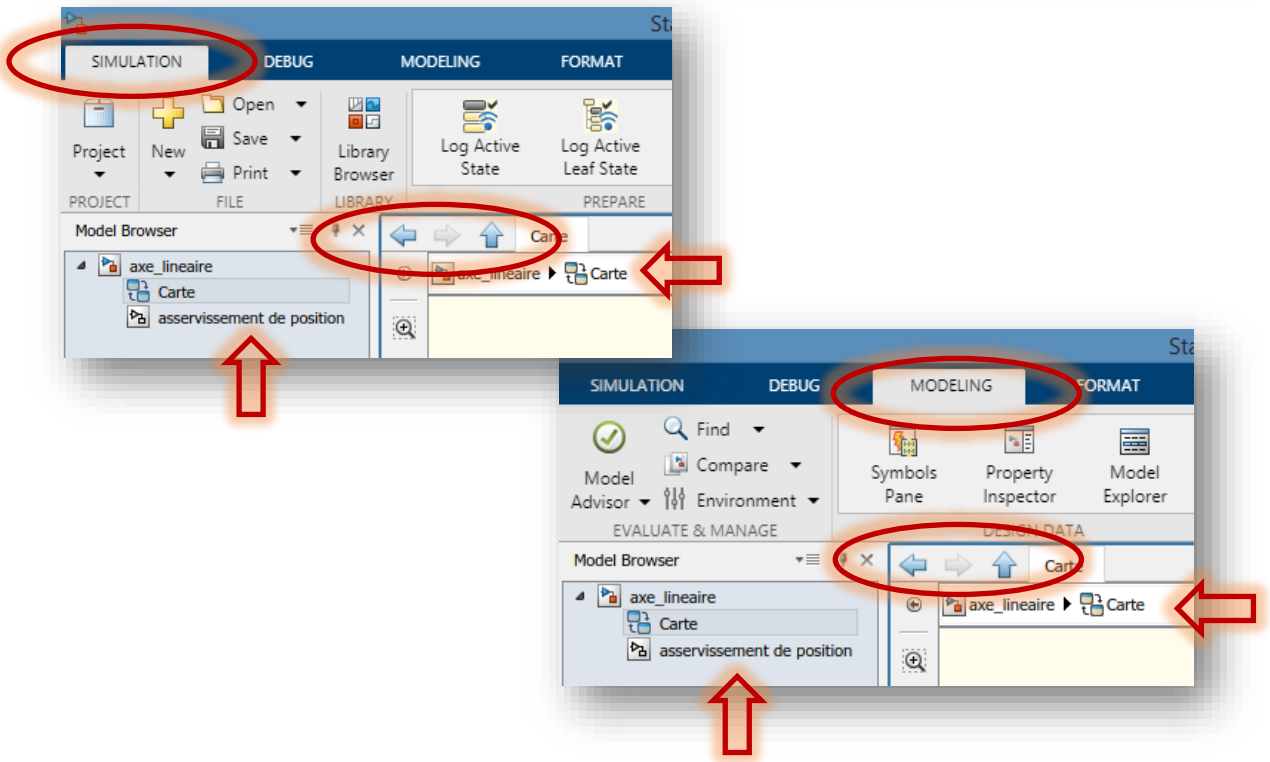

En sélectionnant « **Simulation Pacing** » une autre fenêtre s'ouvre dans laquelle il est possible de régler le temps de simulation :

Après fermeture de cette fenêtre de réglage, l'icône qui permet le lancement de la simulation change. Une hotloge est affectée à la flèche verte. Ce changement s'opère simultanément dans les onglets « **SIMULATION** » et « **MODELING** ».

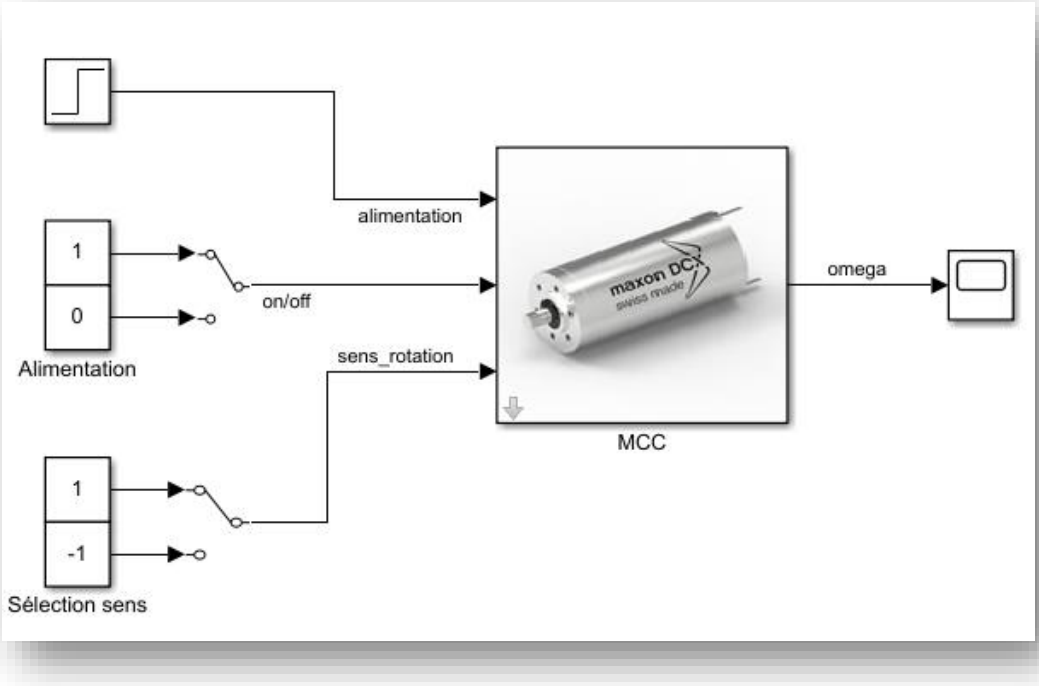



Pour naviguer dans les différents niveaux hiérarchiques du modèle en mode simulation (onglet « **SIMULATION** ») ou en mode modélisation (onglet « **MODELING** »), il est possible d'utiliser les flèches bleues, ou bien le « **Model Browser** » ou encore les onglets situés dans la partie supérieure de la zone graphique.Par exemple :



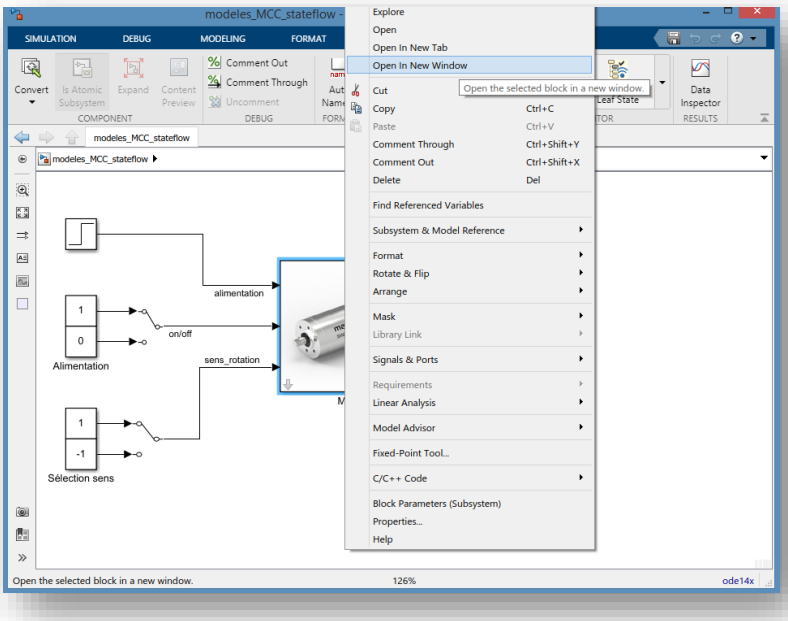



Aussi, afin d'améliorer l'ergonomie et en particulier la visualisation de l'évolution d'une machine à état il est intéressant d'afficher les niveaux hiérarchiques dans des fenêtres différentes. Prenons l'exemple du modèle associé au comportement d'un moteur à courant continu :

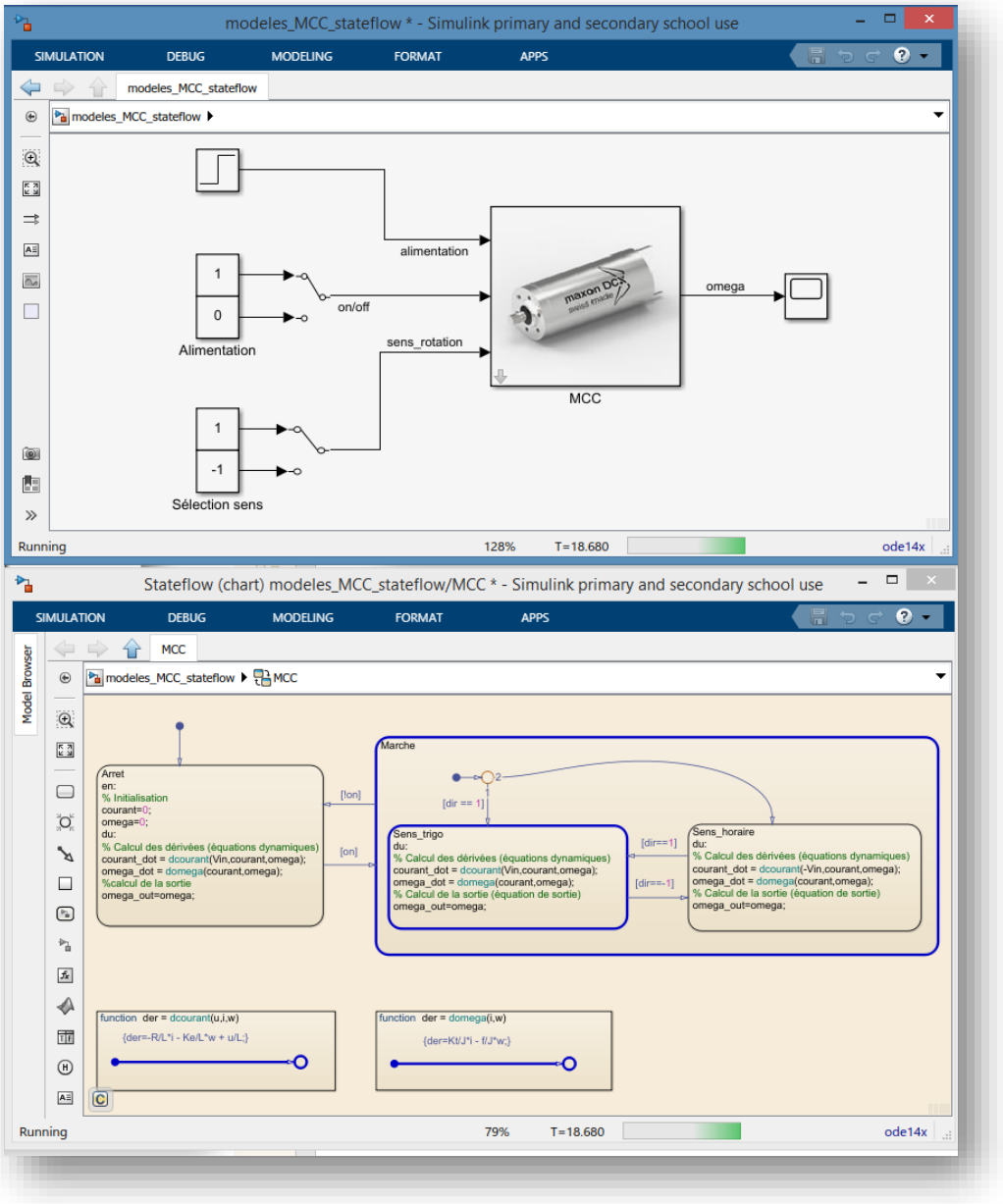



L'interface créée avec **Simulink**® comporte deux **« Switches »** qu'il faudra commuter pour valider notre modèle.

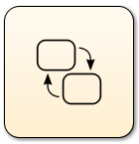
Pour ouvrir le contenu du « **Chart** » associé au bloc MCC dans une nouvelle fenêtre et ainsi visualiser son évolution, il faut sélectionner le bloc par un clic gauche puis ouvrir le menu contextuel avec un clic gauche et sélectionner « **Open in new window** ».



Ainsi il est possible de suivre l'évolution de la machine à état tout en pilotant le moteur avec les switches :

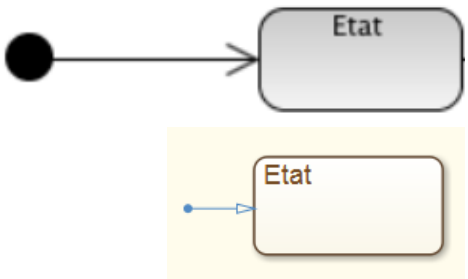



Enfin, je conseille au lecteur de naviguer dans les différents menus et onglets pour découvrir les différentes commandes offertes par le logiciel.



# Chapitre 2

# SysML State Machine vs Stateflow®



# **Chapitre 2** - SysML State Machine vs Stateflow®…

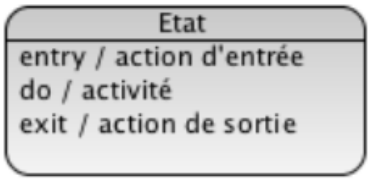
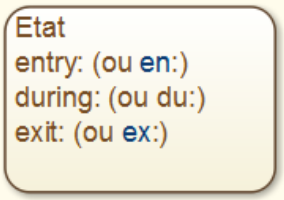
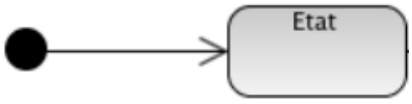
Dans ce dernier chapitre, nous allons mettre en parallèle les objets graphiques utilisés dans la construction des diagrammes d'état avec le langage SysML (Systems Modeling Language) et avec Stateflow®.

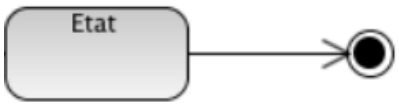
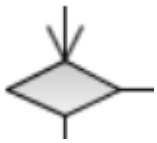
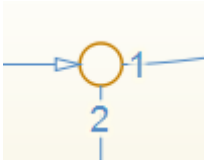
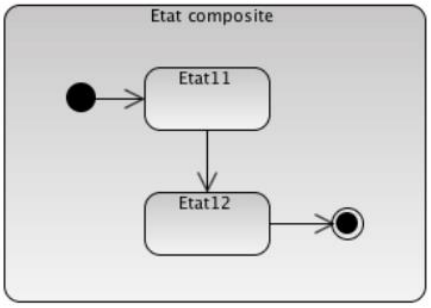
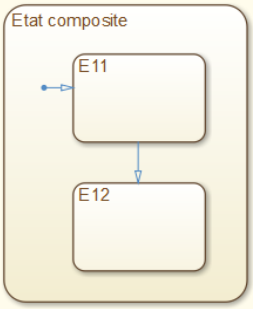
SysML est une extension du langage UML (Unified Modeling Language). Il est dédié à l'ingénierie système. Ce langage est articulé autour de diagrammes propres (diagrammes des exigences, paramétrique…) et de diagrammes issus de UML (cas d'utilisation, diagramme de séquence, diagramme d'état…).

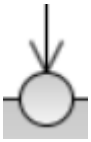
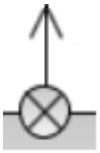
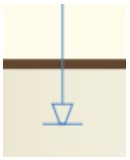
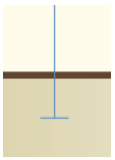
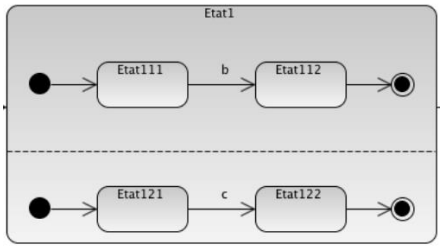
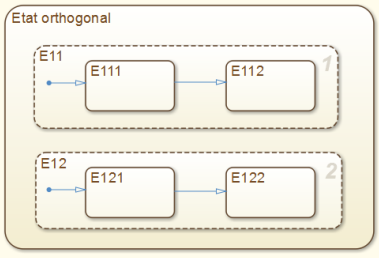
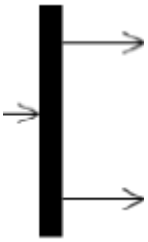
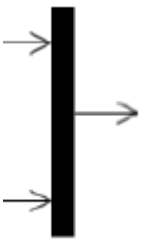
Pour plus de renseignements sur le langage SysML consulter le site officiel :

http://www.omgsysml.org/

Ici nous nous intéresserons uniquement au diagramme d'état (noté stm : « state machine ») dont les principaux éléments syntaxiques sont très proches de ceux d'un diagramme Stateflow® :

| | SysML stm | Stateflow |
|---|---|---|
| **Etat** |  |  |
| **Transition** | evt [condition]/action<br><br>avec :<br>/action : action sur transition | evt [condition]{action}/action<br><br>avec :<br>{action} : action sur condition<br>/action : action sur transition |
| **Etat initial ou jonction par défaut** |  |  |

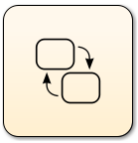
| | | |
|---|---|---|
| **Etat final** |  | inexistant |
| **Point de décision** |  |  |
| **Pseudo-état historique** |  |  |
| **Pseudo-état historique Profond** |  | inexistant |
| **Etat composite** |  |  |
| **Etat composite masqué** |  |  |

| | | | | |
|---|---|---|---|---|
| **Points de connexion** |  entrée |  sortie |  entrée |  sortie |
| **Etat orthogonal** |   |   | | |
| **Barre de divergence** |  | inexistant | | |
| **Barre de convergence** |  | inexistant | | |

Dans le cadre de mon enseignement, je propose aux élèves un cours inversé pour lequel les ressources sont disponibles sur ma chaîne YouTube.

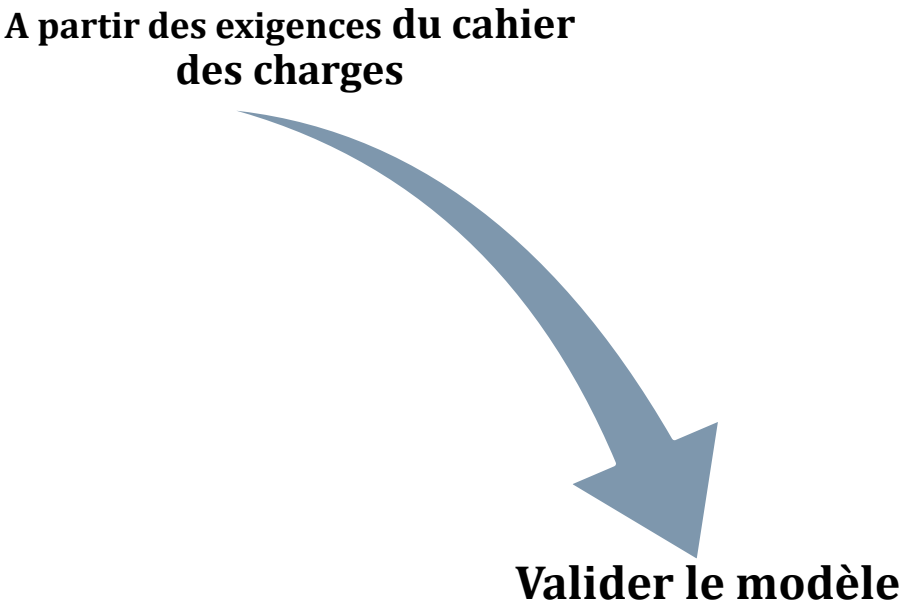
La capsule est composée de trois courtes vidéos présentant les concepts de base du diagramme d'état SysML :


Les applications proposées aux élèves pour s'approprier le cours de cette séquence sont pour la plupart simulées avec la suite Matlab® Simulink®Stateflow®.



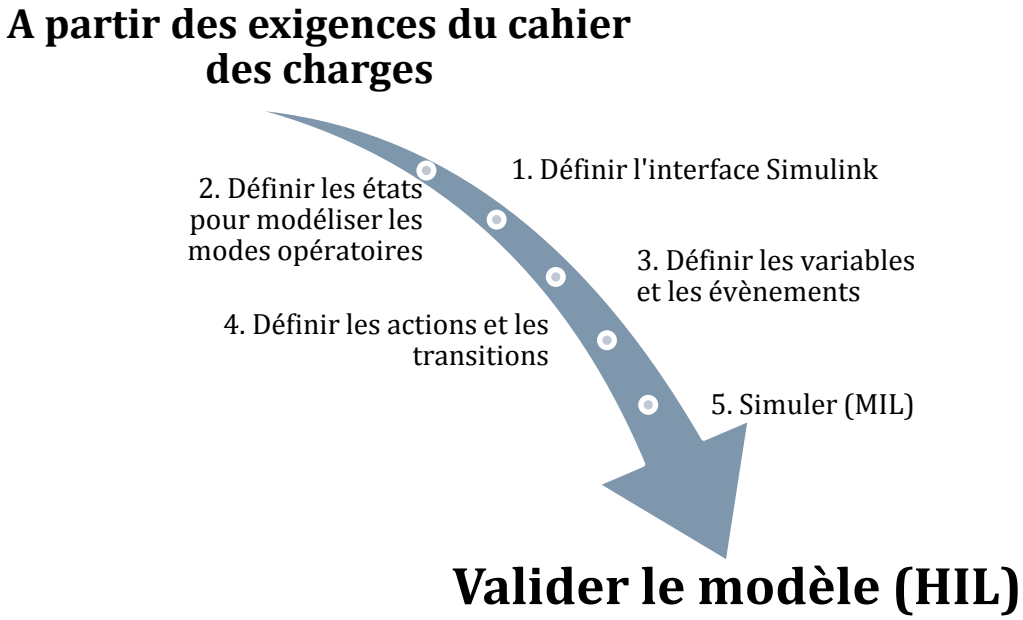
# Chapitre 3

# Proposition de méthodologie



# Chapitre 3 – Proposition de méthodologie …

Dans ce document les différents modèles ont été construits en mettant en œuvre la démarche qui suit :




Le diagramme des exigences donne les éléments du cahier des charges à respecter.

La première étape consiste à créer l'interface Simulink® permettant de simuler le modèle envisagé en insérant des blocs tels que des « **Sources** »pour générer les signaux d'entrée, des « **Sinks** » pour visualiser les résultats, des « **Switches** » pour commander la machine à état etc…

La définition des états caractéristiques des modes opératoires est donnée au cours de la seconde étape du processus de modélisation.

Les variables d'entrée, de sortie ou encore locales sont créées au cours de la troisième étape.

La quatrième étape consiste à mettre en place les actions et les transitions entre les états préalablement envisagés.

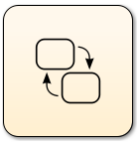
Ensuite, il est nécessaire de choisir un solveur en phase avec le modèle que l'on souhaite simuler. Est-ce un modèle continu, discret ou hybride ? C'est l'objet de la cinquième et avant dernière étape.

Enfin, la simulation peut avoir lieu. Au cours de cette dernière étape il sera bien souvent nécessaire de « debugger » la machine à état et de reprendre une partie des étapes précédentes.

Quand le modèle est validé, il peut être intéressant de retravailler sa forme en insérant des images par le biais de masques, de colorer certains éléments ou encore ajouter des commentaires pour une meilleure lisibilité.

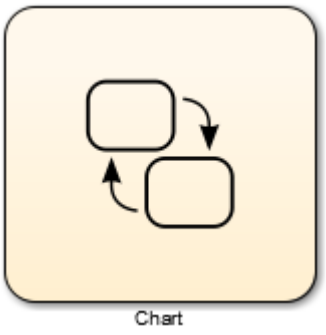
Aussi le programme écrit avec cet outil graphique peut être chargé dans une cible après avoir été compilé. Ce qui permet de valider avec un prototype les fonctions définies dans le cahier des charges.

Une introduction à l'approche **Model Based Design** peut avantageusement enrichir cette démarche en travaillant, dans un premier temps, dans un environnement « **Model In the Loop (MIL)** » pour répondre aux exigences du cahier des charges puis, dans un second temps, dans un environnement « **Harware In the Loop (HIL) »** en validant le comportement d'une cible réelle.

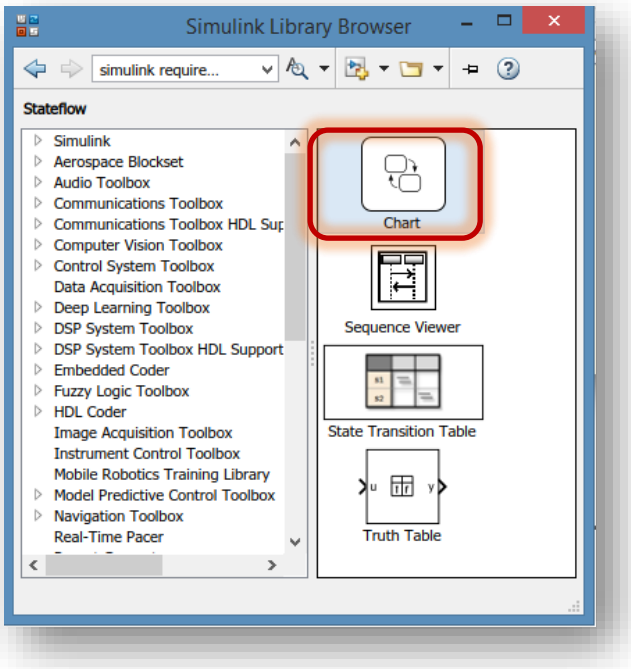


# Chapitre 4

# *Le « Chart »*

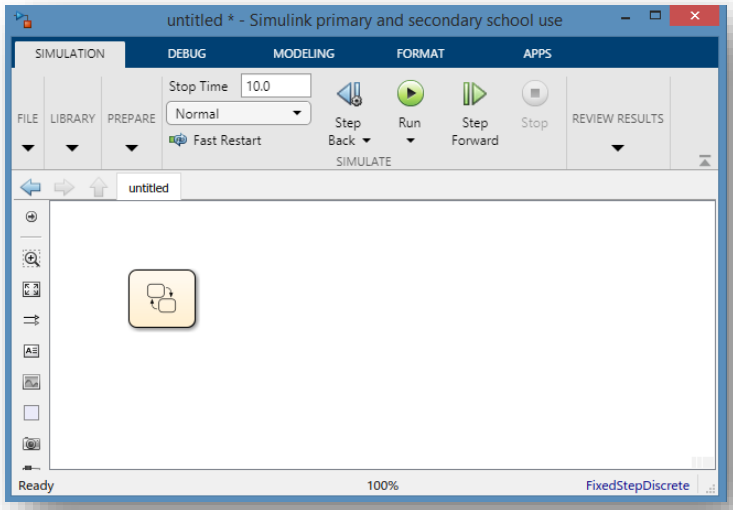


# Chapitre 4 - Le « Chart »…

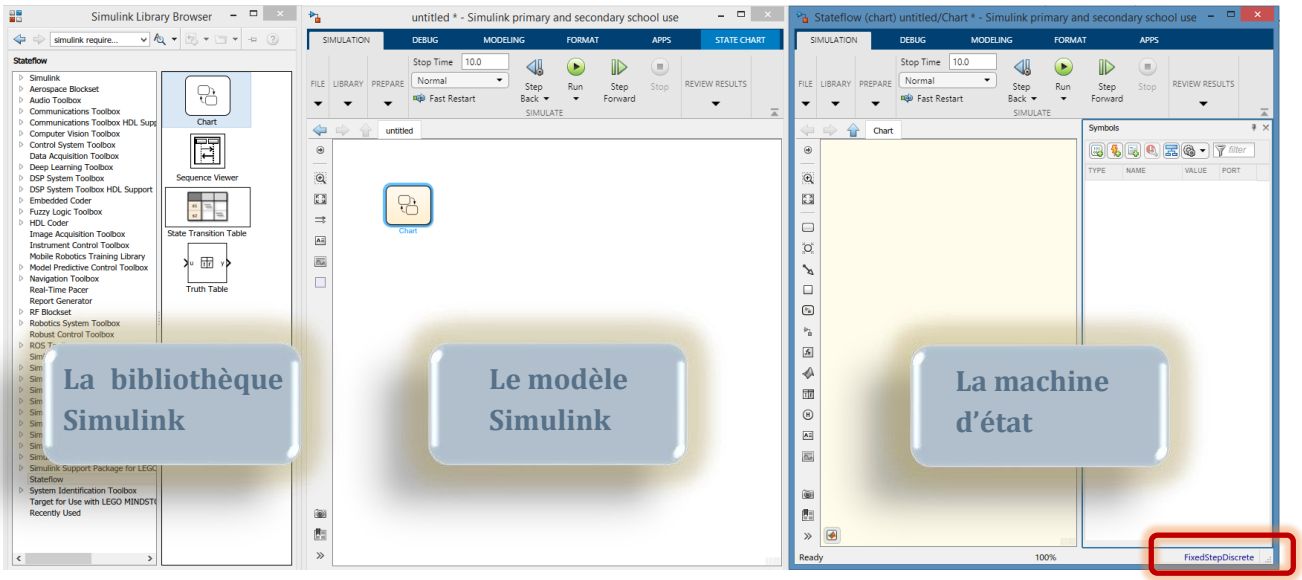



Pour construire une machine d'état, il faut lancer Matlab® puis la bibliothèque Simulink®. Ouvrir Stateflow® et glisser un « **Chart** » (ou diagramme) dans la fenêtre de construction du modèle. Un double-clic sur le bloc permet d'accéder à son contenu c'est-à-dire au diagramme état-transition.

On pourra judicieusement ouvrir le contenu du bloc « **Chart** » dans une nouvelle fenêtre en cliquant bouton droit puis « **Open In New Window** ». Cette disposition sera intéressante au moment de la simulation comme nous le verrons un peu plus tard.

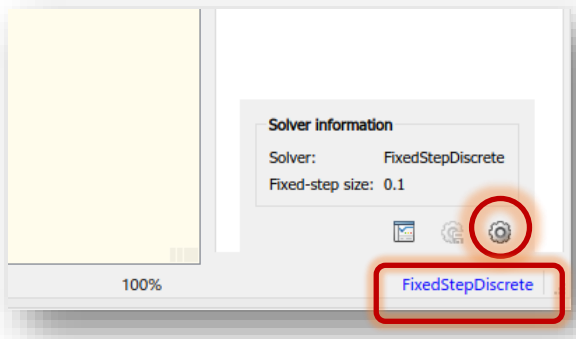



Votre écran affichera alors les trois fenêtres suivantes :

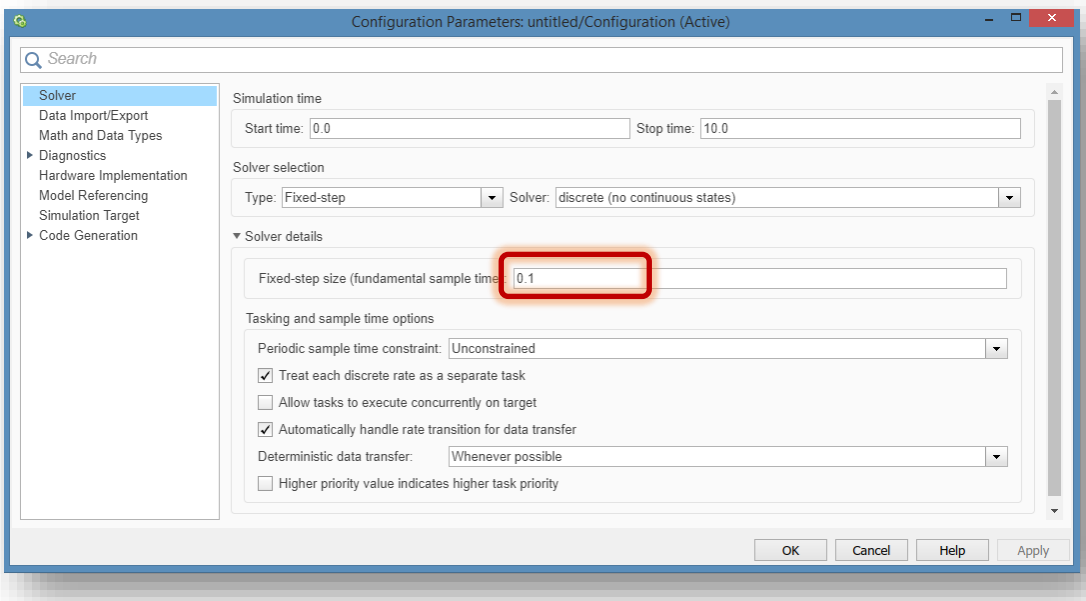



Le solveur est automatiquement choisi : « **discrete (no continuous states)** »

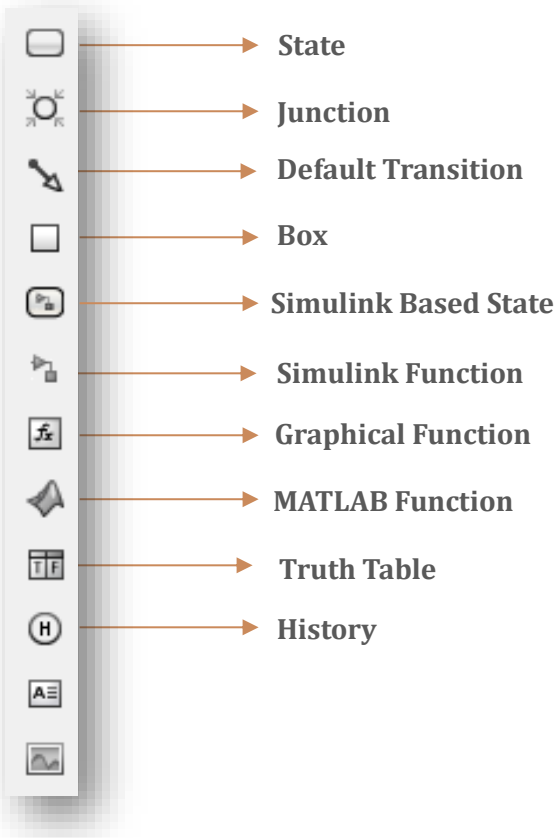
En cliquant sur le lien en bas à droite de l'écran, une fenêtre s'ouvre, donnant quelques informations :

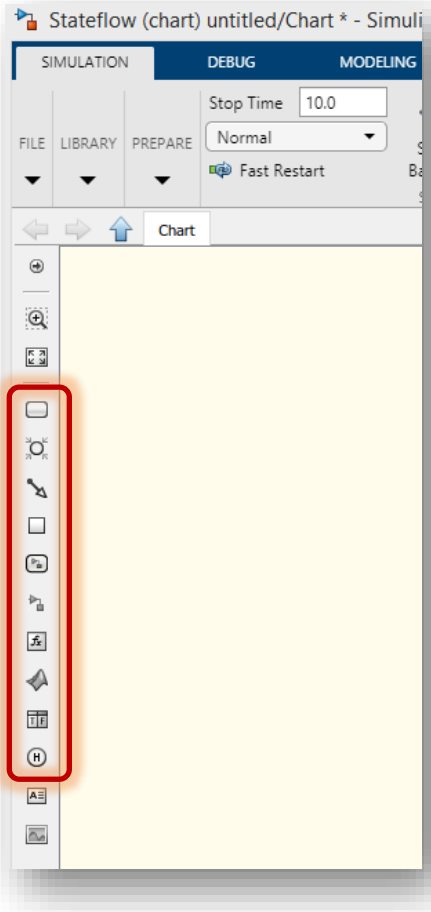



Si la valeur du pas de calcul, fixe, « **fixed step size** » ne convient pas il est possible de la changer en cliquant alors sur l'icône 



La palette d'outils nécessaire à la construction d'un diagramme état-transition par un « glissé/déposé » est la suivante. Elle comporte plusieurs objets dont certains sont graphiques:




## *A propos du fonctionnement de la machine d'état…*

Le comportement dynamique de la machine d'état est par défaut synchronisé avec l'horloge qui définit le temps d'échantillonnage choisi pour la simulation du modèle (« **sample time** »), c'est-à-dire le temps correspondant à un pas de calcul du solveur. A chaque pas, un nouvel évènement noté « **tick** » survient, il y a alors possibilité d'évolution de la machine vers un nouvel état.

La machine peut aussi être soumise à des évènements extérieurs autorisant le passage à un nouvel état. Il est important de bien distinguer ces modes d'évolution de la machine d'état.

Détaillons maintenant le contenu de la palette permettant la construction de machines d'état.

## 4.1- STATE…

Cette icône, un rectangle aux coins arrondis, représente un état. L'activité ou la non-activité d'un état dépend de l'occurrence d'un évènement et/ou d'une condition de garde.

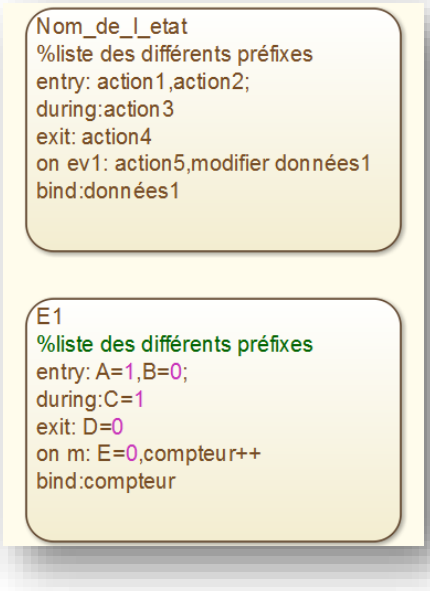
### 4.1.1 - « Label » et mots clé d'un état


Le nom de l'état ou son étiquette (« **state label** ») est saisi dans le champ « **Name** » en haut à gauche du rectangle. Les actions sont saisies en cliquant sur « {…} » au-dessous du nom de l'état.

Les actions associées aux états peuvent avoir lieu :

- A l'activation de l'état : « **entry : action** ; » ou « **en : action ;** »
- A la désactivation de l'état : « **exit : action ;** » ou « **ex : action ;** »
- Pendant l'activité de l'état : « **during : action ;** » ou « **du : action ;** »

« **entry** », « **during** », « **exit** » sont appelés préfixe (prefix) ou mot-clé (keyword). Ils peuvent s'écrire de manière plus synthétique : « **en** » pour « **entry** », « **du** » pour « **during** », « **ex** » pour « **exit** ».




Si plusieurs actions sont associées à un mot-clé, elles seront séparées par un point-virgule ou une virgule.

A la liste précédente nous pouvons ajouter d'autres mot-clés : « **on** » et « **bind** ».

En effet nous avons la possibilité de réaliser des actions à l'occurrence d'évènements interne lorqu'un état est activé. On utilisera alors le préfixe « **on** » pour chacun des évènements internes déclencheurs : « **on *évènement* : action ;** »

Le préfixe « **bind** » signifie que les données qui suivent le mot-clé sont « liées » à l'état comportant ce préfixe. Dans ce cas, seul cet état ou des états enfants peuvent modifier les données précisées.

Par contre, ces données peuvent être utilisées par d'autres états de la machine.

En absence de mot-clé, une action est réalisée en entrant (« **entry** ») dans l'état.

Afin de préciser certains éléments, il est aussi possible de saisir un commentaire dans un état en le précédant d'un % comme le montre la figure précédente.

**Un premier petit exercice…**

Stateflow® ouvert, vous allez créer un premier « **State Chart** ».

Saisir l'état E1 précédent pour vous familiariser avec la syntaxe.

Les couleurs des caractères qui apparaissent dans l'état sont générées automatiquement (vert pour les commentaires, jaune orangé pour les évènements, fushia pour les valeurs numériques). Cela permet de détecter rapidement une faute de frappe ou de syntaxe.

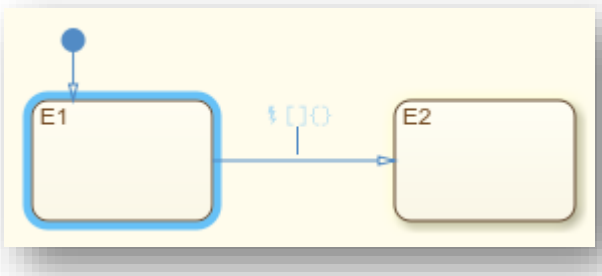
Avant de construire des machines plus sophistiquées, il est souhaitable de lire ce qui suit à propos des transitions, des opérateurs temporels, des décompositions exclusive et parallèle.

Cette synthèse permet de comprendre les principes de base du comportement d'une machine d'état construite avec Stateflow®.

## *A propos des transitions et des évènements…*

Le passage d'un état à l'autre se fait par une transition qui se matérialise par une liaison orientée entre un état source et un état pointé. Nous parlerons dans ce cas de transition externe.

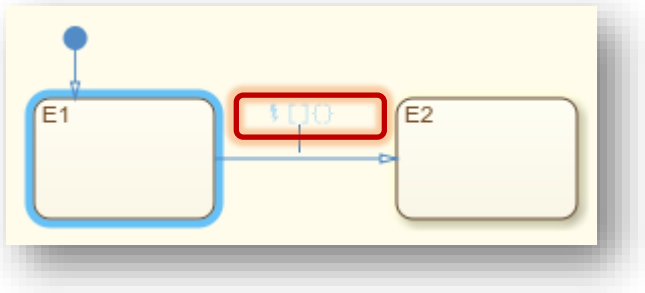
Le tracé de la transition se fait naturellement en amenant le curseur de la souris sur le contour de l'état source et en cliquant avec le bouton gauche de la souris jusqu'au contour de l'état pointé.



### 4.1.2 - « Label » d'une transition

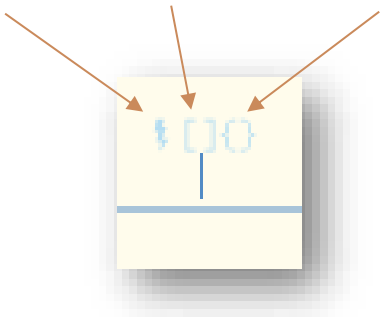
La transition est caractérisée par une étiquette ou **« transition label »** (saisie dans la zone contenant le point d'interrogation) qui décrit les circonstances ou les conditions de passage d'un état à un autre.

L'étiquette peut contenir un évènement et/ou une condition et/ou une action sur condition et/ou une action sur transition.




Le format général de l'étiquette d'une transition s'écrit alors:

**Evènement [condition] {action sur condition} / action sur transition**



Ce qui s'interprète de la façon suivante :

- A l'occurrence de l'évènement la transition est évaluée.
- Si la condition est vraie alors la transition est valide et si l'action sur condition existe, elle est executée.
- L'action sur transition, si elle existe, est aussi réalisée. Si la transition est constituée de plusieurs segments, l'action sur transition sera exécutée seulement lorsque la transition toute entière aura été franchie.

Si l'évènement n'est pas précisé dans l'étiquette de la transition, la condition qui suit, si elle existe, sera évaluée à chaque pas de calcul.

Si un évènement extérieur est considéré, il faudra le définir. Il s'affichera alors sur le dessus du « **Chart** » préalablement créé, comme nous le verrons plus loin.

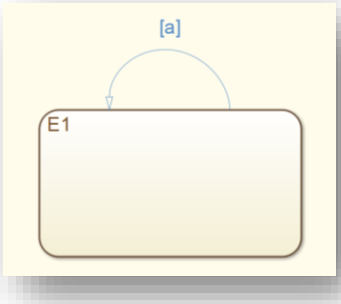
Pour expliciter une transition, il est possible d'ajouter un commentaire. La syntaxe serait par exemple la suivante par :

**[condition] {action sur condition}/*commentaire*/**

### 4.1.3 - Cas particuliers de la transition réflexive et de la transition interne

La transition réflexive (« **self loop transition** ») part d'un état ou d'un pseudo-état et y revient. L'état source et l'état pointé est donc le même.

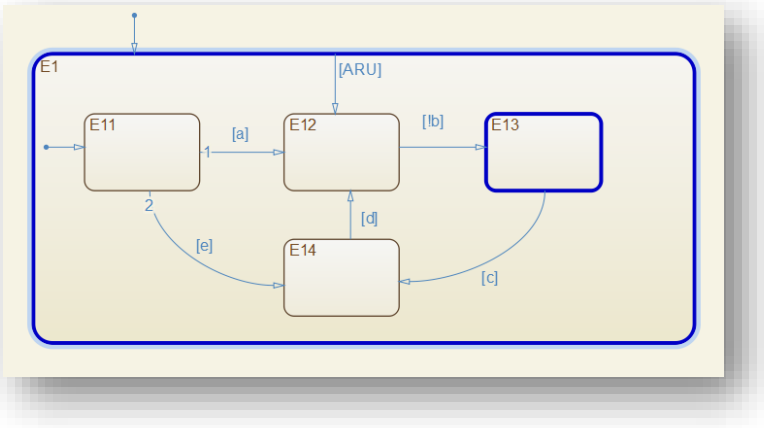
Quand la condition [a] est vraie alors que l'état E1 est actif, il est désactivé puis aussitôt réactivé au pas de calcul suivant.




La transition interne (« **inner transition** ») ne sort pas de son état source. Elle peut pointer un état enfant ou bien un pseudo-état.

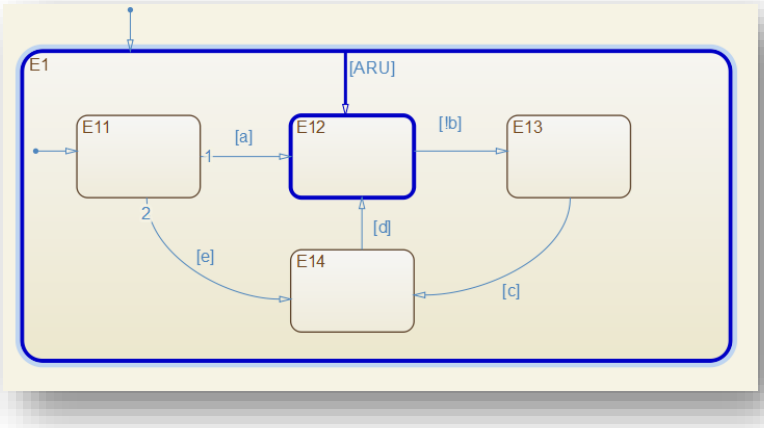
Prenons l'exemple du forçage dans un état particulier :

Dans le cas qui nous intéresse ici, l'état parent E1 et l'état enfant E13 sont actifs.




A chaque pas de calul, les transitions associées à l état actif de plus haut niveau hiérarchiquue sont d'abord évaluées. Donc ici la transition entre E1 et E12 est évaluée à chaque pas de calcul.

La condition [ARU] devient vraie. E13 est alors désactivé et E12 devient actif : c'est le forçage dans un état donné. On force l'activation de E12 lorsque la condition [ARU] est vraie.

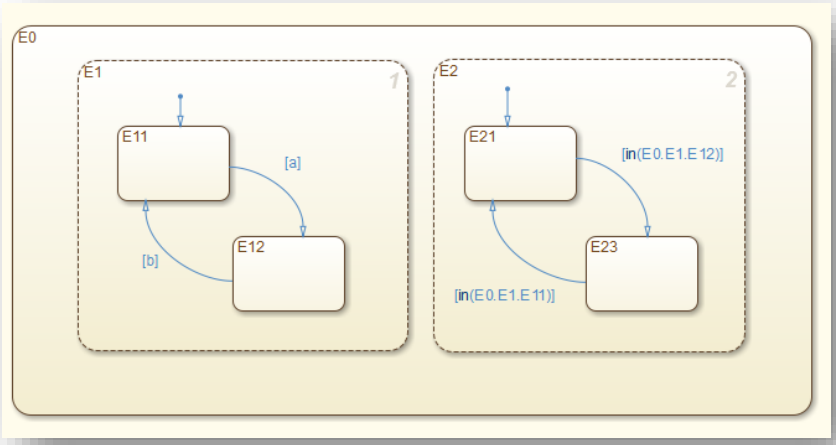



La transition interne peut pointer une jonction « **History** » (voir le paragraphe consacré à la jonction « **History Junction** »).

Enfin l'utilisation de cette transition peut aussi être intéressante pour simplifier la structure d'un état composite ( voir le paragraphe consacré à la jonction : « **Junction** »)

### 4.1.4 - Prise en compte de l'activité d'un état dans une transition

La prise en compte dans une transition de l'activité d'un état est possible si la variable « **in(état à préciser avec son niveau hiérarchique)** » est vraie, par exemple :




Ici le passage de l'état E21 à E23 se fait si l'état E12 est actif.

Le retour à l'état E21 a lieu si l'état E11 est actif.

### 4.1.5 - Les évènements extérieurs

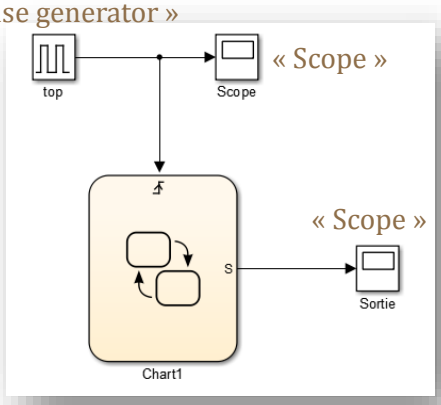
**Un peu de pratique…**

Nous allons construire une machine à état comportant deux états E1 et E2. Le passage de l'un à l'autre se fera à l'occurrence d'un évènement extérieur noté « **top** ».

L'évènement pourra être un front montant (« **rising** »), un front descendant (« **falling** ») ou encore un front montant ou descendant (« **either** »).

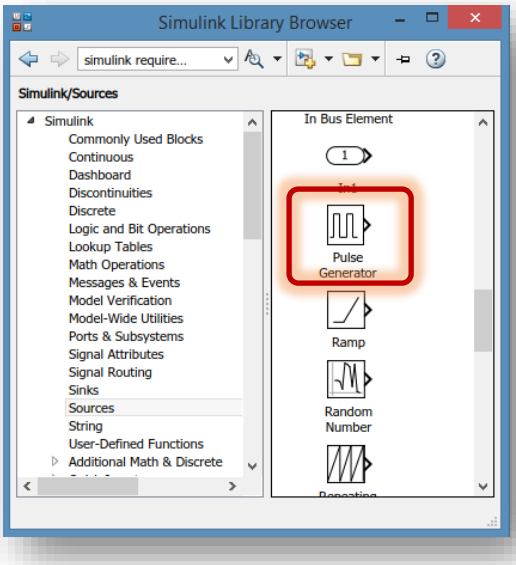
Dans Simulink® construisons l'exemple ci-contre :

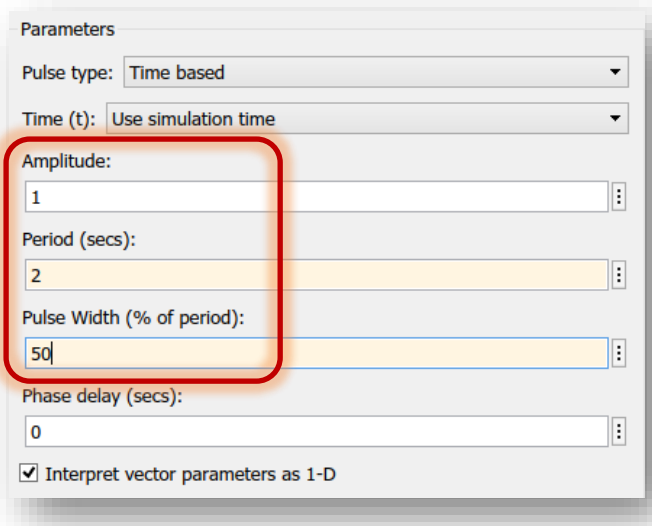
« Pulse generator »




« State chart »

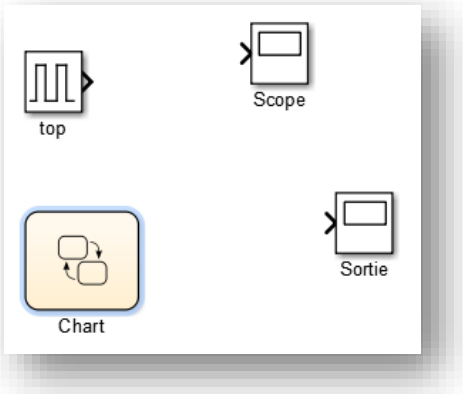
Dans le menu « **Sources** » de Simulink®, sélectionner le bloc « **Pulse Generator** », choisir un signal carré d'amplitude 1, d'une période de 1s et de largeur d'impulsion 50% de la période :




Les « **Scopes** » sont extraits de la bibliothèque « **Sinks** » de Simulink® et permettent de visualiser les résultats de la simulation.

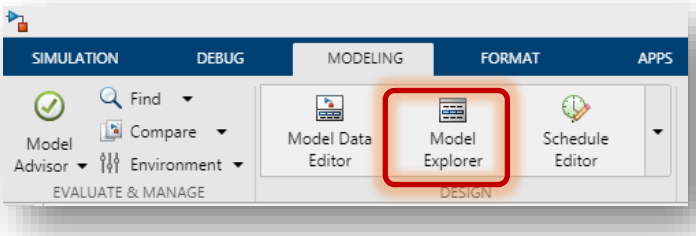
Pour terminer, glisser un « **State Chart** » à partir de Stateflow®.

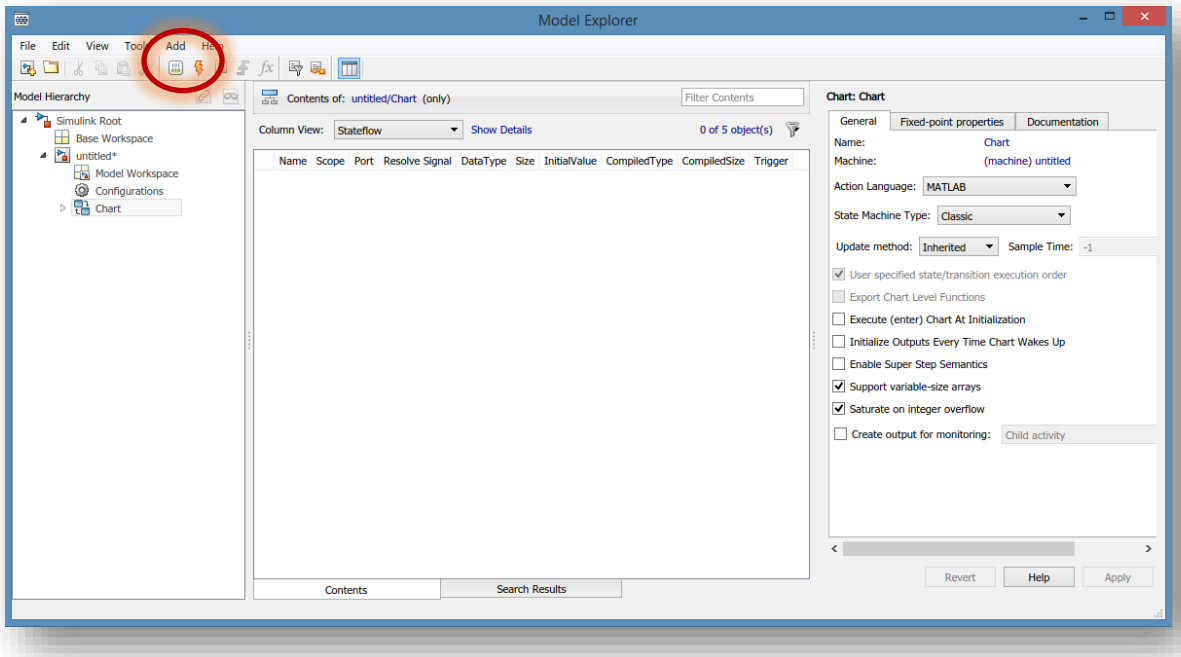



eIl faut maintenant mettre en place les entrées et sorties du « **Chart** ». On choisit d'appeler la sortie « **S** » et l'évènement extérieur « **top** ». L'occurrence de l'évènement « **top** » se fait sur chaque front montant du signal carré « **top** ».

Sélectionner le « **Chart** », afficher le menu contextuel avec le bouton droit de la souris puis sélectionner « **Explore** ». Une fenêtre s'ouvre, il s'agit de l'explorateur de modèle (« **Model Explorer** »).

Une autre possibilité pour ouvrir l'explorateur et de cliquer sur l'onglet « **MODELING** » de Simulink® dans le bandeau supérieur et de sélectionner « **Model Explorer** » dans le menu « **DESIGN** ».

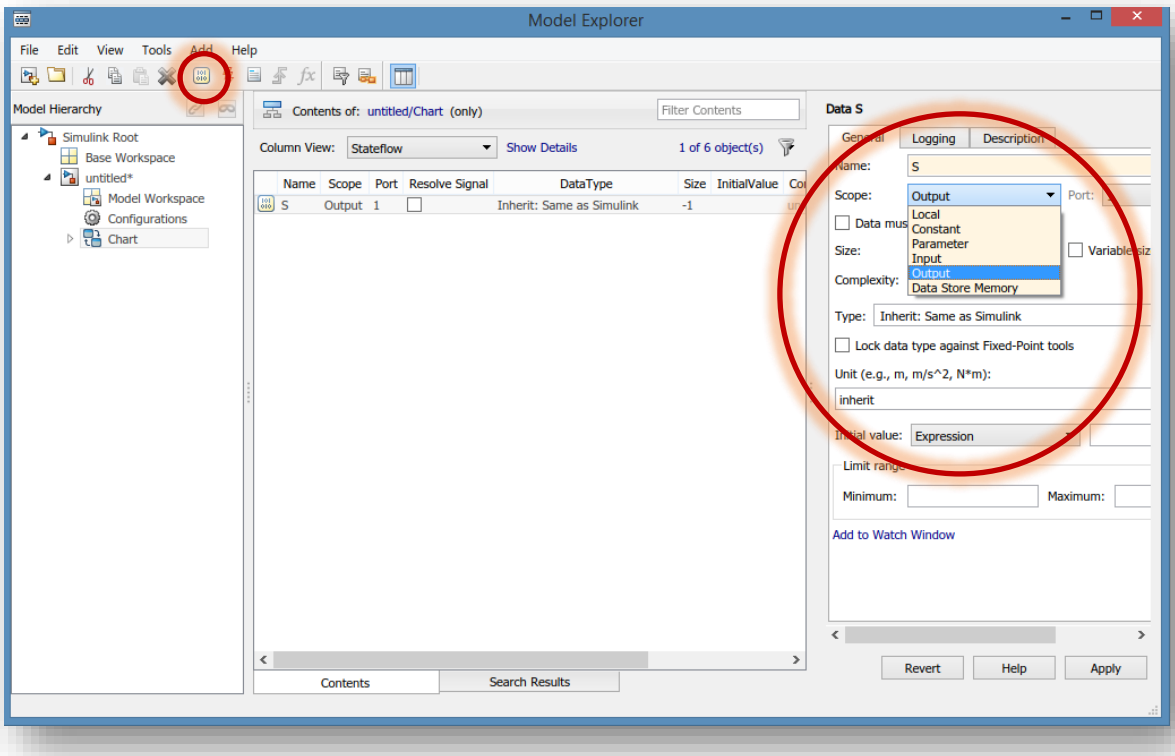




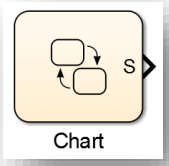

L'explorateur de modèle va permettre la définition des entrées, des sorties, des évènements. Nous utiliserons les deux icônes du bandeau supérieur suivantes :

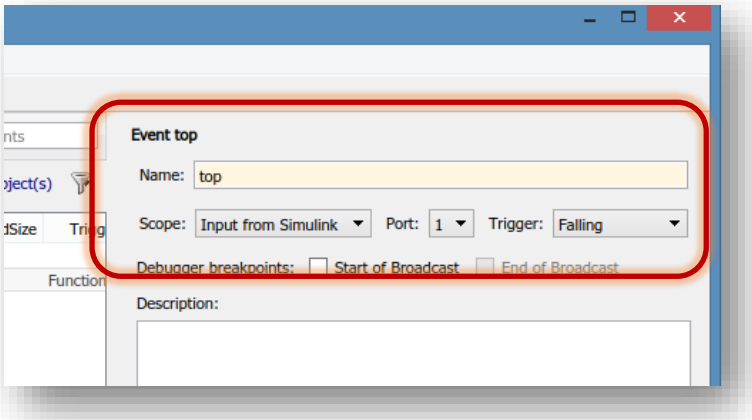
Permet l'ajout d'une donnée « **data** » (entrée,

Permet l'ajout d'un évènement « **event** »



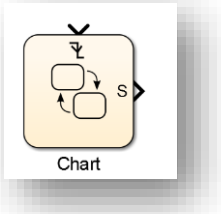
Nous pouvons vérifier que la sortie « **S** » est désormais présente sur le côté droit du « **Chart** ».



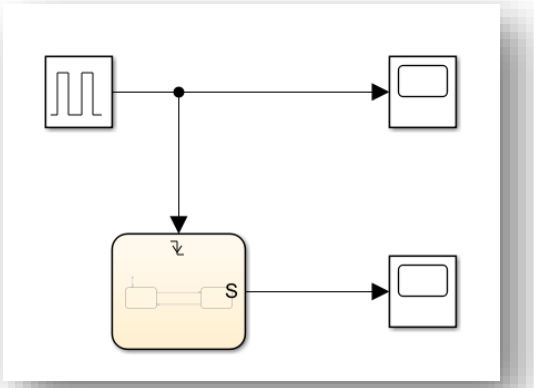

De la même manière, on définit l'évènement « **top** ».

Le signal dont on souhaite extraire l'évènement « **top** » pointe le « **Chart** » sur le dessus. Il correspond à une entrée depuis Simulink® (« **Input from Simulink** »). On choisit le front descendant (« **Trigger : Falling** ») du signal pour réveiller la machine.

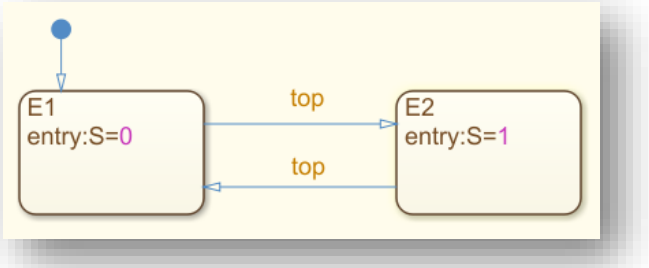



Nous pouvons vérifier la création d'un port de déclenchement (« **Trigger** ») sur le dessus du « **Chart** ». La nature de l'évènement est symbolisée ici par un front descendant.

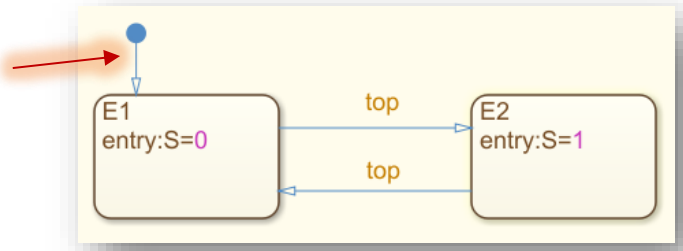
Reste à relier les différents blocs :




Un double clic permet d'ouvrir le « **Chart** ». Tracer par exemple le diagramme suivant :

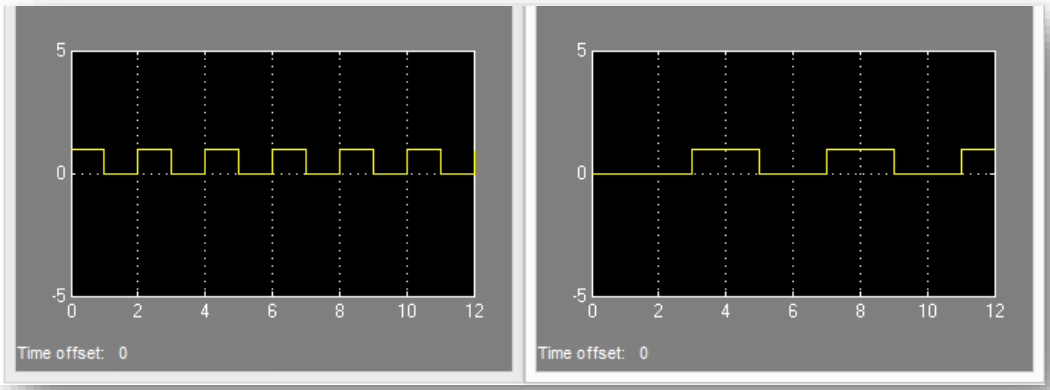


Cette transition est une transition par défaut qui permet de sélectionner l'état qui doit être actif lors du réveil de la machine (voir paragraphe suivant).




Lancer la simulation en cliquant sur « Run » dans le bandeau supérieur. Elle donne les les chronogrammes suivants (double clic sur les « **Scopes** »):

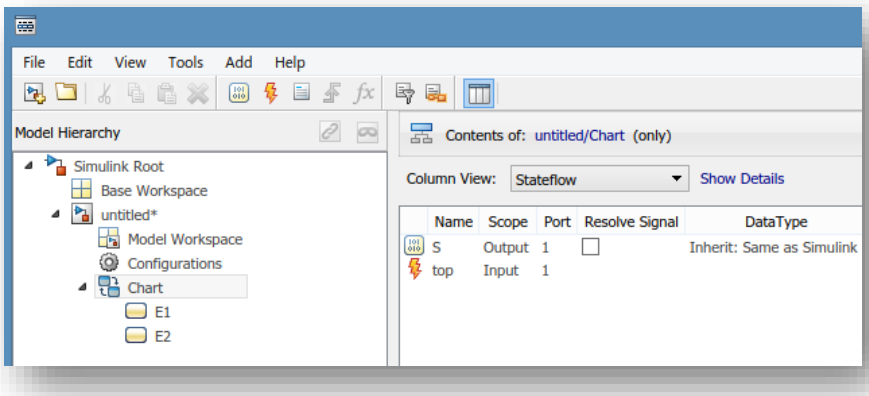
A t=1 s, le premier front descendant réveille la machine et S=0.



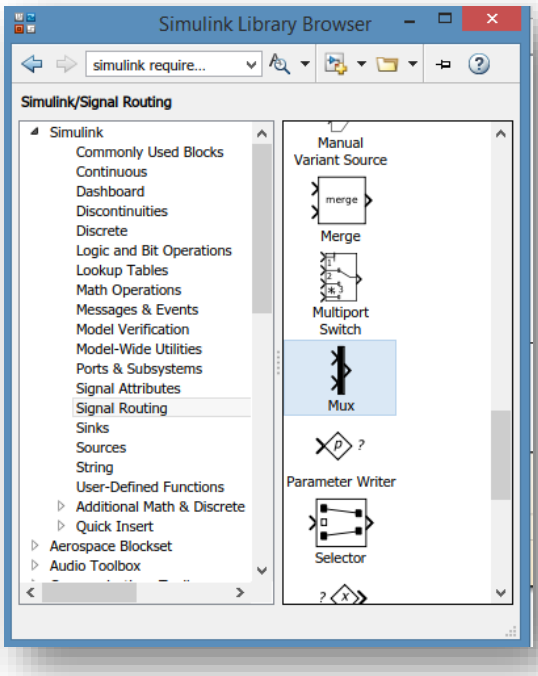

A t=2 s, le second front descendant autorise le changement d'état et S=1. Ainsi de suite…

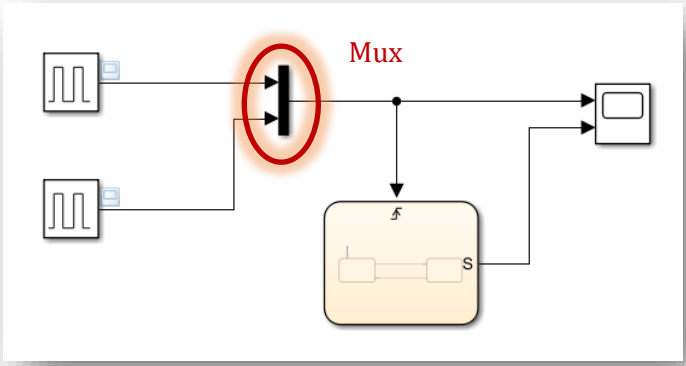
**Remarque :** dans cet exemple les états sont activés à l'occurrence du front descendant de « top ». L'action associée se fait en entrant dans l'état. Ici nous avons utilisé le mot-clé « entry ».

Un retour dans l'explorateur de modèle permet de visualiser l'arborescence du modèle construit ainsi que le contenu du « **Chart** » :

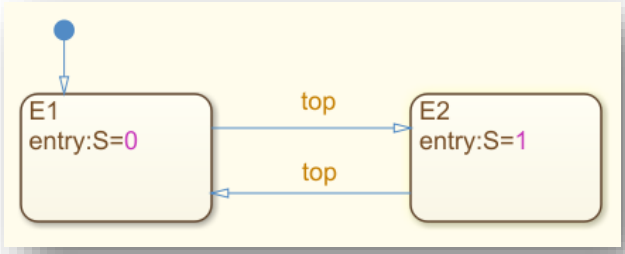


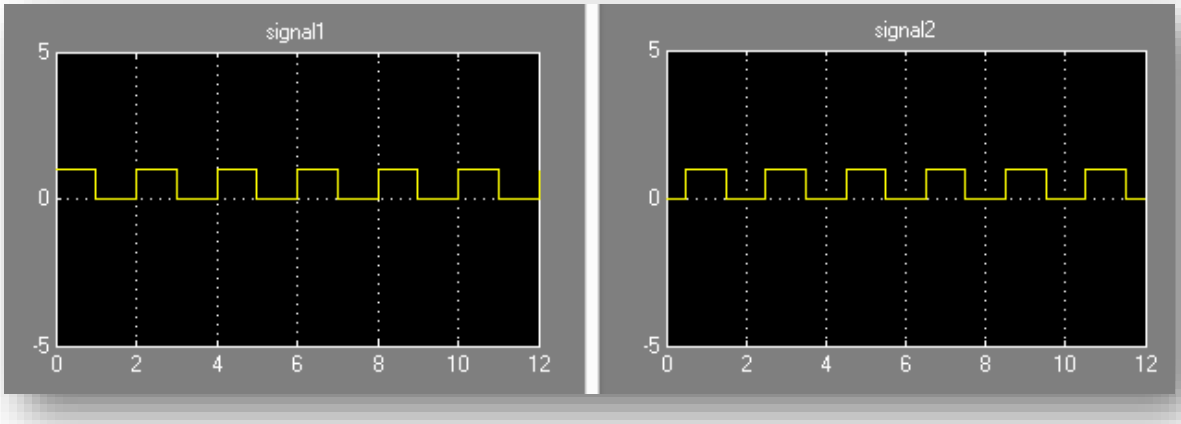
Il est tout à fait possible de combiner deux signaux dans Simulink® et générer les évènements qui en seraient issus. Il faut alors utiliser le bloc « **Mux** » qui se trouve dans la bibliothèque « **Signal Routing** » de Simulink® :




Dans le cas qui nous intéresse nous considérons que les évènements « **clic** » et « **clac** » correspondent aux fronts montants des deux signaux obtenus grâce aux deux générateurs de signaux

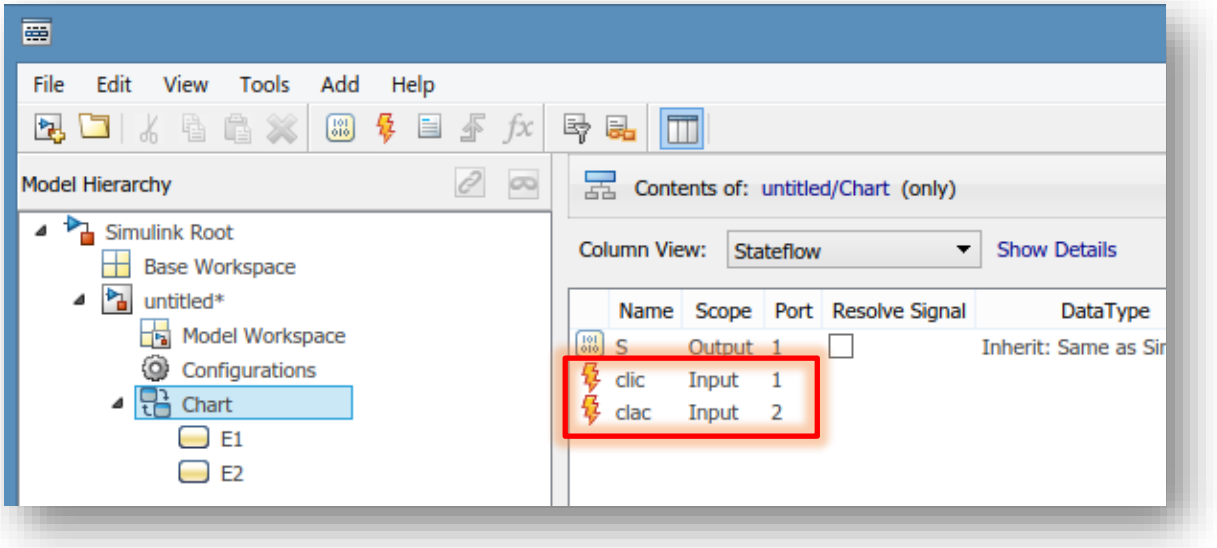


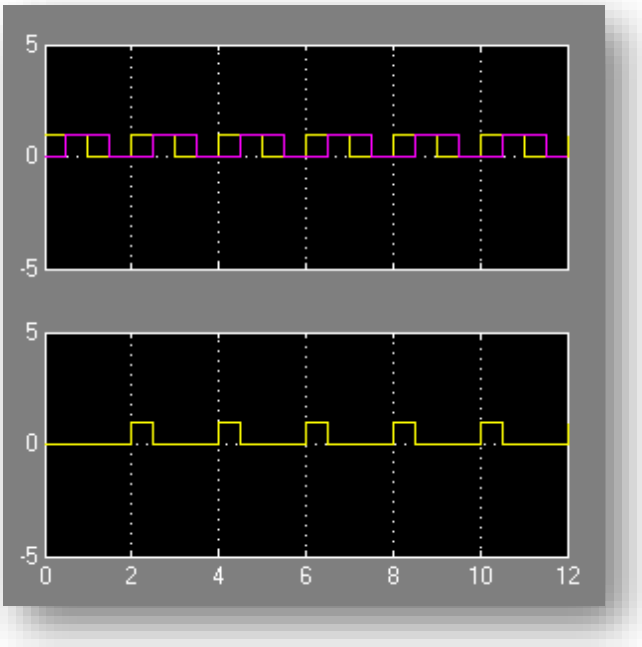

(« **Pulse Generator** »). Les signaux sont déphasés d'un quart de période.

Ensuite, il faut affecter un port du bloc « **Mux** » aux évènements que l'on veut prendre en compte. Les ports sont numérotés du haut vers le bas de manière croissante.

Ici, « **clic** » est affecté au port 1 et « **clac** » au port 2 :




Les résultats de la simulation donnent les chronogrammes suivants :




Le premier « **Scope** » donne le signal multiplexé en sortie du bloc « **Mux** ». Le second donne la sortie du « **Chart** ».

Le signal jaune est lié au port 1 du bloc « **Mux** » et correspond donc à « **signal clic** ».

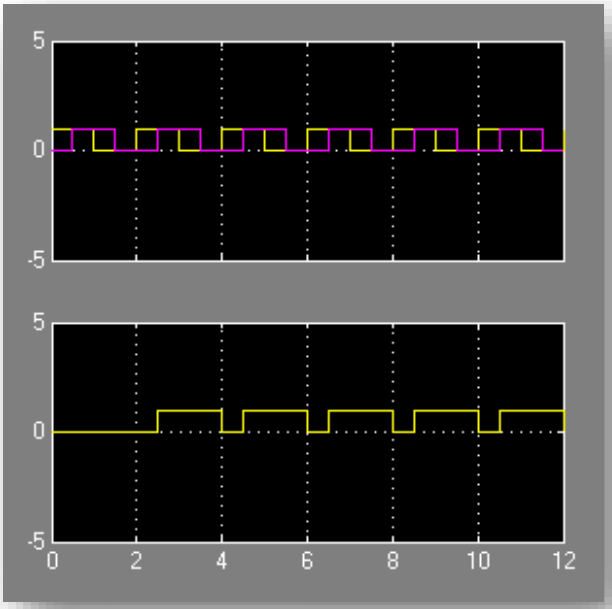
Le signal violet, déphasé par rapport au signal précédent, est lié au port 2 du bloc « **Mux** » et correspond donc à « **signal clac** ».

Le « **Chart** » est réveillé sur le premier évènement, c'est-à-dire à l'occurence du premier front montant. Cet évènement a lieu à t = 0.5 s, il s'agit de l'évènement « **clac** ». A cet instant E1 est activé et S = 0.

Le passage de E1 à E2 se fera à l'occurrence de « **clic** » (front montant du « **signal clic** ») soit à t = 2s, dans ce cas S = 1 jusqu'à l'occurrence de « **clac** » (front montant du « **signal clac** ») et ainsi de suite.

Pour montrer l'importance de la connexion des signaux au bloc « **Mux** », nous pouvons intervertir l'affectation des ports : « **signal clic** » sur le port 2 et « **signal clac** » sur le port1.

Nous obtenons le résultat ci-contre.




Le « **Chart** » est réveillé sur le premier évènement, c'est-à-dire à l'occurence du premier front montant. Cet évènement a lieu à t = 0.5 s, il s'agit de l'évènement « **clic** ».

A cet instant E1 est activé et S = 0. Le passage de E1 à E2 se fera à l'occurrence de « **clic** » (front montant du « **signal clic** ») soit à t = 2.5 s, dans ce cas S = 1 jusqu'à l'occurrence de « **clac** » (front montant du « **signal clac** ») à t = 4s et ainsi de suite.

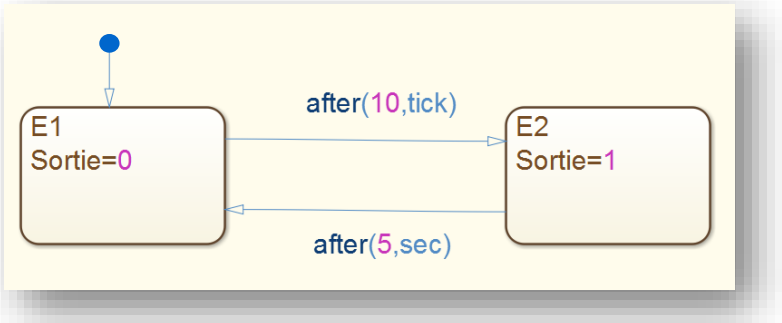
## *A propos des opérateurs temporels…*

L'étiquette d'une transition peut aussi contenir un opérateur temporel : « **after** », « **before** », « **at** », « **every** », « **temporalCount** ».

**Remarque préliminaire :**

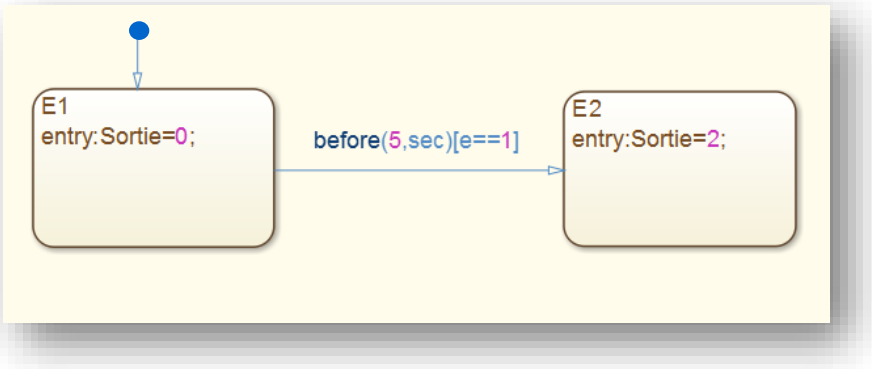
- le temps de simulation peut s'exprimer en seconde (sec), en milliseconde (msec), en microseconde (usec). Le temps de simulation est un réel positif. Il dépend du processeur et de ses capacités à mener les calculs liés au modèle. Il ne correspond pas au temps réel.
- Le temps d'échantillonnage (« sample time ») du solveur s'exprime en tick. Il correspond à un pas de calcul et il est ajusté au moment de la définition du solveur. Le nombre de ticks est donc un nombre entier.

## L'opérateur « after » :




Dans l'exemple ci-contre, le passage de l'état E1 à l'état E2 se fait après 10 pas de calcul (« **ticks** ») et le passage de l'état E2 à l'état E1 se fait lui après 5 secondes du temps de simulation. Une modification du « **Sample Time** » du solveur modifie la durée du passage de E1 à E2. En effet une augmentation du « **Sample Time** » augmente la durée d'un « **tick** » et inversement.

## L'opérateur « before » :



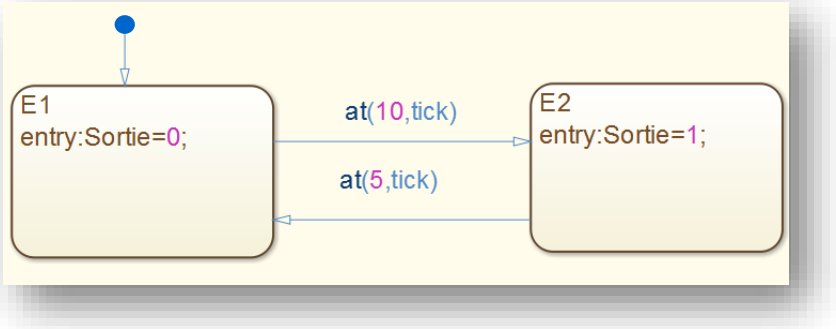

Ici, la transition sera franchie si la condition [e==1] est vraie avant que le temps de simulation atteigne 5 secondes.

Par conséquent, un opérateur temporel peut être associé à une condition. Il peut aussi être présent dans une expression logique.

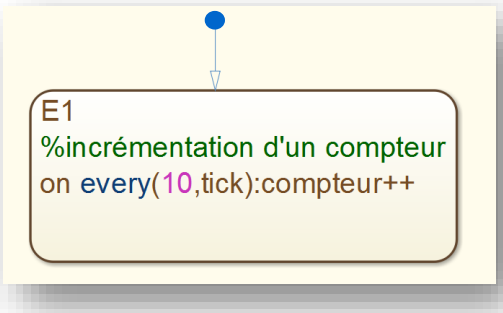
Par exemple : **[a && before(10,sec)] qui se lit « a ET before(10,sec) »**

## L'opérateur « at » :

L'opérateur temporel « **at** » définit la date à laquelle l'état E2 sera activé si E1 est activé et réciproquement. Les dates ne peuvent être exprimées qu'en « **tick** » ce sont donc des nombres entiers.



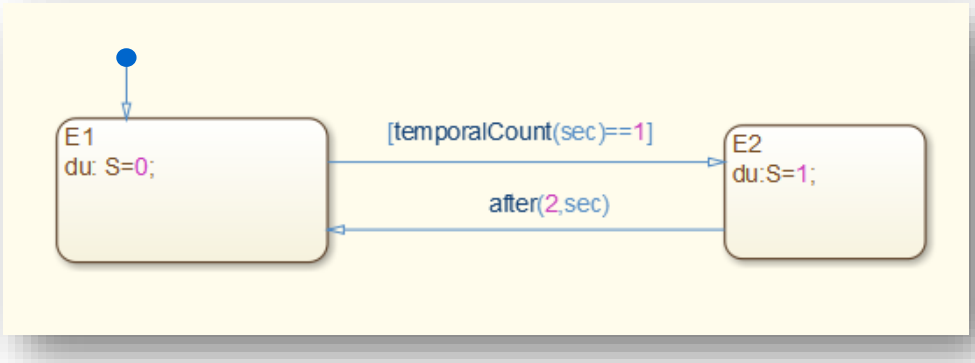
## L'opérateur « every » :




« **every(n,tick)** » est vrai au nième pas de calcul. Cet opérateur sera exprimé obligatoirement en nombre de « **ticks** ». Ici, on incrémente d'une unité un compteur (« **compteur++** ») tous les 10 pas de calcul tant que l'état E1 est actif.

## L'opérateur « temporalCount » :

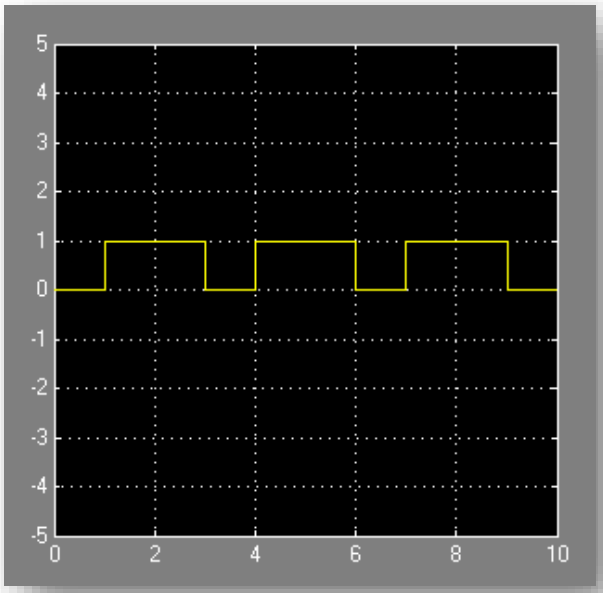
Pour évaluer la durée de l'activité d'un état, nous disposons de l'opérateur « **temporalCount** ». Le compteur est remis à 0 à chaque activation de l'état associé.




Dans l'exemple ci-dessus, quand E1 est activé un compteur décompte le temps d'activation de l'état. Ici, ce temps est mesuré en secondes.

Lorsque le compteur atteint 1s alors la condition [temporalCount(sec)==1] devient vraie, E1 est désactivé et E2 est activé.

Après 2s, E2 est désactivé et E1 réactivé. Ainsi de suite… Le scope associé à la sortie S confirme bien ce comportement.



## *A propos des super-états ou états composites …*

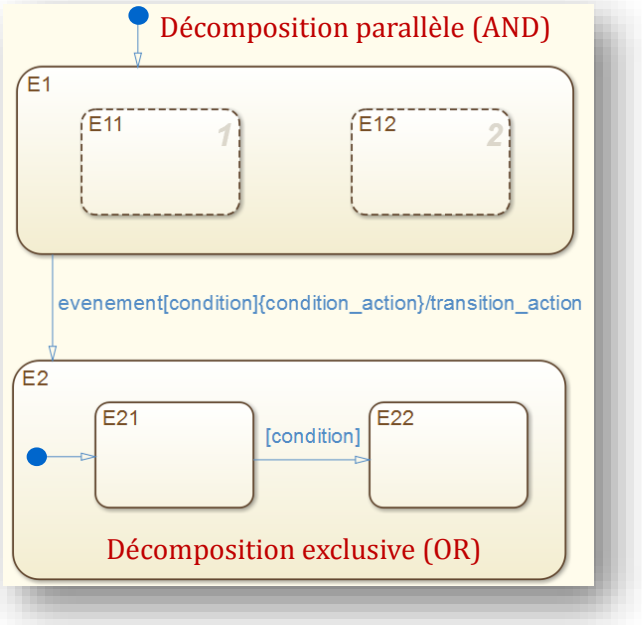
### Décomposition exclusive (OR) ou parallèle (AND) des états composites

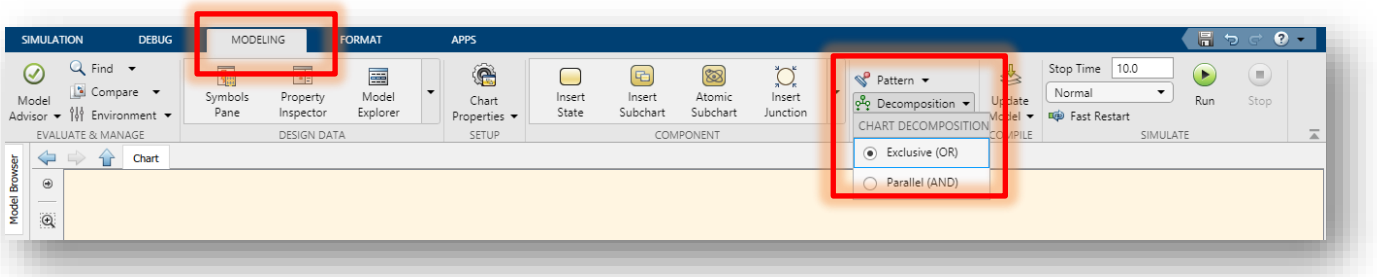
Dans la figure qui suit, le super-état E1 est un état parallèle. Lorsque E1 est activé alors E11 ET E12 sont actifs simultanément.

La présence d'une transition par défaut affectée à E11 ou E22 n'a pas lieu d'être du fait de la simultanéité de l'activation des états.

Ces sous-états présentent un contour en pointillés.

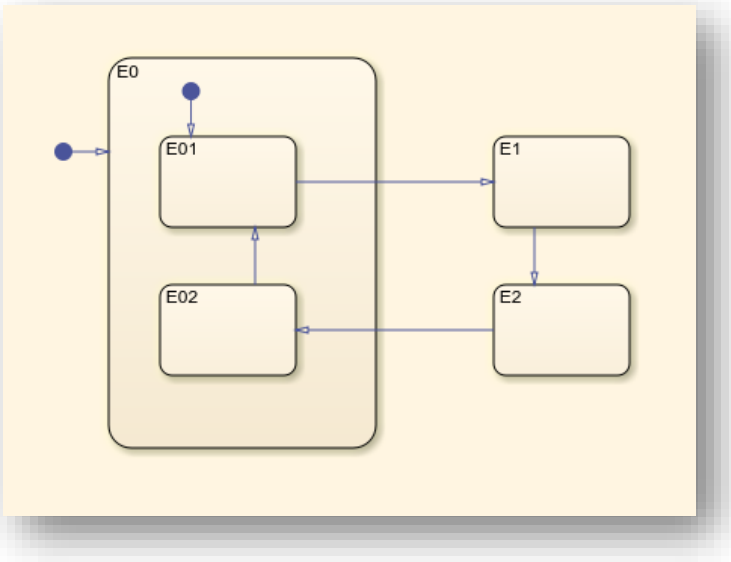
On accède à la décomposition en sélectionnant le super-état par un clic gauche, puis clic droit, « **Decomposition…** » ou à partir de l'onglet « **MODELING** » du bandeau supérieur : sélectionner le super-état par un clic, puis cliquer sur « **Decomposition** » du menu « **EDIT** » du bandeau supérieur. Sélectionner alors la décomposition souhaitée :




Les quelques notions de base concernant les états et les transitions exposées jusqu'ici sont bien sûr à connaître pour construire des machines d'état et pour en comprendre la dynamique.

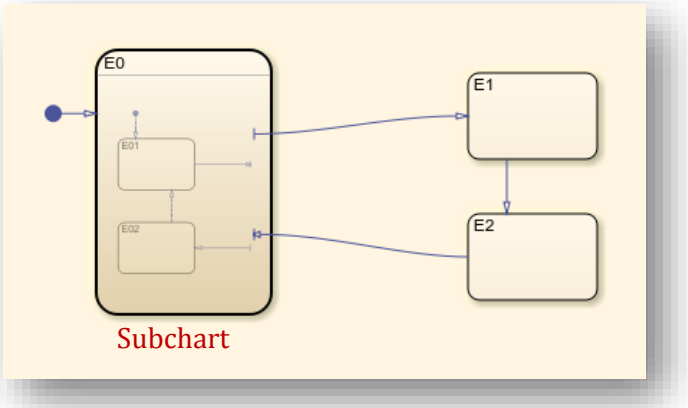
## Utilisation d'un « Subchart »

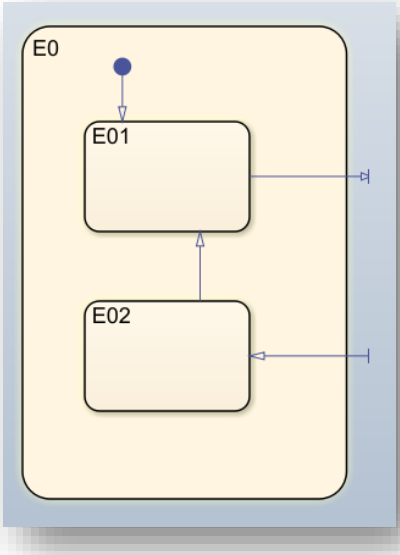



Si un super-état ou état composite contient de nombreux états il est possible de réduire le diagramme en masquant le contenu de l'état composite. On transforme le surper-état en « **Subchart** » :

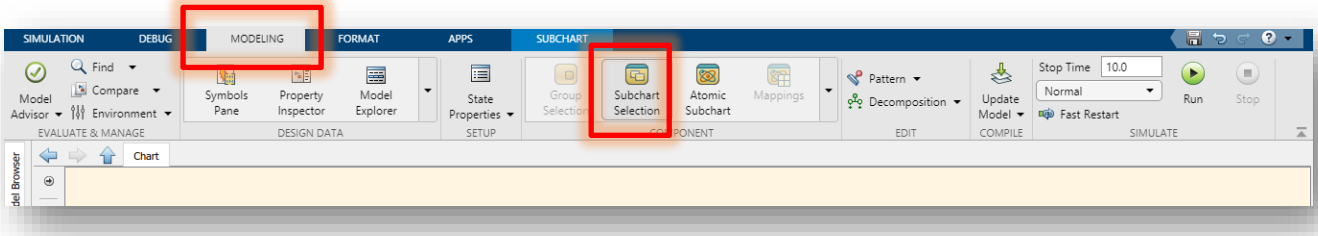
Afficher le menu contextuel par un clic droit, puis sélectionner «**Subchart & Group** » puis « **Subchart** ».

Un double-clic sur le « **Subchart** » ainsi créé permet d'afficher l'état composite dont le contenu a été mis en transparence.




Une autre possibilité pour la création d'un « Subchart » est de sélectionner le super-état, dans l'onglet « **MODELING** » cliquer sur le bouton « **Subchart Selection** » du menu « **COMPONENT** »



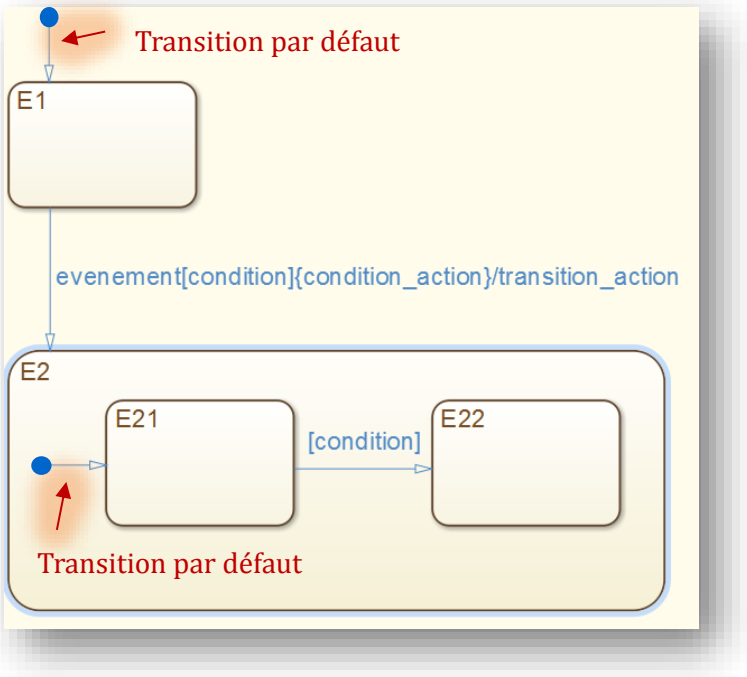
## 4.2- DEFAULT TRANSITION

Une transition par défaut impose l'activation d'un état en cas d'ambiguité.

Cette transition pointe un état mais n'a pas d'état source apparent. Le format de l'étiquette est identique à celui d'une transition ordinaire. Elle est obligatoire dès que le diagramme comporte au moins deux états.

Dans l'exemple ci-dessous lorsque la machine est excitée, E1 est activé.

Quand la transition entre E1 et E2 est franchie, le super-état ou état composite E2 est activé et E1 désactivé.
Au moment de l'activation de E2, E21 est alors activé…



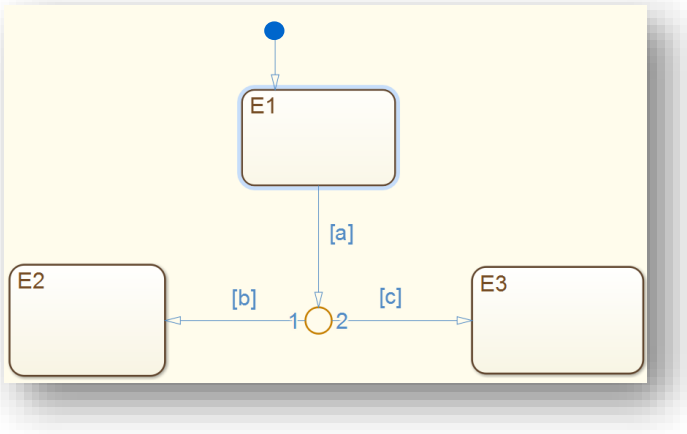

**Remarque :**
Ici le super-état E2 est le parent des sous-états E21 et E22.

E1 et E2 ont le même niveau hiérarchique.

E21 et E22 ont eux aussi le même niveau hiérarchique. Ce sont les états enfants de E2.

## 4.3- JUNCTION

La jonction (« **Junction** ») est un pseudo-état utilisé par exemple pour factoriser des expressions booléennes associées à des conditions.




Dans l'exemple ci-contre, lorsque la condition [a] est vraie, la jonction devient active (c'est un pseudo-état) mais E1 n'est pas désactivé pour autant.

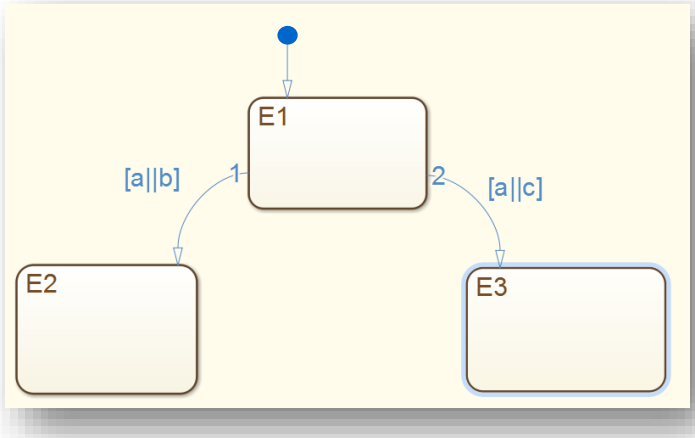
E2 sera activé si [b] devient vraie. E1 est alors simultanément désactivé.

Ou bien E3 sera activé si [c] devient vraie. E1 est alors simultanément désactivé.

Le comportement précédent est identique au suivant :

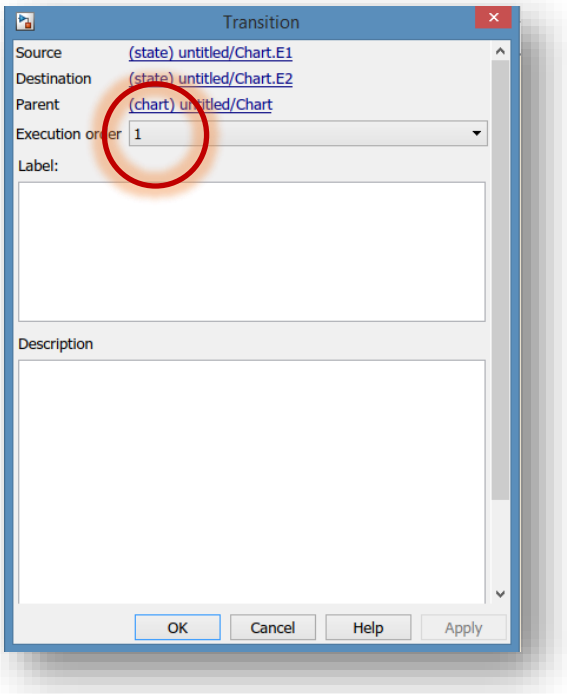
On remarquera que dans le premier cas l'évaluation de [b] se fait avant celle de [c] (transition dont la source est repérée 1 sur la jonction). Par conséquent en cas de simultanéité, l'activité de E2 sera prioritaire.

Cette priorité est retrouvée dans le second cas (transition dont la source est repérée 1). La condition [a||b] est évaluée avant [a||c].



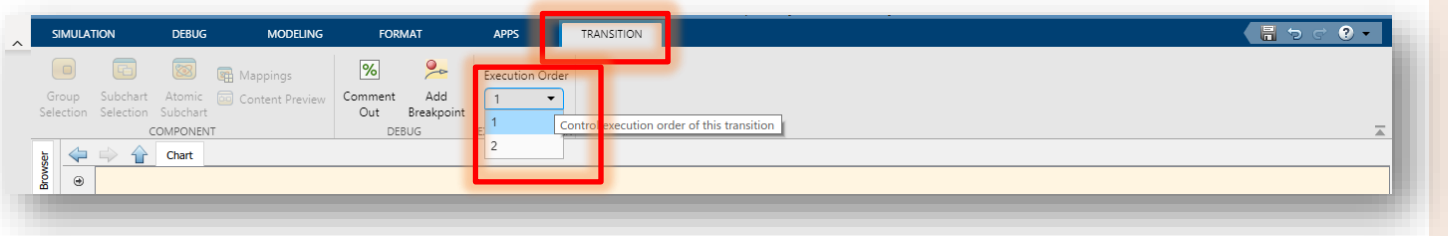
**Info** : la syntaxe utilisée est celle du langage C :

| | équivaut à…. |
|---|---|
| **[a\|\|b]** | **a+b** |
| **[a&&b]** | **a.b** |
| **[~a]ou[!a] suivant la version** | $\bar{a}$ |

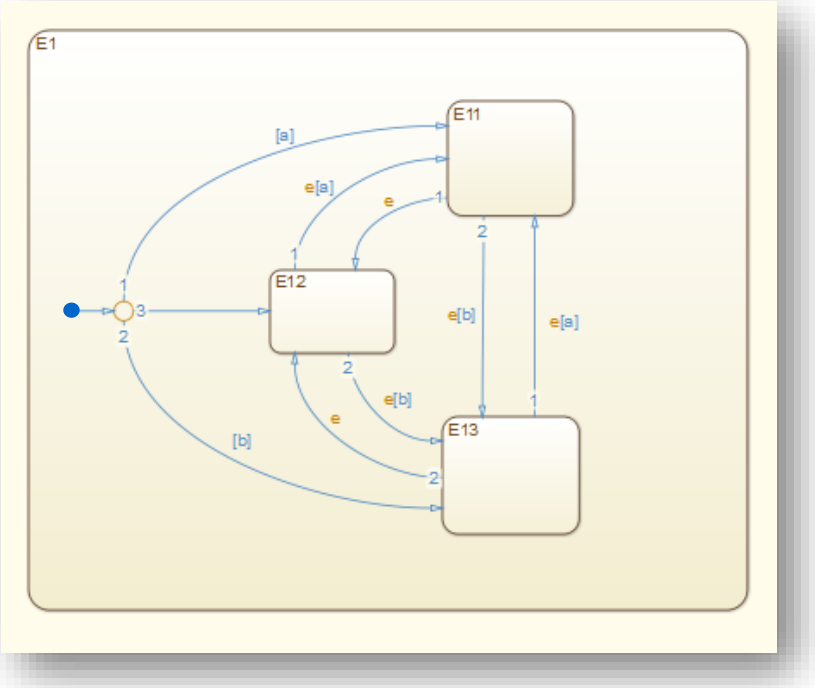



L'ordre d'évaluation des transitions peut être modifié en sélectionnant la ou les transitions à modifier, clic droit, « **Properties** ». Sélectionner l'ordre d'évaluation dans la liste déroulante « **Evaluation order** ».

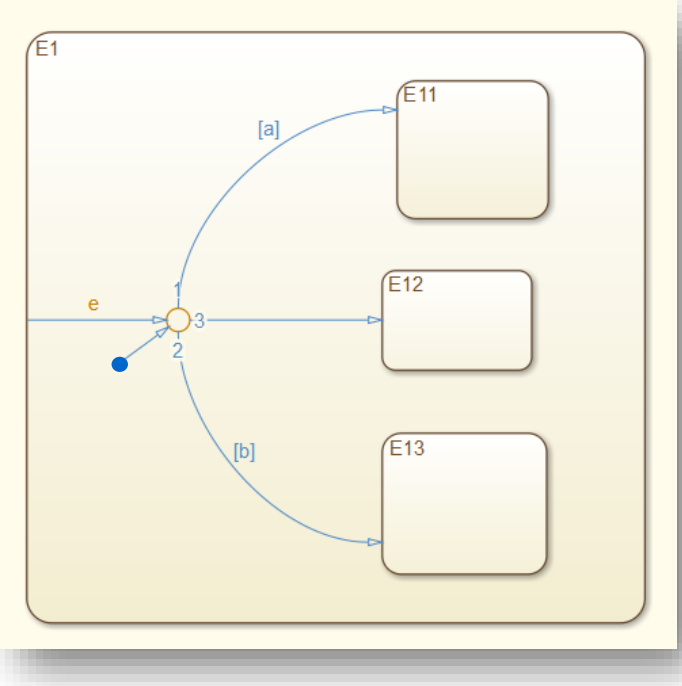
Une autre façon de modifier l'ordre d'évaluation des transistions, consiste à cliquer sur une transition et dans l'onglet « **TRANSITION** » qui apparait dans le bandeau supérieur sélectionner l'ordre d'évaluation souhaité :



Un autre exemple de simplification utilisant une transition interne :

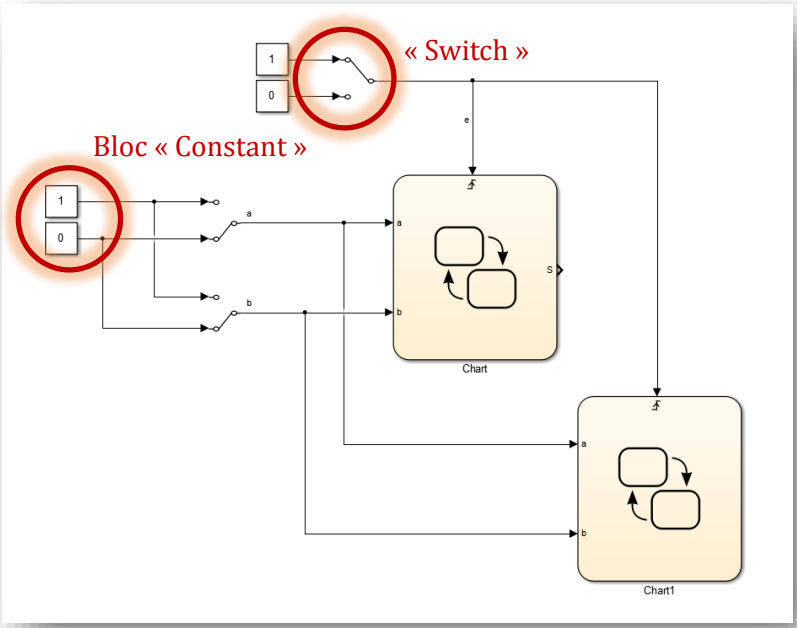



Le cas ci-contre présente un nombre de transitions important ce qui présente l'inconvénient d'alourdir le diagramme. Ici une simplification est envisageable en utilisant une transition interne, comme on peut le voir sur la figure qui suit.




Prenons le cas du passage de l'état E11, considéré actif, à l'état E13. Le passage se fera à l'occurence de l'évènement e et à condition que [b] soit vraie. Sur le premier diagramme la transition est clairement identifiable. Sur le second elle ne l'est pas. Néanmoins nous savons que si E11 est actif alors que l'évènement e survient, cet état est désactivé et la jonction ré-activée, et si [b] est vraie alors l'état E13 devient actif.

**Un petit exercice de saisie et de simulation…**

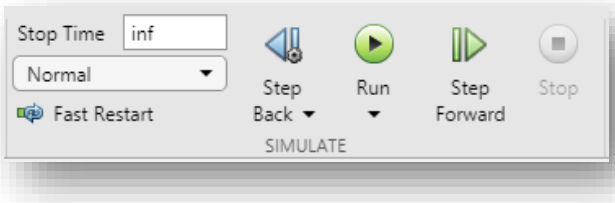



A titre d'exercice, vous pouvez assez facilement vérifier la similitude des comportements des deux machines.

Ci-contre se trouve le modèle Simulink® permettant de simuler et de comparer le comportement des deux machines d'état simultanément.

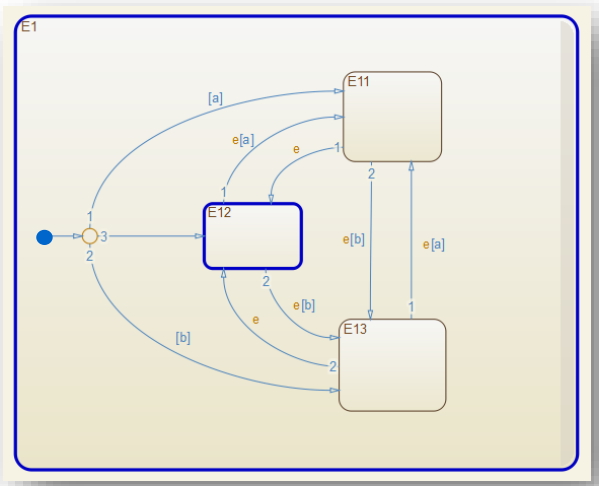
Les « **Switches** » se trouvent dans le menu « **Signal Routing** » de Simulink®.

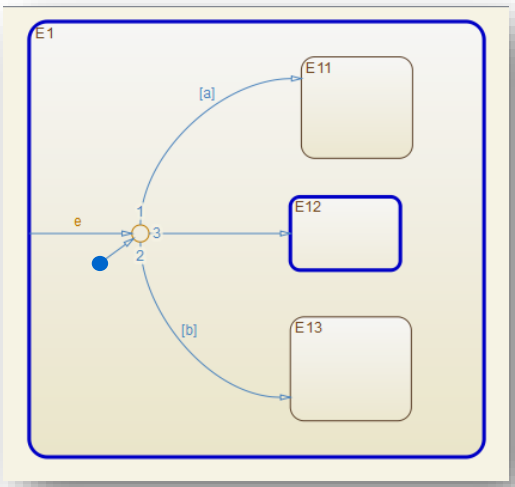
Les blocs 0 et 1 sont des blocs « **Constant** » qui se trouvent dans le menu « **Commonly Used Blocks** ».




Pour la simulation choisir un temps de simulation infini en saisissant « **inf** » dans le champ correspondant du bandeau supérieur.

Vérifier le comportement des machines d'état en double cliquant sur les « **Switches** » pendant la simulation pour les faire commuter.



## *A propos des « Flow Charts »...*

Les « **Flow Charts** » ou diagrammes de flux sont pratiques pour décrire les structures algorithmiques de base comme les structures alternatives ou de choix, et les structures itératives ou répétitives.

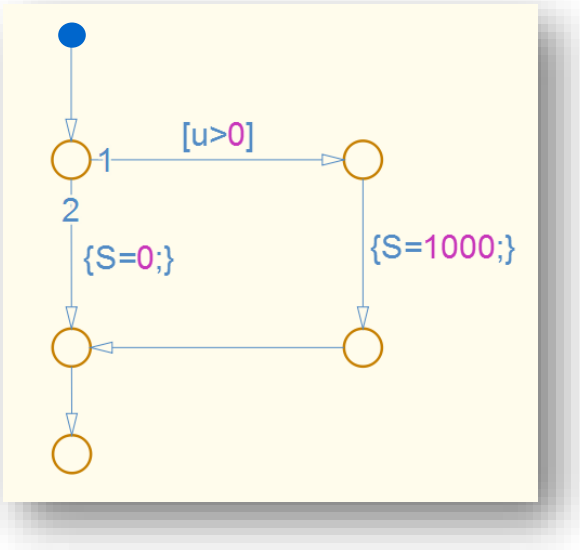
Le diagramme de flux contrairement aux diagrammes d'état (« **State Chart** ») ne comporte pas d'état mais uniquement des transitions et des jonctions.

**Exemple de diagramme de flux :**

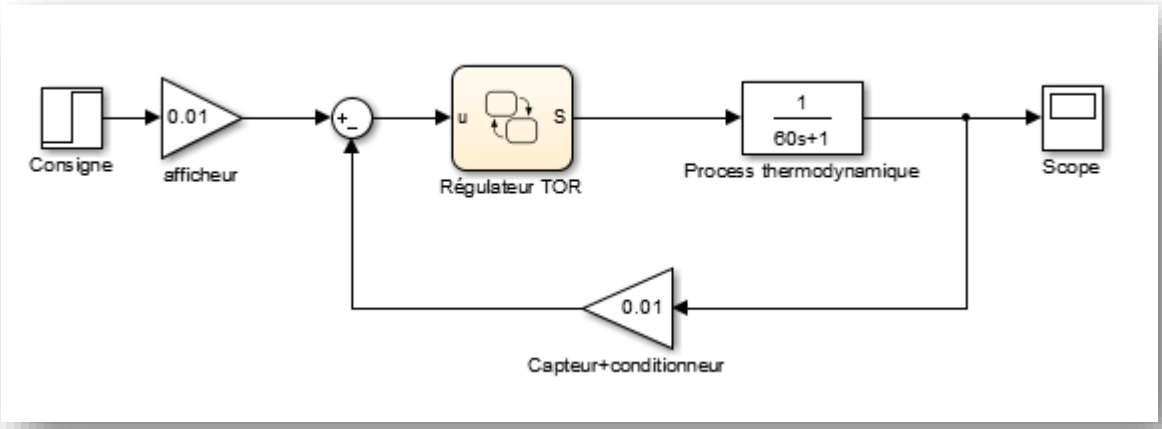
Il s'agit là d'une structure alternative :

Si u>0 alors S=1000 sinon S=0

L'entrée du diagramme se fait par une transition par défaut qui pointe une jonction. La structure se termine obligatoirement par une jonction.




Cet exemple peut être intégré à une boucle de régulation TOR de la température d'un four industriel :

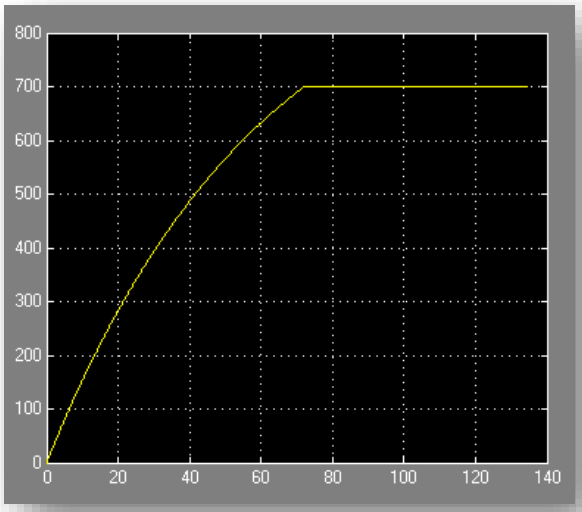


Le système thermodynamique est modélisé par une fonction de transfert du premier ordre de gain unitaire et de constante de temps 60s.

La sortie S du régulateur Tout Ou Rien (TOR) correspond à la puissance électrique nécessaire au chauffage des résistances. La puissance maximale admissible est 1000 W.

La consigne est un échelon de température de 700 °C.

Le résultat de la simulation s'affiche dans le « **Scope** » :



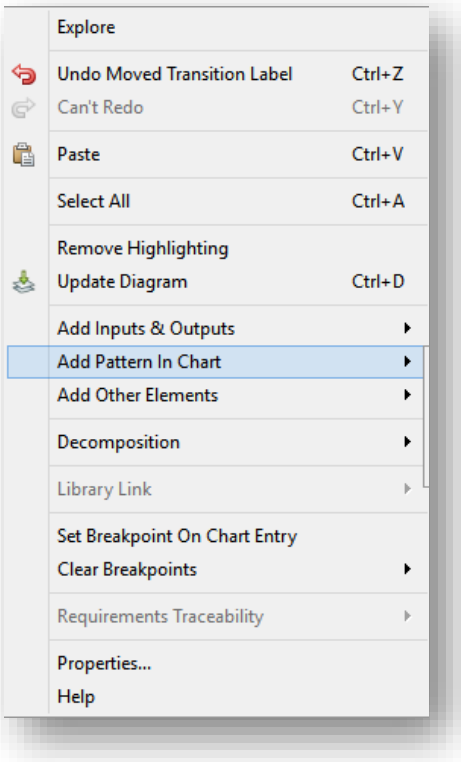

Si la température de consigne n'est pas atteinte, l'écart u est positif et dans ce cas les résistances sont alimentées.

Quand l'écart est nul les résistances ne sont plus alimentées.

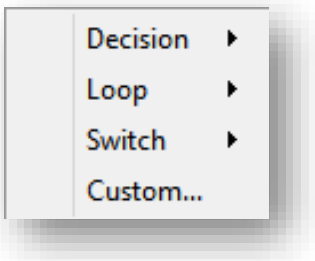
En absence de perturbation c'est-à-dire sans injection d'air frais, la température du four reste constante.

Le tracé du diagramme de flux précédent peut se faire par glissé-déposé de jonctions et en reliant ces différentes jonctions les unes aux autres. Reste alors à saisir les conditions et les actions.

Stateflow® propose un outil qui génère automatiquement les structures algorithmiques. Pour ajouter une structure dans un diagramme on peut utiliser la commande « **Add Pattern In Chart** » du menu contextuel.



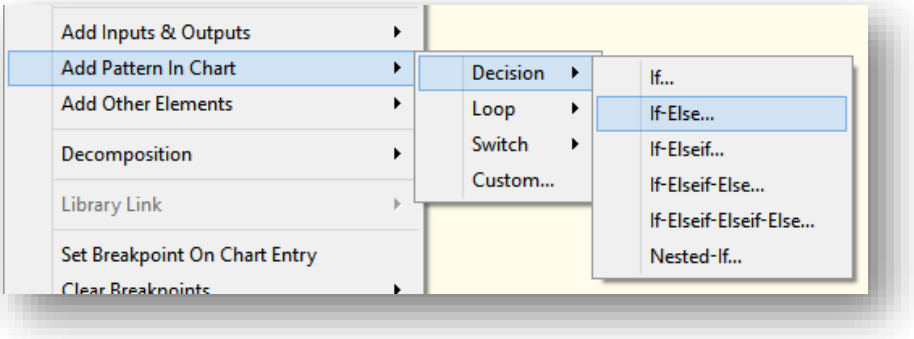

Un second menu s'ouvre proposant différentes structures algorithmiques, dont :



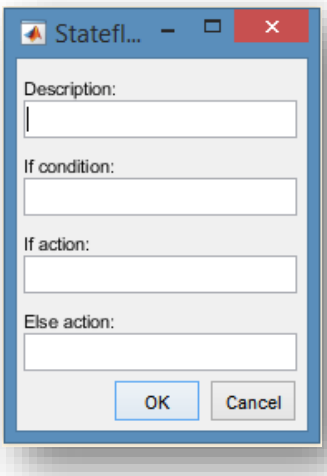

« **Decision** » : structures alternatives,

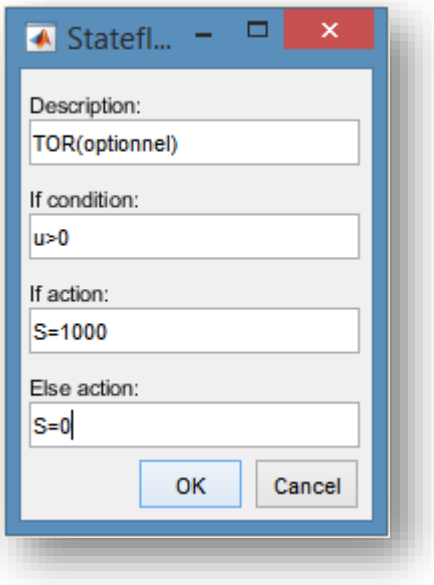
« **Loop** » : structures répétitives.

Dans notre cas nous allons cliquer sur « **Decision** » et choisir la structure alternative « **if-else…**» :



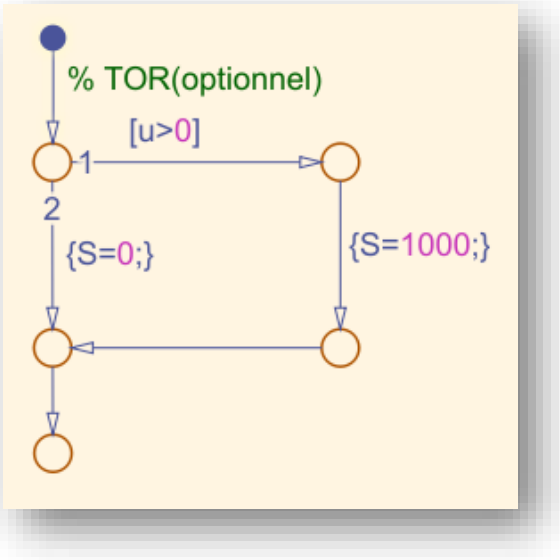

La fenêtre ci-contre s'ouvre :



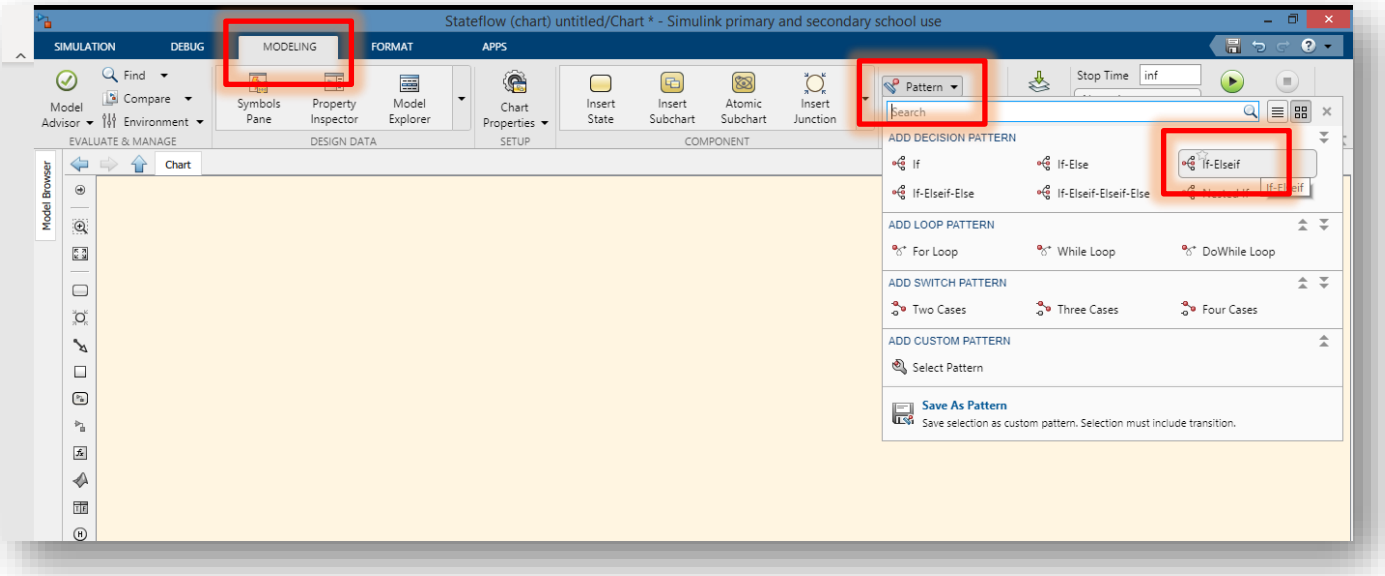



Il suffit de la remplir comme suit pour générer automatiquement la structure souhaitée en cliquant sur le bouton OK.

Nous obtenons bien le diagramme attendu :




Un autre moyen d'ajout de structure est de cliquer sur « **Pattern** » dans le menu « **EDIT** » de l'onglet « **MODELING** » du bandeau supérieur :



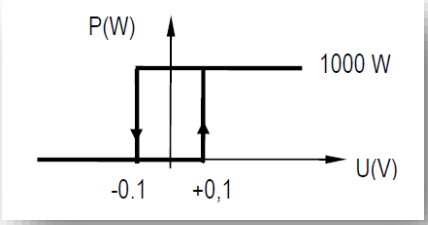
Dans la pratique on introduit une plage de mise en service du régulateur TOR autour d'un point de fonctionnement en introduisant un hystérésis.

Celui-ci peut être ajustable ou bien fixe. Il est donné communément par les constructeurs en pourcentage de la pleine échelle de la sortie.

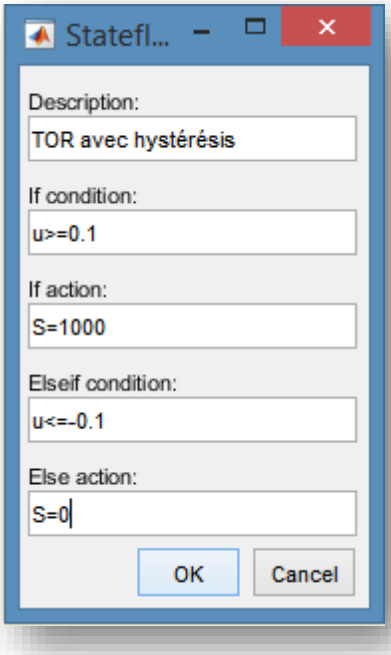
Par exemple, pour le régulateur OMRON E5CSV en mode TOR (ON-OFF) le constructeur annonce un hystérésis de 0.2% de la pleine échelle (Full Scale : FS) :

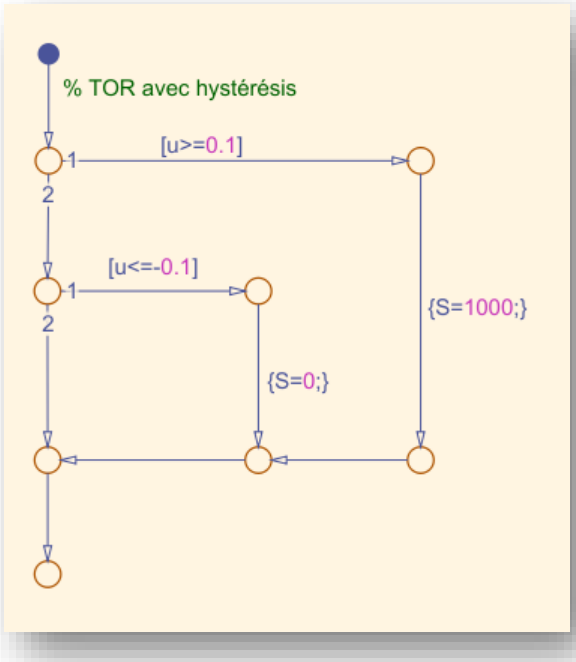

| | | |
|---|---|---|
| **Setting accuracy** | Thermocouple (See note 1.): | (±0.5% of indication value or ±1°C, whichever is greater) ±1 digit max. |
| **Indication accuracy (ambient temperature of 23°C)** | Platinum resistance thermometer (See note 2.): | (±0.5% of indication value or ±1°C, whichever is greater) ±1 digit max. |
| **Influence of temperature** | R thermocouple inputs: | (±1% of PV or ±10°C, whichever is greater) ±1 digit max. |
| **Influence of voltage** | Other thermocouple inputs: | (±1% of PV or ±4°C, whichever is greater) ±1 digit max. |
| | Platinum resistance thermometer inputs: | (±1% of PV or ±2°C, whichever is greater) ±1 digit max. |
| **Hysteresis (for ON/OFF control)** | 0.2% FS (0.1% FS for multi-input (thermocouple/platinum resistance thermometer) models) | |
| **Proportional band (P)** | 1 to 999°C (automatic adjustment using auto-tuning/self-tuning) | |
| **Integral time (I)** | 1 to 1,999 s (automatic adjustment using auto-tuning/self-tuning | |
| **Derivative time (D)** | 1 to 1,999 s (automatic adjustment using auto-tuning/self-tuning) | |

Introduisons alors un hystérésis dans le modèle du régulateur:

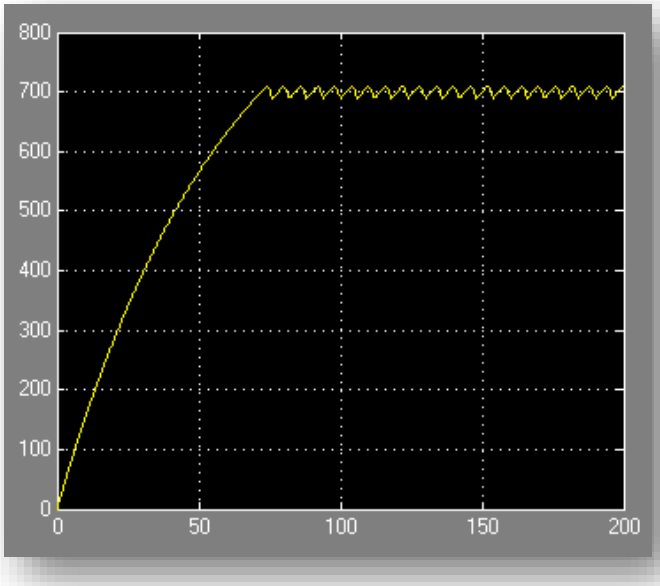



Une structure « **If-Elseif…** » convient :

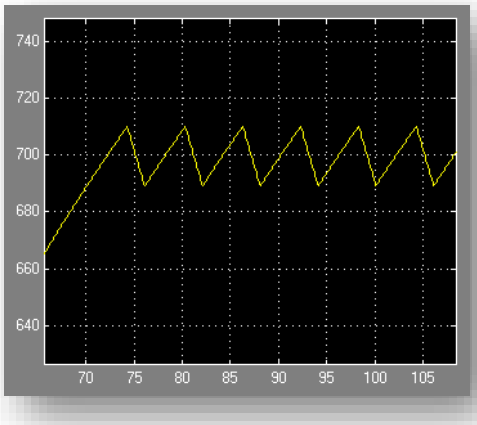


La simulation donne le résultat suivant :




La température évolue à plus ou moins 10 °C autour de 700 °C.

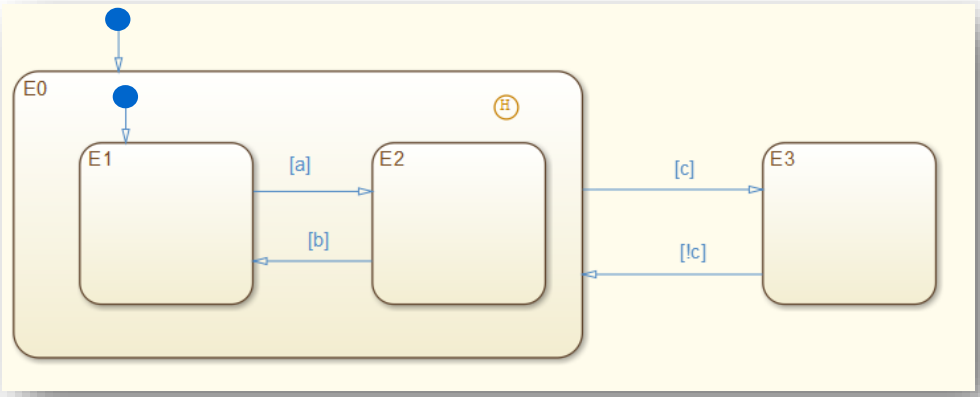



Ce résultat se retrouve aisément par un calcul simple.

La simulation a été effectuée sur 200s avec un pas fixe du solveur de 0.1s.

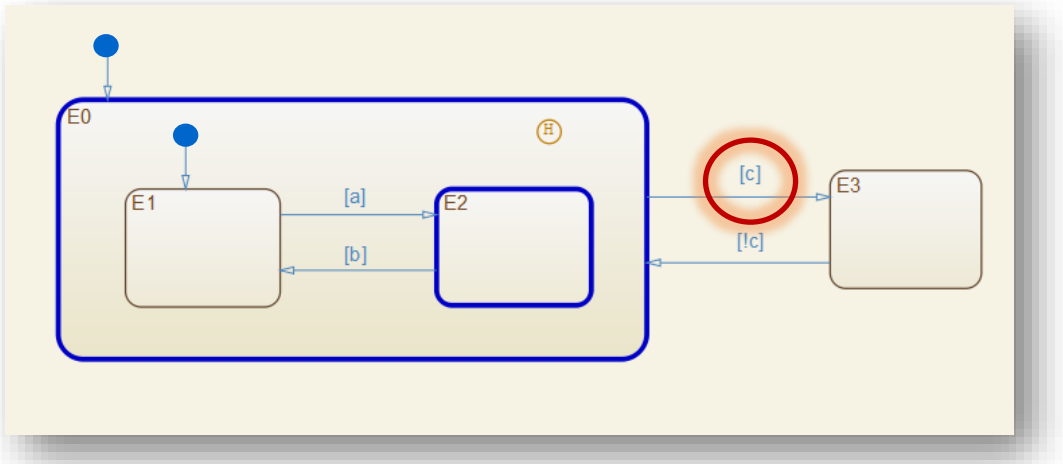
## 4.4- HISTORY JUNCTION

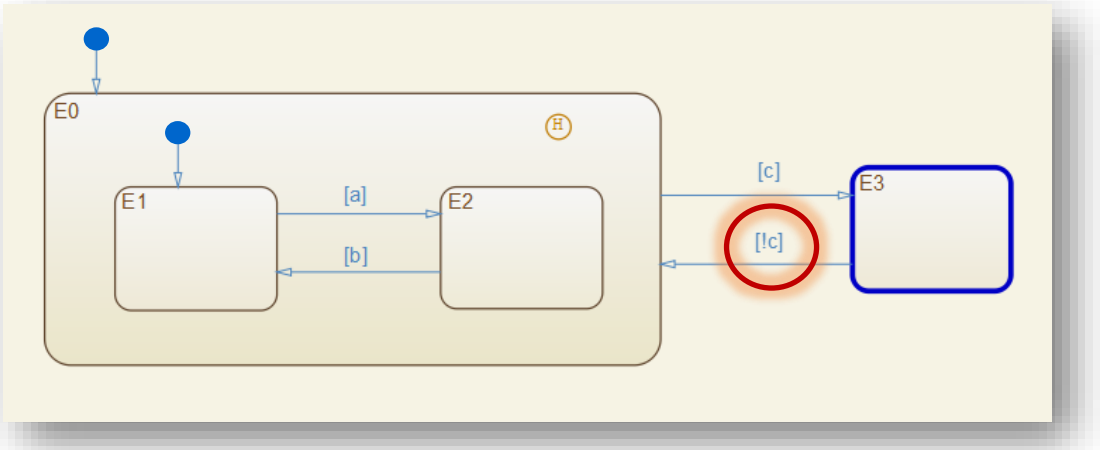
Il s'agit d'un pseudo-état qui mémorise l'activité des sous-états d'un super-état ou état composite. Cela permet donc de réactiver un état dans la situation dans laquelle on l'a quitté. Les propriétés de la «**History junction** » ne s'appliquent qu'au niveau de hiérarchie dans lequel elle apparait.

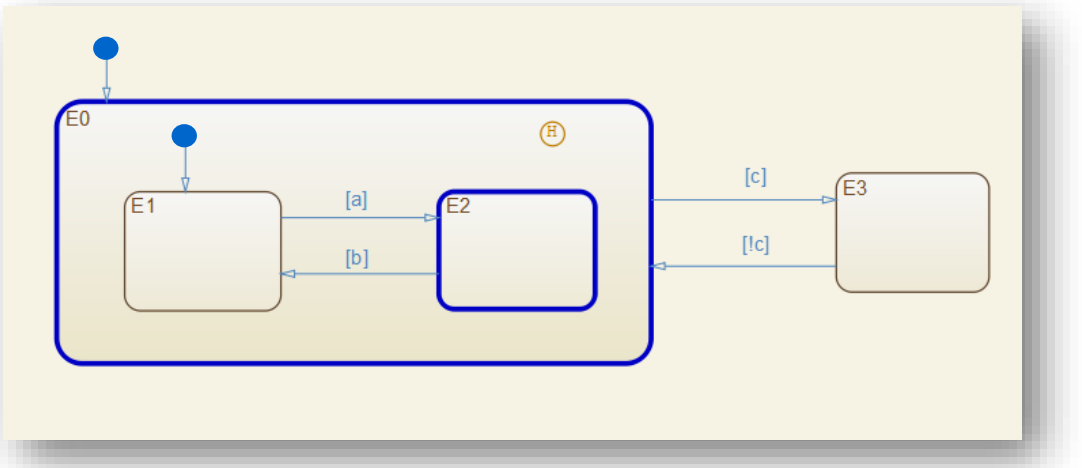



Considérons que l'état E0 et l'état E2 soient actifs. Si la condition [c] est vraie, E3 est alors activé et E0 désactivé.

La jonction « **History** » permet de mémoriser l'activation des sous-états au moment de la désactivation de E0, ici on a mémorisé l'activation de E2. Si bien que lorsque la condition [!c] devient vraie, E3 est désactivé et E0 est activé à nouveau mais la présence de la jonction « **History** » permet de ré-activer E2.

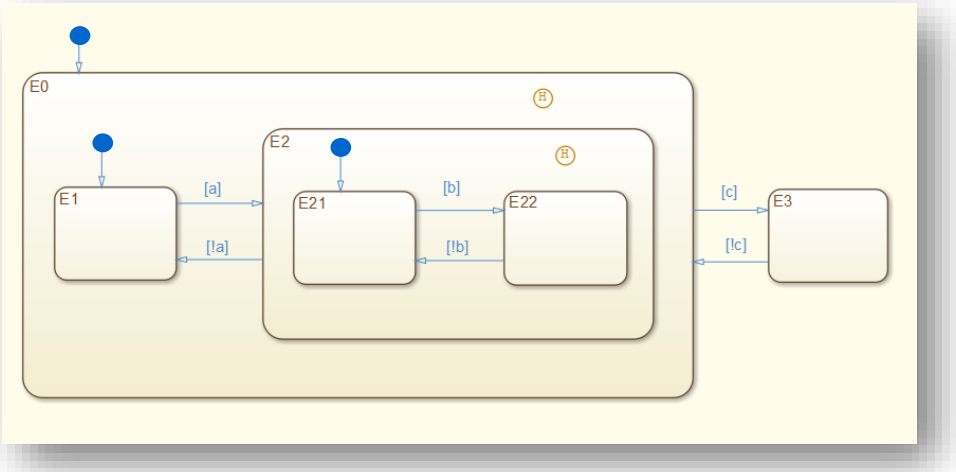




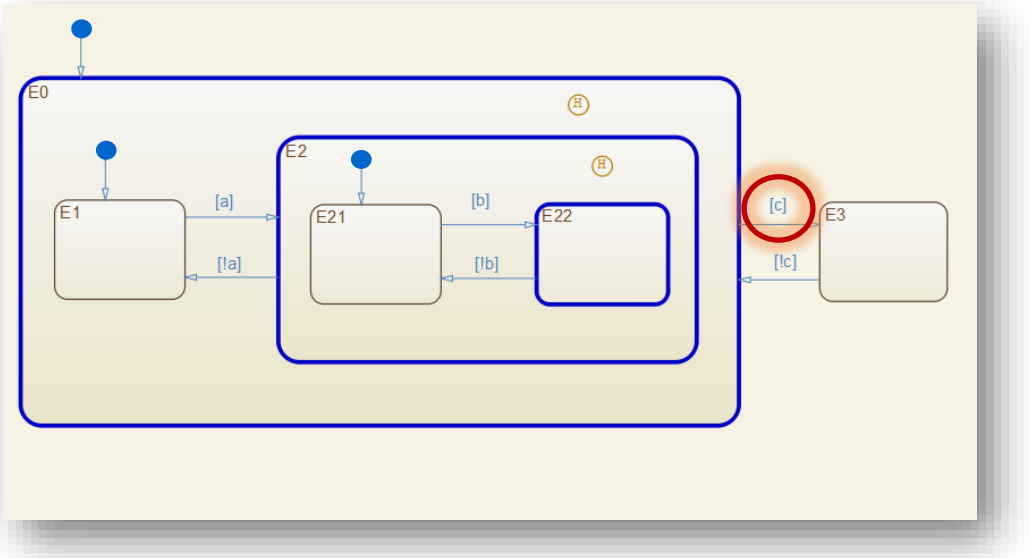

On notera qu'en absence de la jonction « **History** », la transition par défaut pointant l'état E1 imposerait l'activation de cet état à la réactivation de l'état E0.

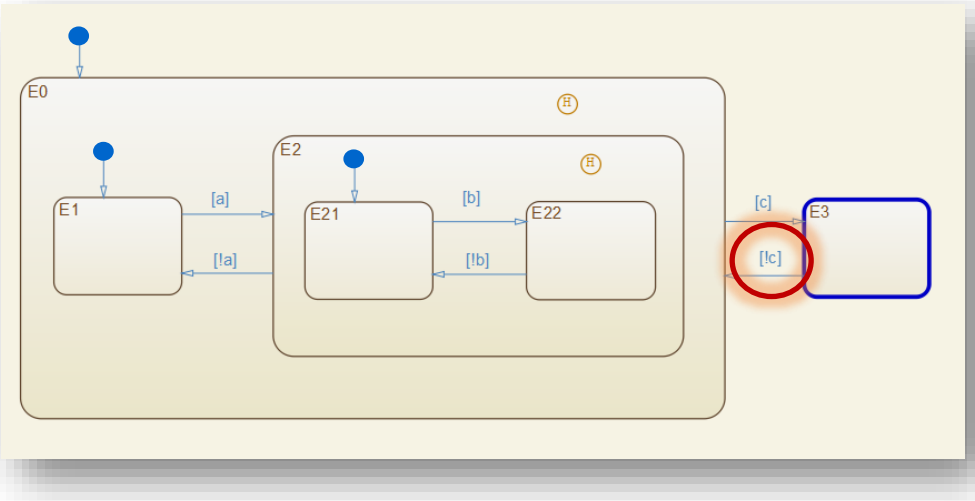
**Autre exemple** :

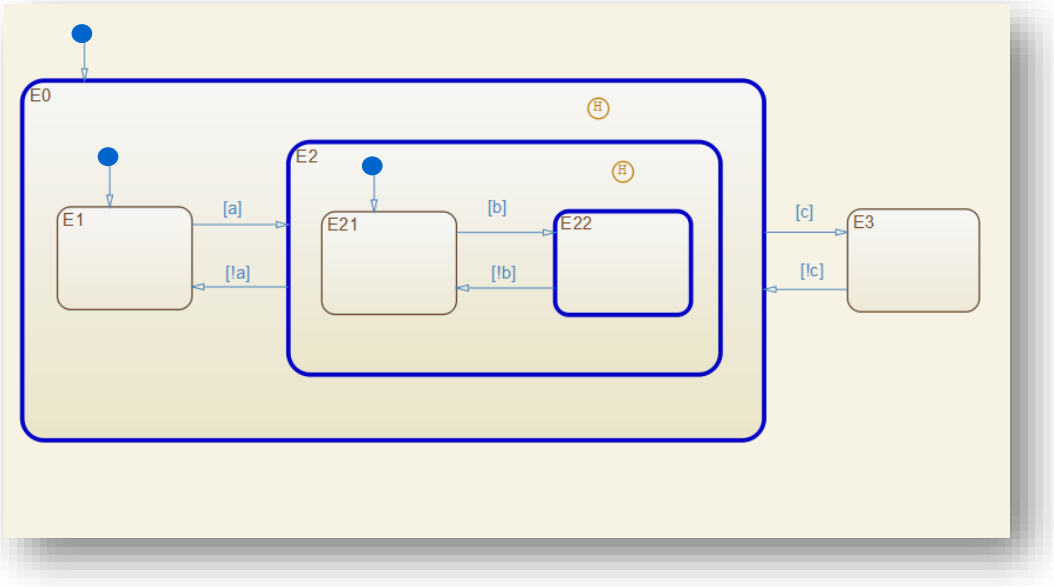



Nous retrouvons maintenant dans le diagramme ci-dessus deux jonctions « **History** ». Elles apparaissent à deux niveaux hiérarchiques différents.

Imaginons que E0, E2, E22 soient actifs. Quand la condition [c] est vraie, E3 est activé alors que les trois états précédents sont désactivés. Lorsque la condition [!c] devient vraie à son tour on retrouve la situation quittée c'est-à-dire E0, E2, E22 actifs.



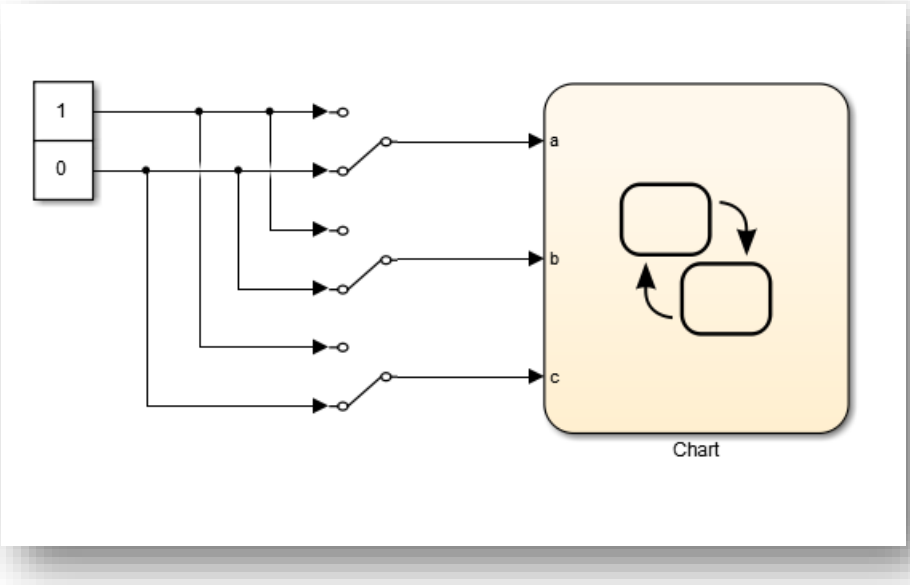






**Remarque** : si la jonction « **History** » n'existait pas dans l'état E2, E22 ne serait pas activé mais ce serait bien l'état E21 compte tenu de la présence de la transition par défaut sur E21.

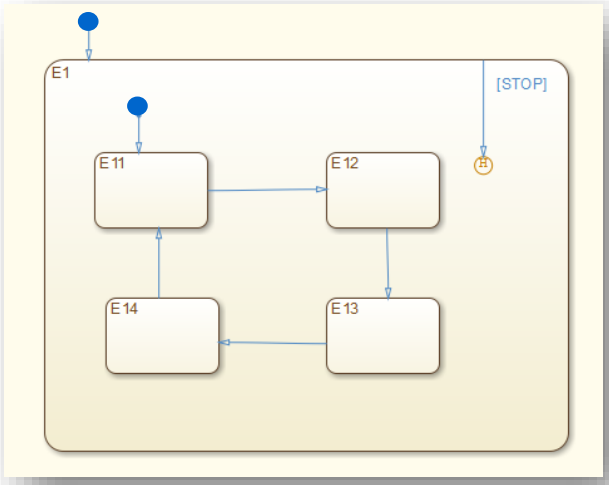
**Un autre petit exercice de saisie et de simulation…**

Des simulations sont possibles en disposant des jonctions « History » où vous le souhaitez. Ci-dessous le modèle Simulink® correspondant :

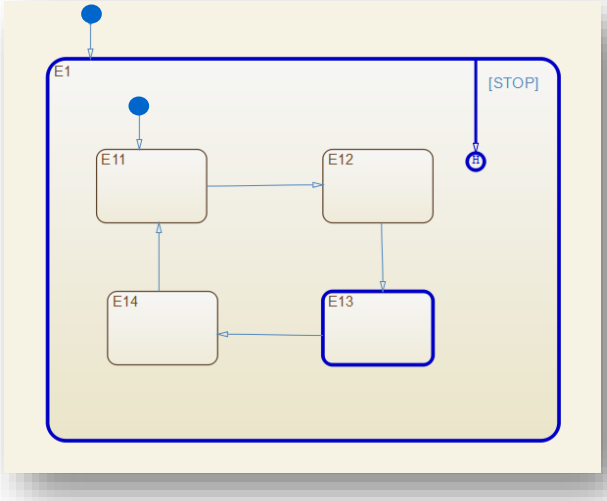



Dans le paragraphe consacré aux transitions internes, nous avons évoqué la situation où celle-ci pouvait pointer une jonction « **History** ».

Ce sera le cas lorsqu'on souhaite par exemple figer la machine dans son état courant :




En effet dans cet exemple, si la condition [STOP] est vraie alors qu'un des quatre états E11, E12, E13, E14 est actif celui reste actif. L'activité de l'état courant a été mémorisée.



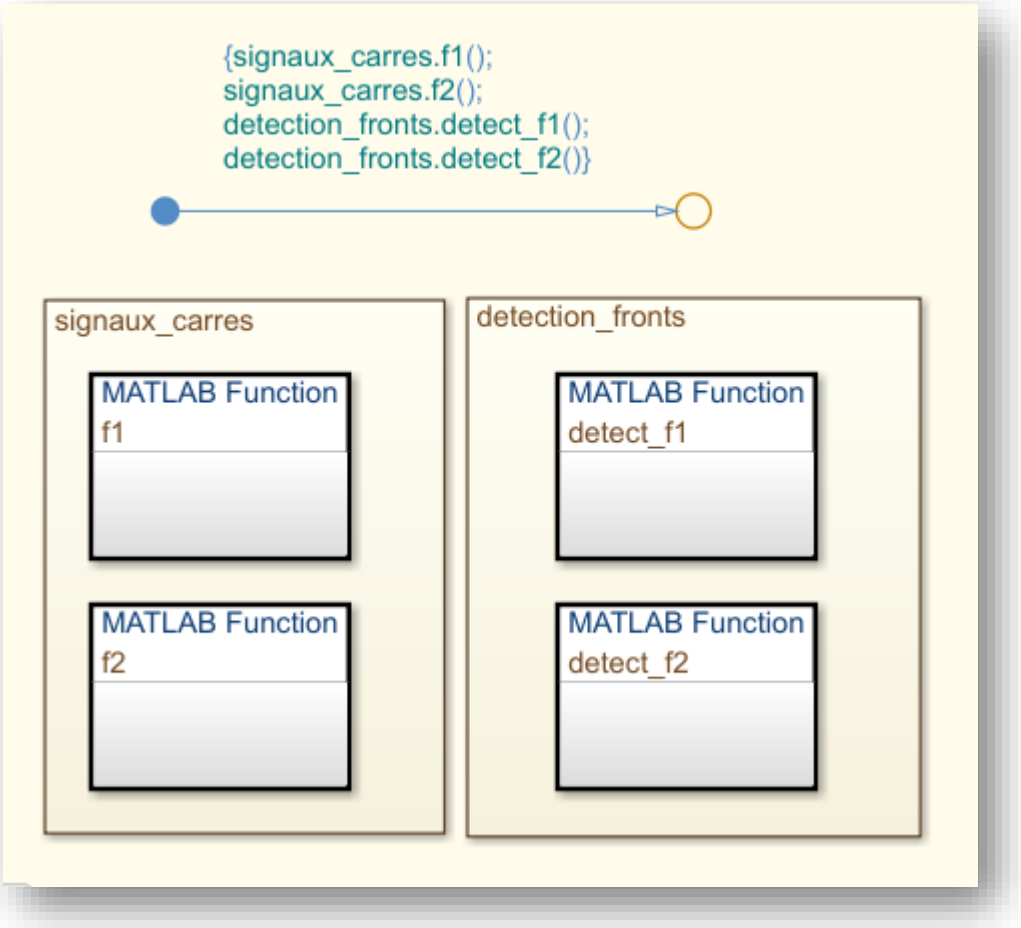

Dès que la condition [STOP] n'est plus vraie l'état E1 reprend son mode d'évolution normal.

## 4.5- BOX

La boîte (« **Box** ») est un élément qui permet le regroupement d'objets graphiques de Stateflow® comme des états, des fonctions… Les « **Boxes** » peuvent être réutilisées dans d'autre modèles.

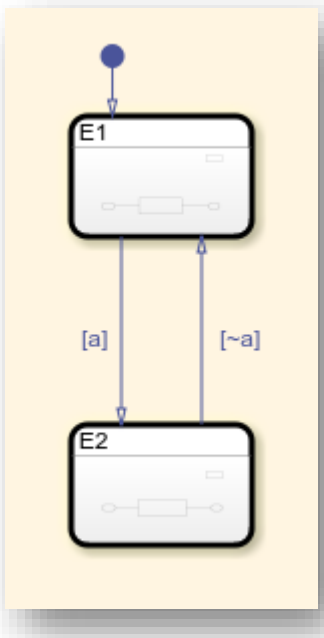
Par exemple, ci-dessous le « **Chart** » fait appel à quatre fonctions MATLAB qui sont « rangées » par paires dans deux « **boxes** » : « **signaux_carres** » et « **detection_fronts** ».

Nous construirons et commenterons un peu plus tard ce diagramme.



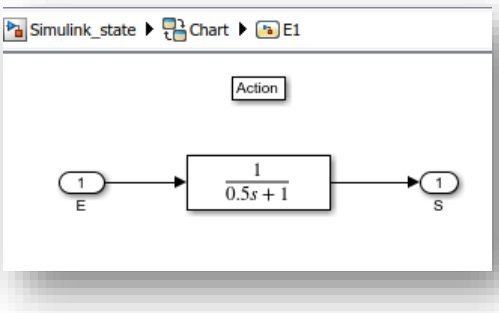
## 4.6- SIMULINK BASED STATE

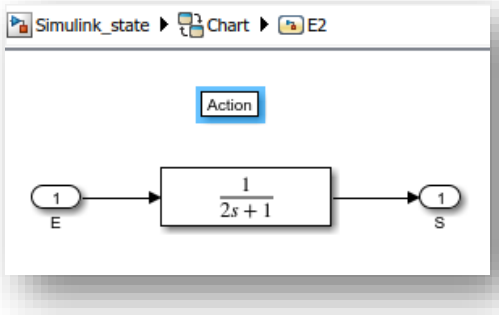
Cet élément est utilisé dans le cadre de la modélisation de systèmes hybrides, c'est-à-dire des systèmes qui basculent d'un état à l'autre sous certaines conditions et pour lequel l'activité est continue. Cette activité peut ainsi être décrite par un modèle Simulink®. Prenons un cas très simple :




Comme le montre le diagramme ci-contre Le système possède 2 états. Le passage de l'état E1 à l'état E2 se fait si la condition a est vraie et le passage de l'état E2 à l'état E1 se fait si la condition a n'est plus vraie.

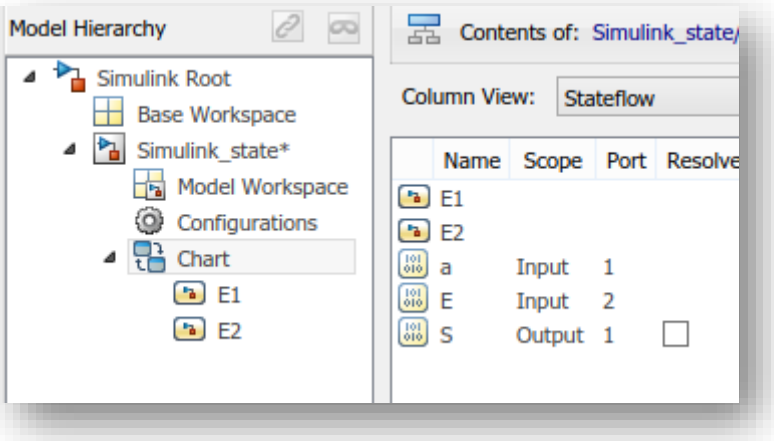
E1 et E2 sont deux états Simulink (« **Simulink state** »). Dans ces états ont été créés des modèles Simulink® élémentaires. Le bloc « **Action** » est automatiquement créé :



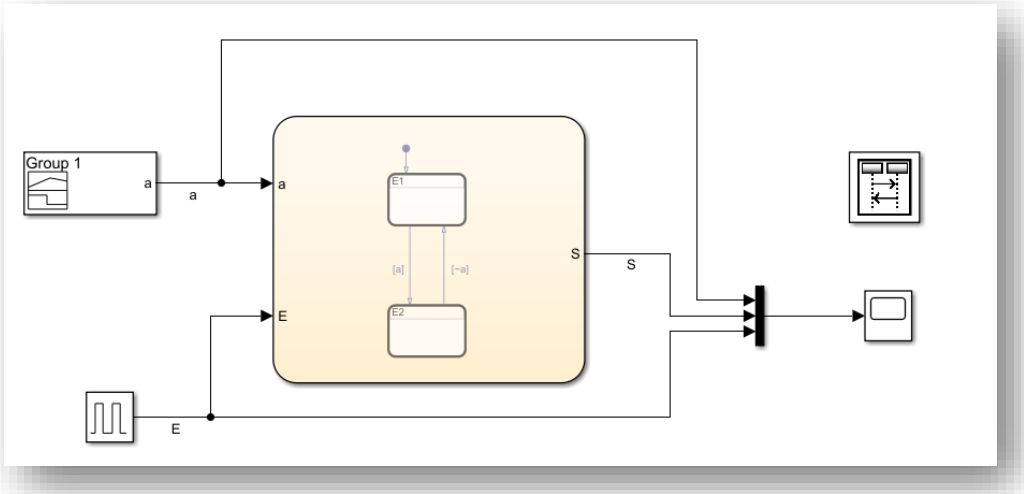

Lorsque E1 est acivé, le comportement du système est celui d'un élément du premier ordre de gain unitaire et de constante de temps 0.5 s. Lorsque E1 est activé, le

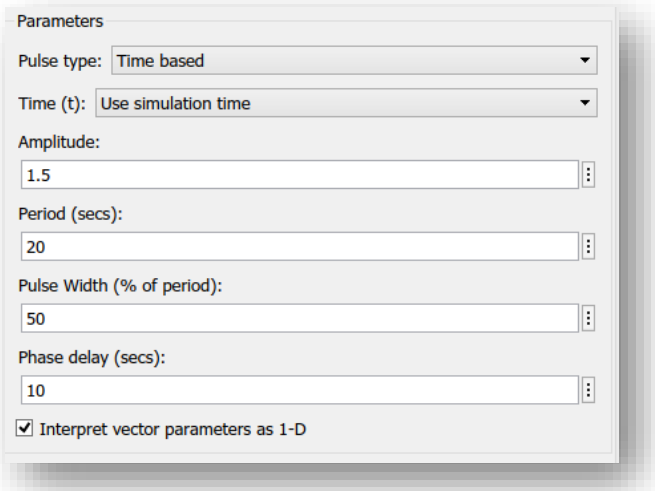
comportement du système est celui d'un élément du premier ordre de gain unitaire et de constante de temps 2 s.




Les entrées et sorties mises en place dans les modèles Simulink® correspondent aux entrées et sorties du « Chart ». Les variables associées à ces entrées et sorties, ici E et S, doivent être saisies dans l'explorateur de modèle.

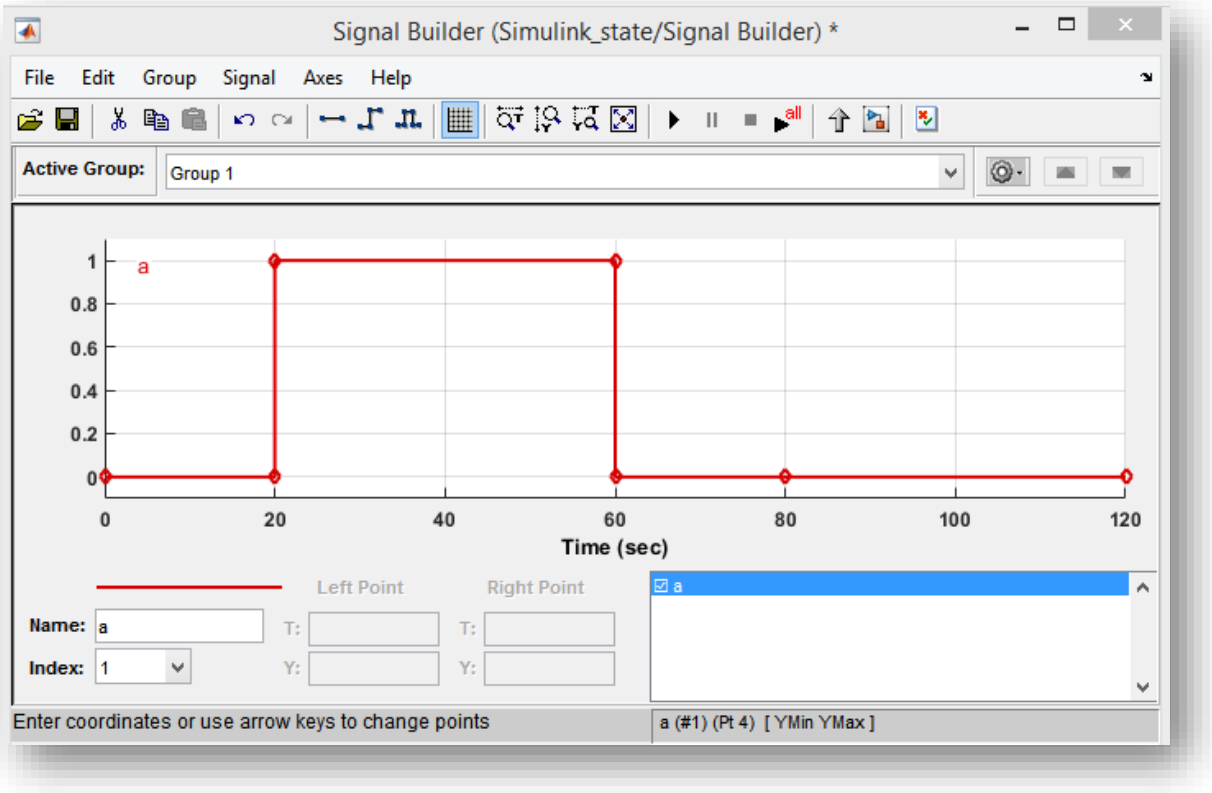
Le modèle de premier niveau est le suivant :



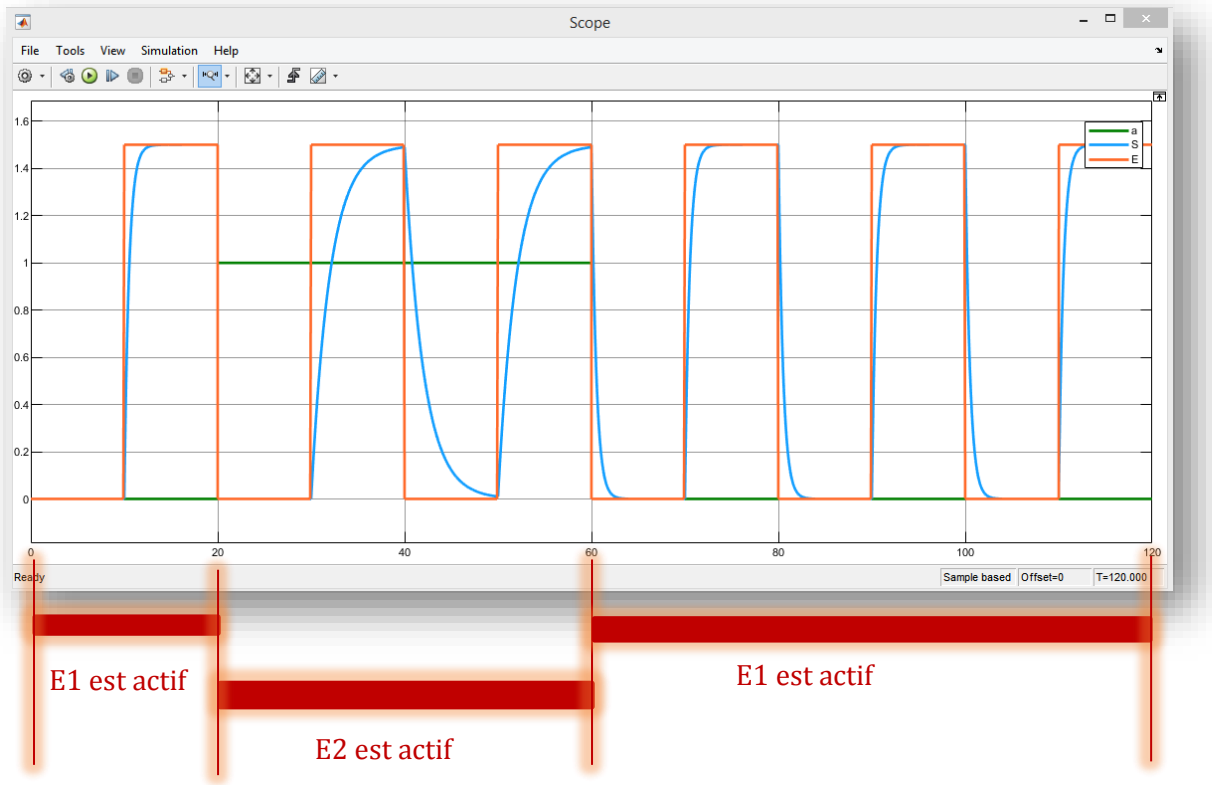




L'entrée E est un signal carré d'amplitude 1.5 et de période 20 s.

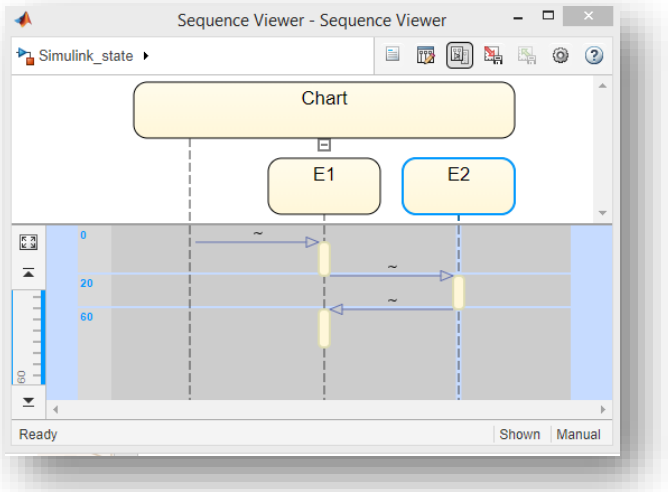
A la variable a est associé le signal suivant :




Un « Scope » affiche les résultats de la simulation qui s'étend sur 120s :



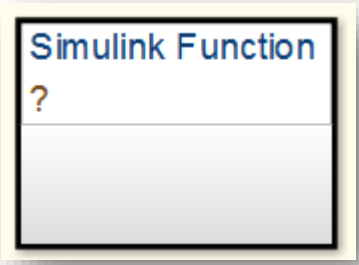

Ce que confirme le « **Sequence Viewer** » :




## 4.7- SIMULINK FUNCTION

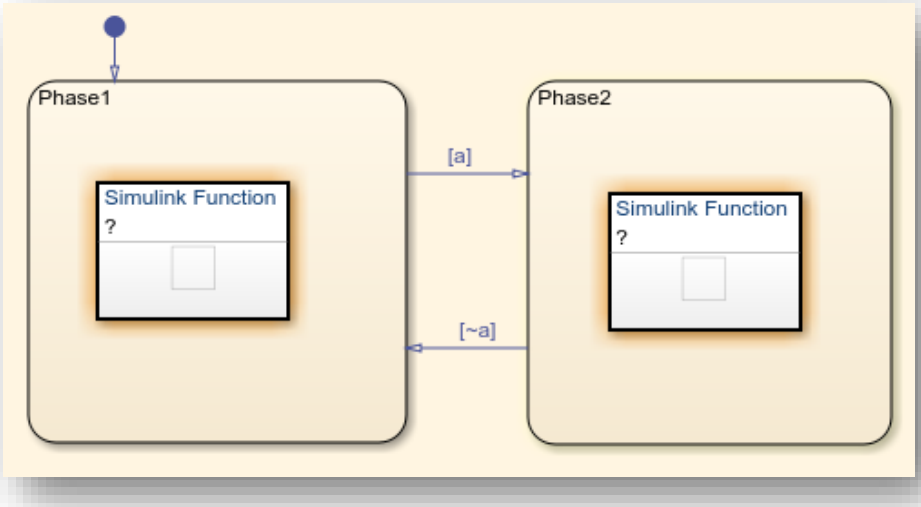
Cet outil permet l'introduction de fonctions utilisant des schémas blocs construits avec les différentes bibliothèques de Simulink®.

Une fonction Simulink® est représentée par un bloc rectangulaire glissé-déposé dans un « **Chart** ». :




Prenons un exemple simple pour illustrer la démarche.

Nous allons considérer une machine à état constituée de deux états : Phase1 et Phase2

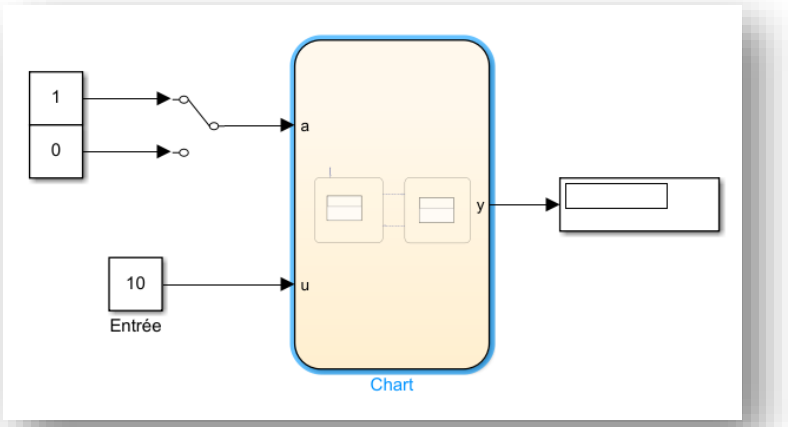



L'état
« **Phase1** » est initialement actif. L'activation de l'état « **Phase2** » se fait si la

condition [a] est vraie. Inversement la réactivation de l'état « **Phase1** » se fait si la condition [ ~a] est vraie.

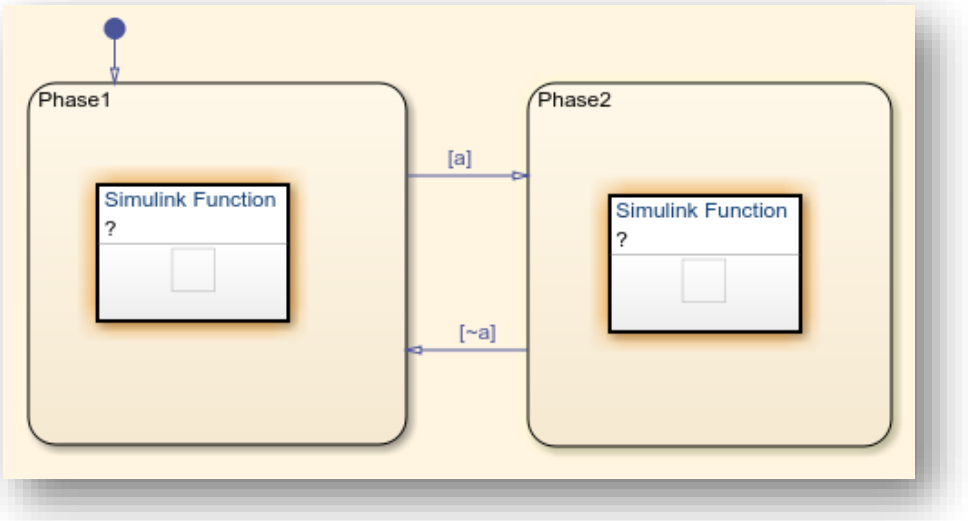
La phase 1 consiste à ajouter 5 à l'entrée du système considérée comme constante et valant 10. La phase 2 multiplie par 3 la même entrée.

L'interface Simulink® sera alors la suivante :



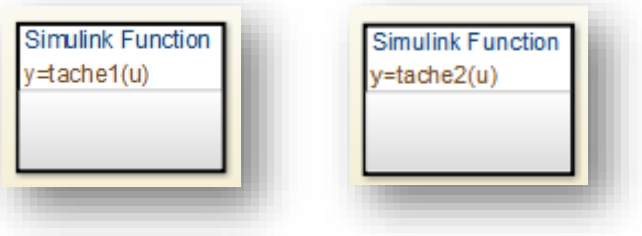

Un double-clic sur le « **Chart** » affiche le diagramme déjà présenté quelques lignes plus hautes.

Nous allons insérer deux fonctions Simulink® qui permettront de réaliser les deux phases décrites précédemment :



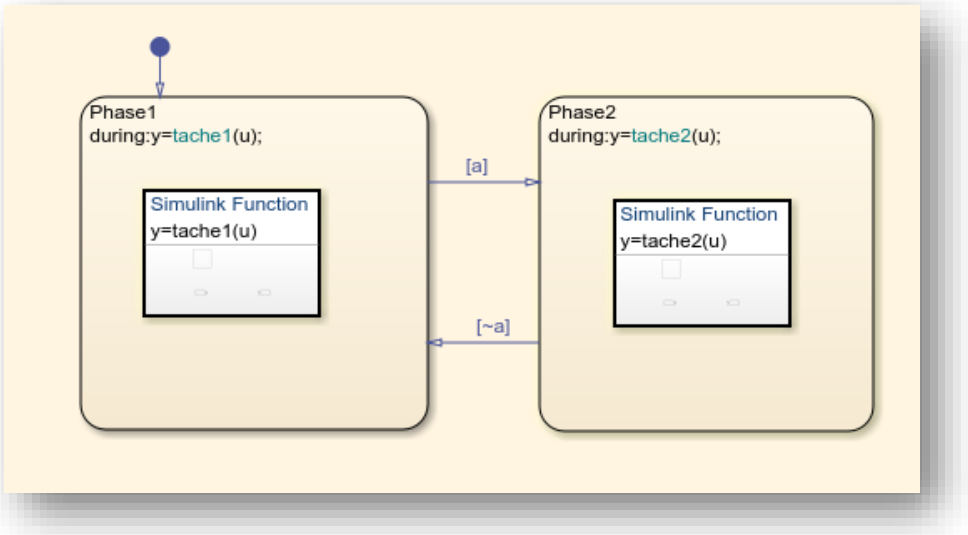

En cliquant sur le point d'interrogation on précise la signature de la fonction qui sera appelée par l'état qui déclenchera le ou les calculs souhaités.

La signature de la fonction précise son nom, ses arguments et sa valeur de retour : « **tache_i** » est le nom de la fonction, « **u** » son argument et « **y** » sa valeur de retour :

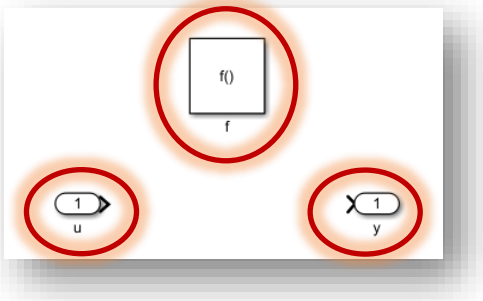



D'une manière générale la syntaxe est la suivante : $(r_1, r_2, \ldots r_n) = \text{nom}(a_1, a_2, \ldots a_n)$

L'invocation de la fonction, ou son appel, se fait pendant l'activité de l'état « **Phase1** » ou « **Phase2** ». A un état correspond une fonction particulière. Par conséquent les fonctions sont insérées dans les états correspondants :




Reste à définir les modèles Simulink® associés aux fonctions en double-cliquant sur les blocs « **Simulink Function** » :

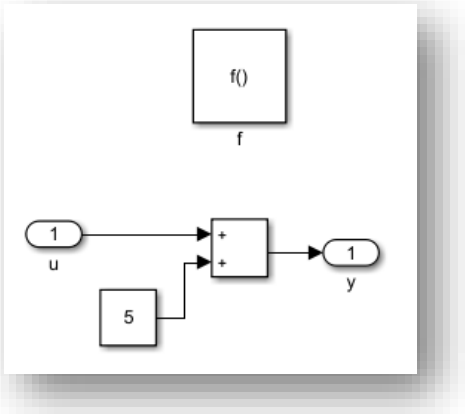


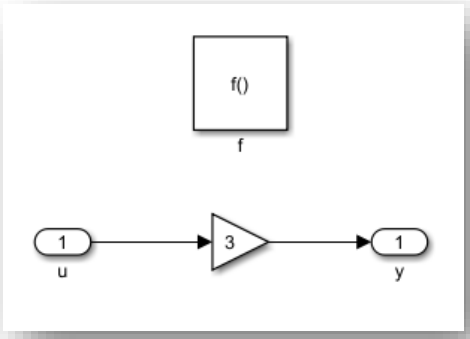

On constate la génération automatique de l'argument de la fonction et de la valeur de retour qui correspondent respectivement à l'entrée et à la sortie du modèle Simulink®.

D'autre part s'affiche un bloc précisant que le sous-système fait appel à une fonction.

Préparons le modèle Simulink® associé à la première phase :

On ajoute 5 à l'entrée « **u** »:

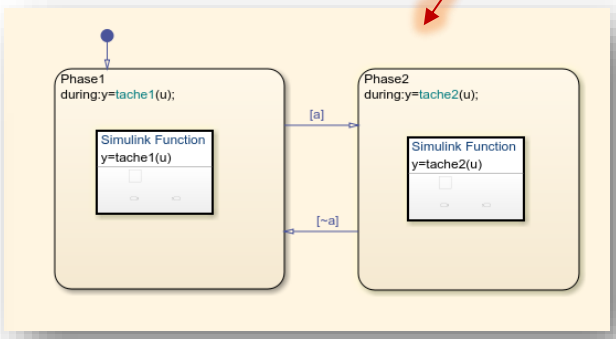


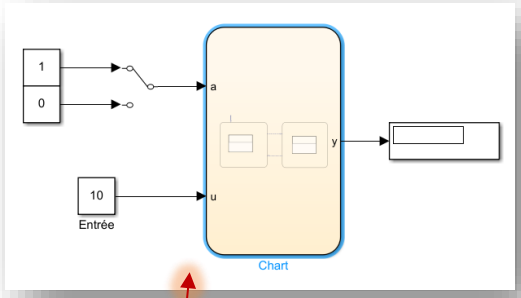




Pendant la seconde phase, « **u** » est multiplié par 3 :

Le modèle global comporte donc les éléments suivants :

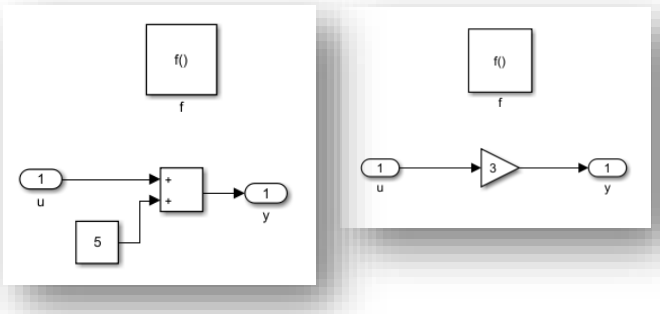
« Chart »



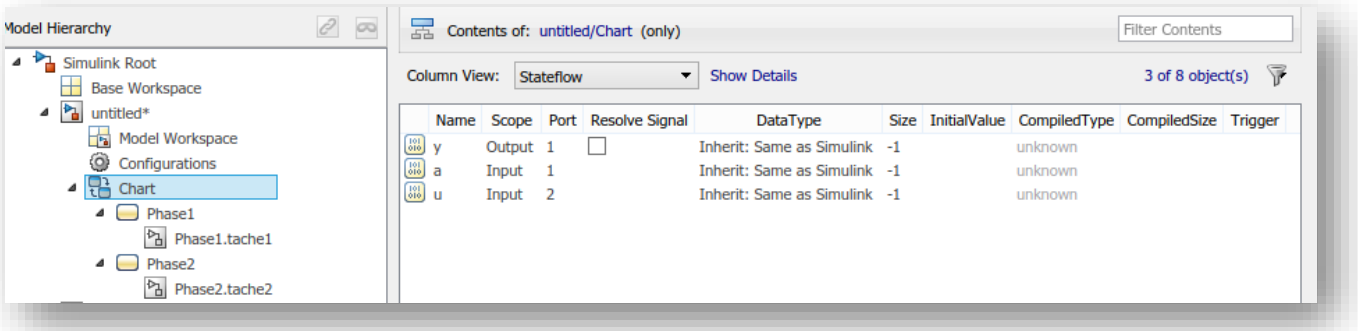




Interface Simulink®

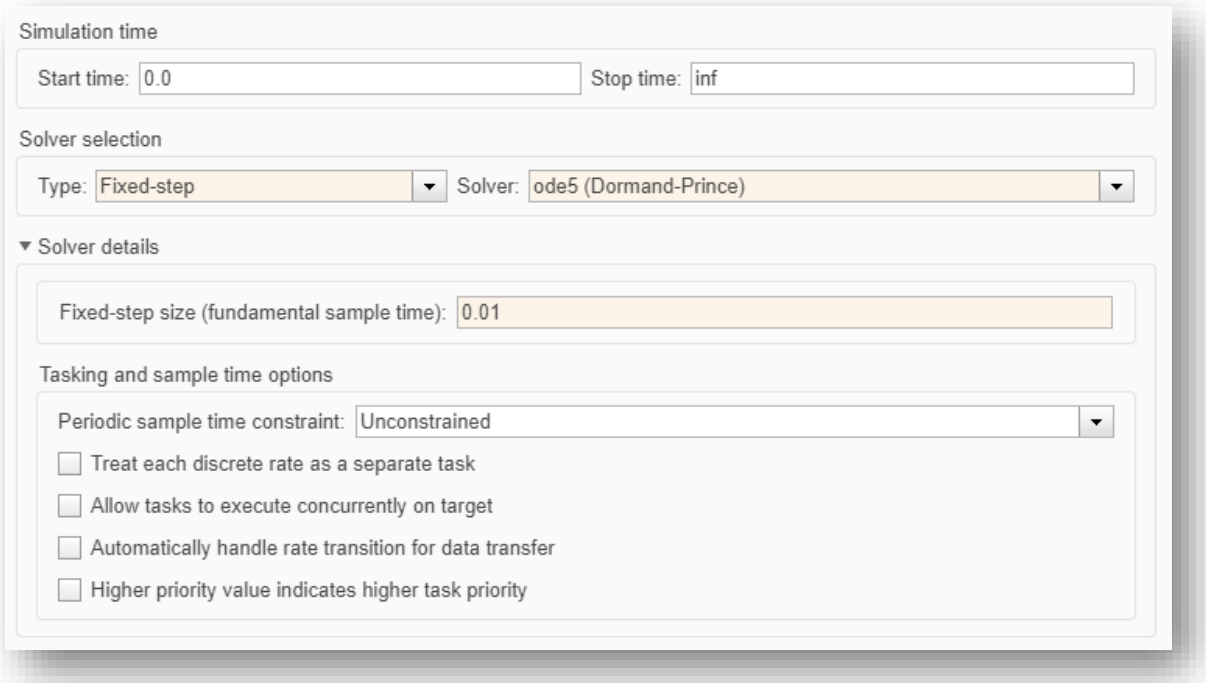
Fonctions Simulink®



L'explorateur du modèle donne son arborescence :



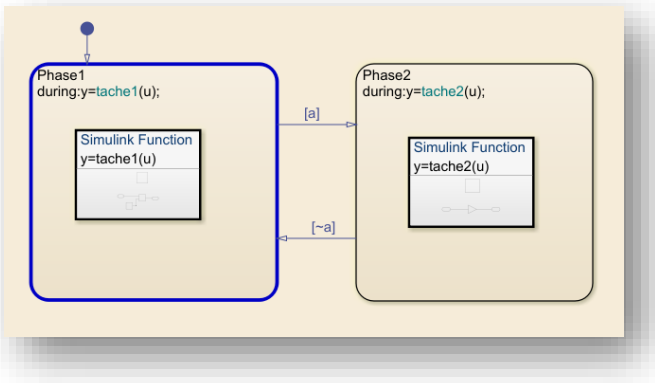

**Atttention :** Ici pour la simulation il convient de choisir un solveur autre que « **discrete (no continuous state)** » On pourra prendre par exemple le solveur « **ode5** » avec un pas fixe de 0.01 s et un temps de simulation infini:

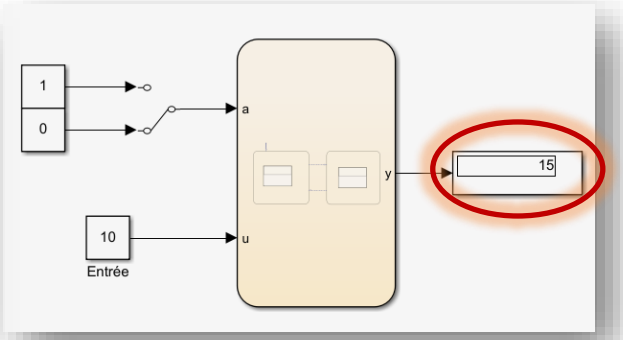



Pour exploiter pleinement les avantages de cet outil une bonne connaissance de Simulink® est indispensable.

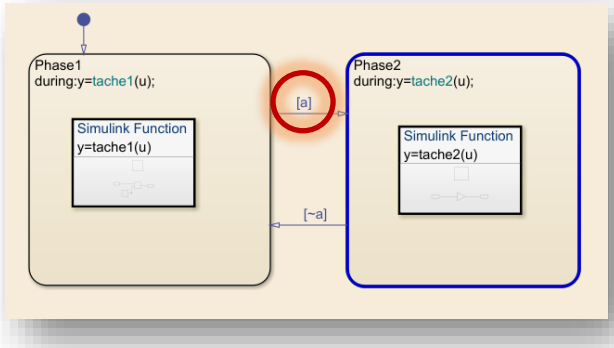
Vérifions les résultats obtenus par la simulation :

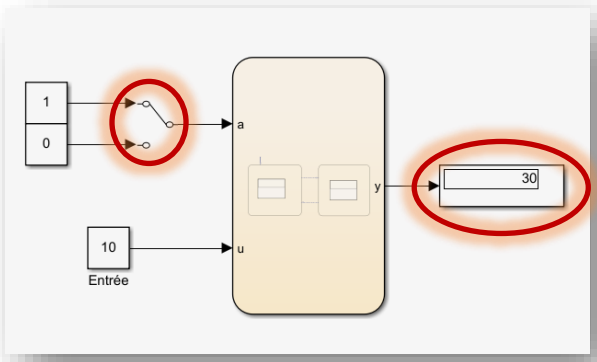
L'état « **Phase1** » est actif. Pendant toute la durée de son activité, on ajoute 5 à l'entrée. On obtient donc en sortie 15.



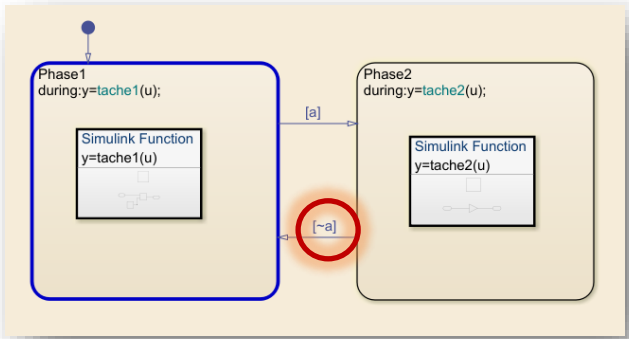



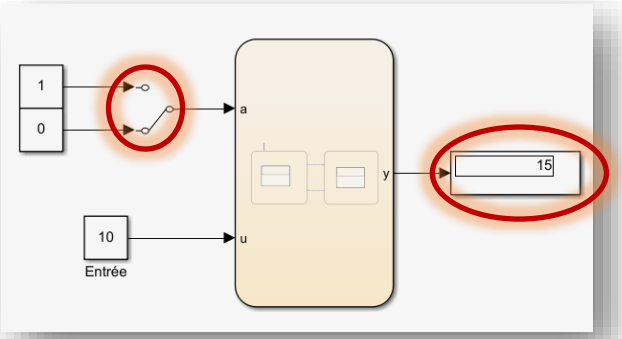

Si la condition [a] est vraie, l'état « **Phase2** » est actif et on multiplie par 3 l'entrée. On obtient 30 en sortie :



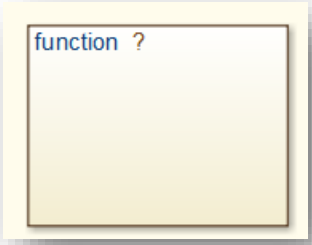

Enfin si la condition [~a] est vraie, on obtient à nouveau en sortie 15 :



## 4.8- GRAPHICAL FUNCTION

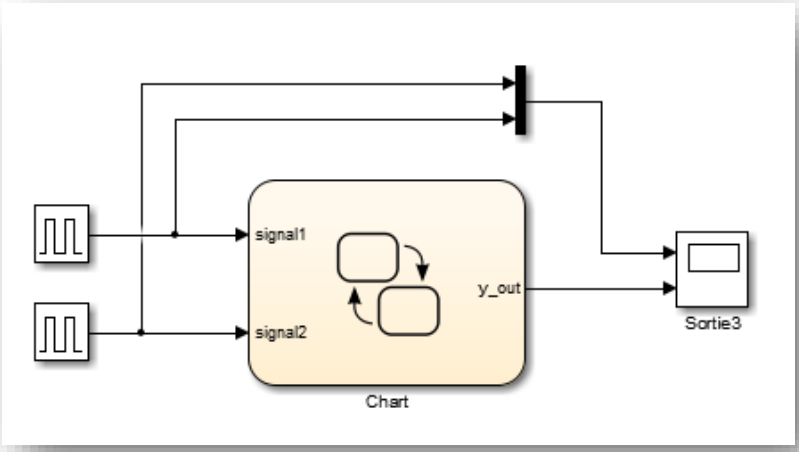
Comme son nom l'indique, cet outil va permettre la programmation de fonctions de manière graphique.



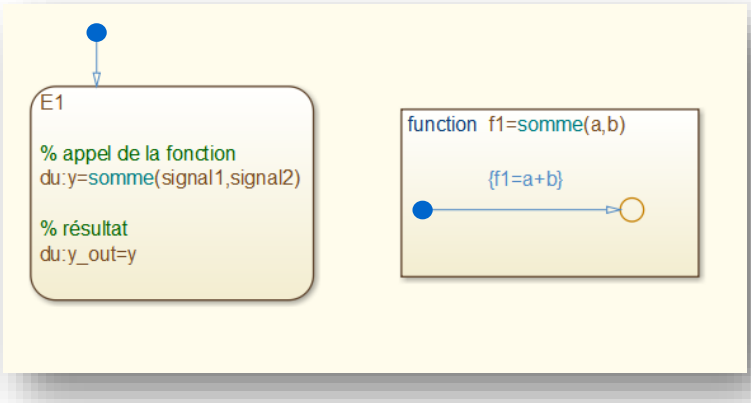

Une fonction graphique est représentée par un rectangle La programmation se fait en créant un diagramme de flux (« **Flow Chart** ») constitué uniquement de transitions et de jonctions.

En cliquant sur le point d'interrogation on précise la fonction à programmer qui sera appelée par l'état qui déclenchera le ou les calculs souhaités.

Illustrons la démarche de mise en œuvre de cet outil avec un exemple.




Nous souhaitons observer en sortie du « **Chart** » la somme de deux signaux carrés d'amplitude unitaire, de période 2 s et dont un des deux est déphasé d'un quart de période, soit 0.5 s.




Le contenu du « **Chart** » est composé d'un état (rectangle aux coins arrondis) et d'une fonction graphique (rectangle).

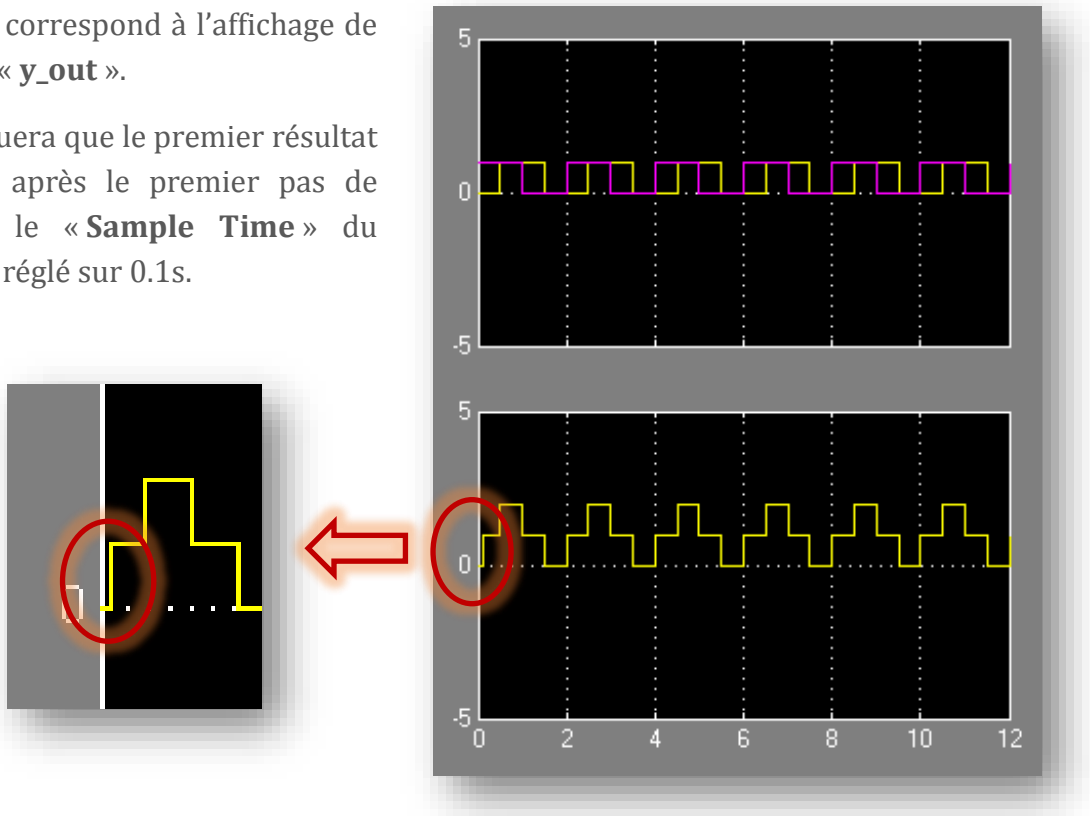
Comme nous l'avons vu précédemment pour la « **Simulink Function** », la signature de la fonction précise son nom, ses arguments et sa valeur de retour : « **somme** » est le nom de la fonction, « **a** » et « **b** » ses arguments et « **f1** » sa valeur de retour.

L'invocation de la fonction, ou son appel, se fait pendant l'activité de l'état E1.

Notons que la somme des arguments se fait ici sur une transition par défaut qui pointe une jonction. Le calcul est lancé à l'occurrence de l'évènement qui réveille la machine.

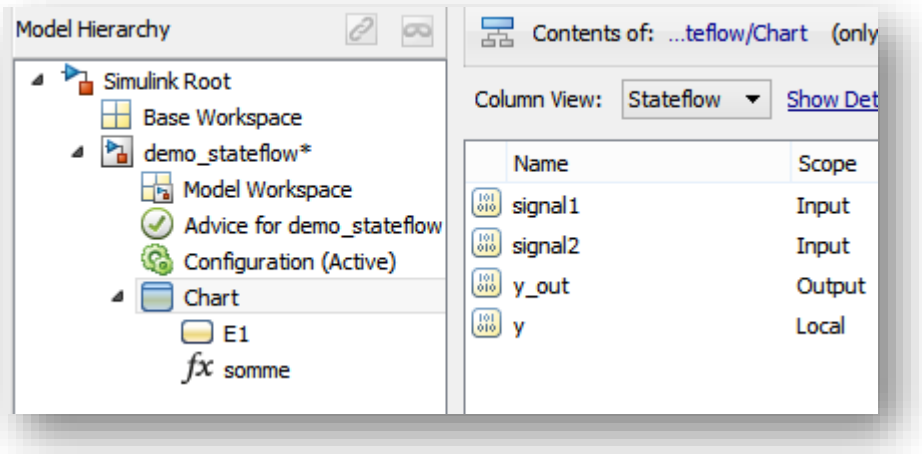
Le résultat correspond à l'affichage de la variable « **y_out** ».

On remarquera que le premier résultat est donné après le premier pas de calcul. Ici le « **Sample Time** » du solveur est réglé sur 0.1s.



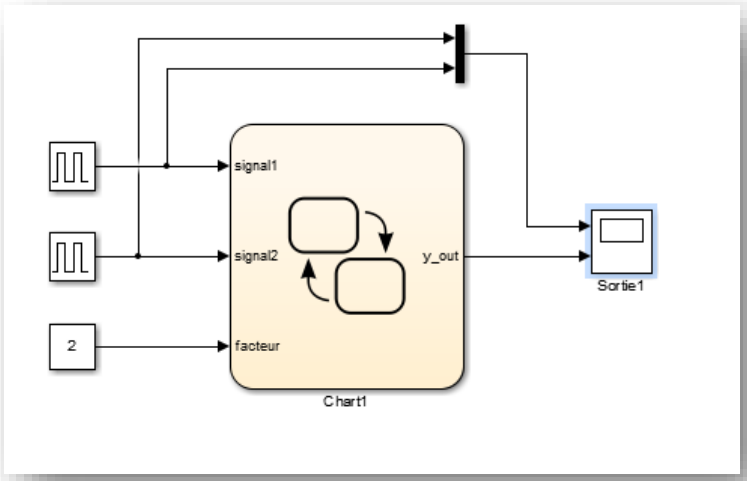

L'explorateur de modèle ci-dessous affiche dans la colonne de droite le nom de la fonction.

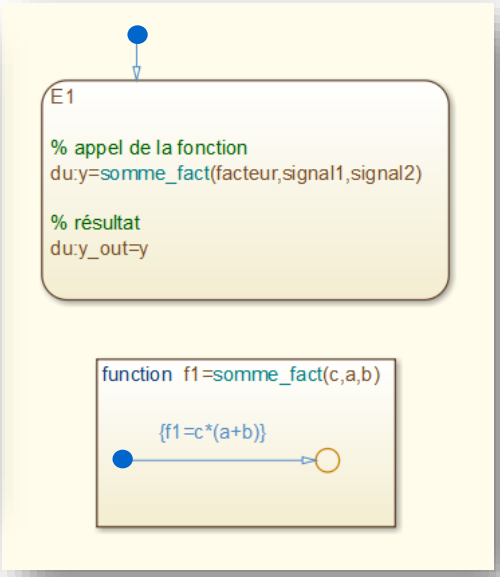
Les entrées et les sorties sont précisées ainsi que la variable locale « **y** ».



**Voici un petit exercice qui va enrichir le précédent** : on souhaite toujours faire la somme des mêmes signaux que précédemment mais cette somme sera multipliée par deux. Ce facteur sera donc une entrée du « **Chart** ».

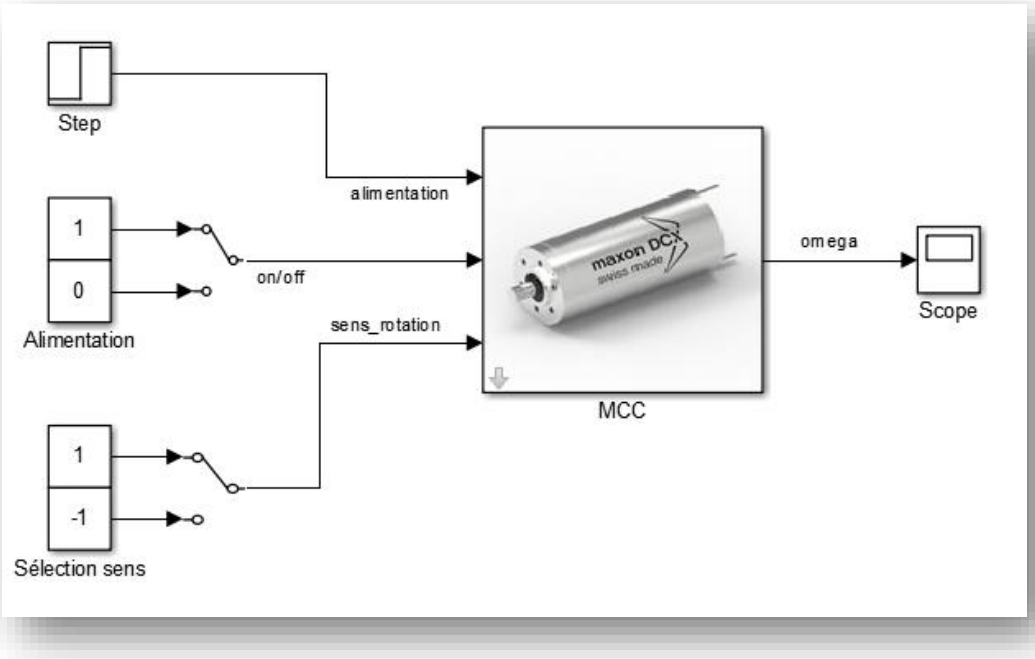
**Eléments de réponse :**



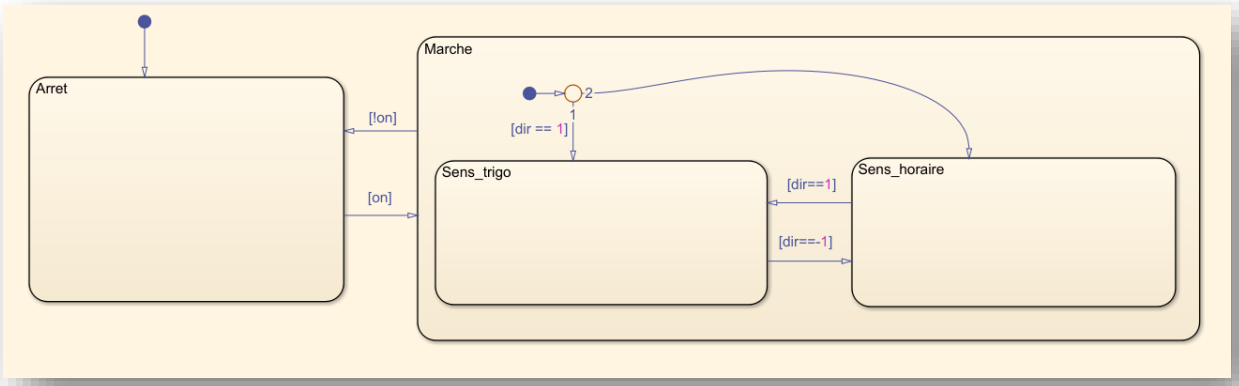

L'utilisation des fonctions graphiques dans un modèle Stateflow® permet aussi de simuler le comportement de systèmes continus. Prenons par exemple le cas d'un moteur à courant continu.

L'interface Simulink® que nous allons mettre en place est la suivante :




Le moteur sera sollicité par un échelon de tension. Un premier « **Switch** » permet d'inverser son sens de rotation, le second autorise ou pas l'alimentation du moteur.

La première étape consiste à tracer le diagramme d'états traduisant le comportement du moteur au cours de son fonctionnement :




La variable « **dir** » donne le sens de rotation du moteur et « **on** » autorise ou non la marche du moteur. Le moteur peut tourner dans un sens ou dans l'autre suivant l'état du « **Switch** » correspondant. Par conséquent dans l'état « **Marche** » la transition par défaut pointe une jonction qui va tester la valeur d'affectation de la variable « **dir** ».

Si « **dir** » vaut 1 le moteur tourne dans le sens trigonométrique sinon il tournera dans le sens horaire.

Maintenant il faut prendre en compte le comportement du moteur. Ecrivons alors le modèle par représentation d'état d'un moteur à courant continu.

**Petit rappel sur la représentation d'état et les équations du moteur à courant continu :**

D'un point de vue mathématique nous définirons des **variables d'entrée**, des **variables de sortie** et des **variables d'état**. Ces dernières sont **observables** à chaque instant et donc **mesurables**. Les valeurs des variables de sortie vont dépendre de celles des variables d'entrée et des variables d'état. Les équations liant ces variables sont appelées **équations de sortie**. Par contre, elles ne suffisent pas pour décrire le comportement dynamique du système. Cette dynamique correspond à la variation des variables d'état au cours du temps dépendante des variables d'entrée et des variables d'état à un instant antérieur. Dans le cas d'un comportement linéaire ces équations sont des équations différentielles. Elles s'écrivent de la manière suivante :

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases}$$

Où U est un vecteur caractéristique des variables d'entrée, X des variables d'état et Y des variables de sortie. A, B, C et D sont des matrices.

La première équation traduit la dynamique du système linéaire alors que la seconde donne l'évolution de la sortie.

Concernant le moteur à courant continu, on rappelle les quatre équations qui décrivent son comportement dynamique :

- La première issue de la loi d'Ohm :

$$u(t) = Ri(t) + L\frac{di(t)}{dt} + e(t)$$

- La seconde issue de l'application du théorème du moment dynamique au rotor :

$$C_m(t) - f\omega(t) = J\frac{d\omega(t)}{dt}$$

- La troisième et la quatrième qui correspondent aux équations de couplage électro-mécaniques :

$$C_m(t) = k_t i(t) \qquad \text{et} \qquad e(t) = k_e \omega(t)$$

  Ici, $u(t)$ est la tension aux bornes du moteur et $i(t)$ le courant qui le parcourt.

  $C_m(t)$ et $\omega(t)$ sont respectivement le couple moteur et la vitesse de rotation du rotor.

  $R$ est la résistance de l'induit et L l'inductance.
  $f$ est le coefficient de frottement visqueux.

$k_t$ et $k_e$ sont respectivement la constante de couple et la constante de force électromotrice.

Nous allons remanier les équations précédentes pour se ramener à la représentation d'état rappelée quelques lignes plus hautes. On trouve :

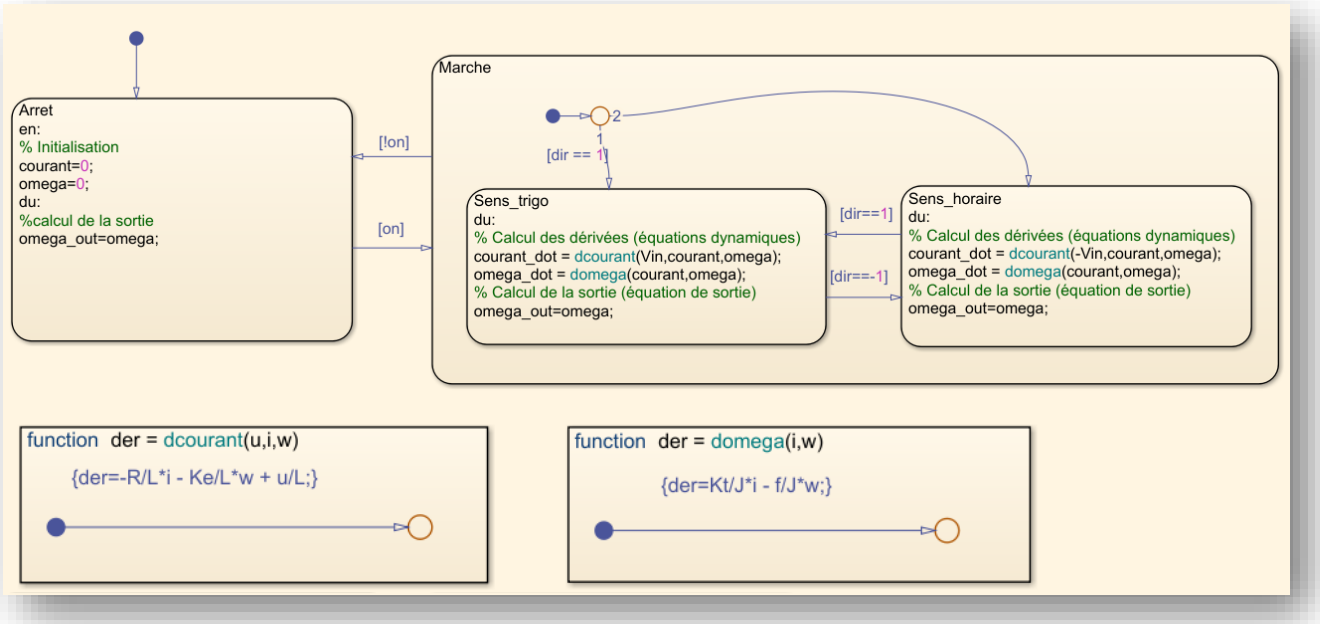
$$\begin{pmatrix} \frac{di(t)}{dt} \\ \frac{d\omega(t)}{dt} \end{pmatrix} = \begin{pmatrix} -\frac{R}{L} & -\frac{k_e}{L} \\ \frac{k_t}{J} & -\frac{f}{J} \end{pmatrix} \begin{pmatrix} i(t) \\ \omega(t) \end{pmatrix} + \begin{pmatrix} \frac{1}{L} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} u(t) \\ 0 \end{pmatrix}$$

Et

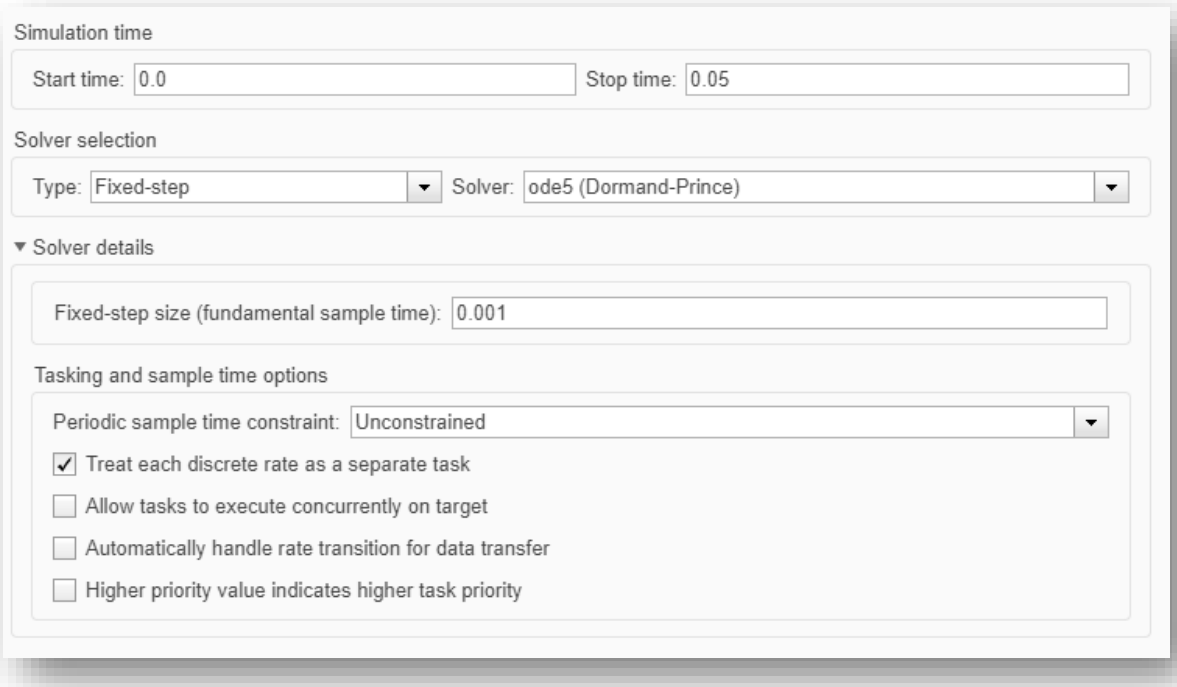
$$\omega(t) = (0\ 1) \begin{pmatrix} i(t) \\ \omega(t) \end{pmatrix} + (0\ 0) \begin{pmatrix} u(t) \\ 0 \end{pmatrix}$$

Nous allons utiliser ces équations dans la machine d'état. Pour ce faire il faudra insérer des fonctions graphiques (« **Graphical Function** ») qui seront appelées par les états correspondants (« **Marche.Sens_trigo** » et « **Marche.Sens_horaire** »).

On obtient donc le diagramme complété suivant:



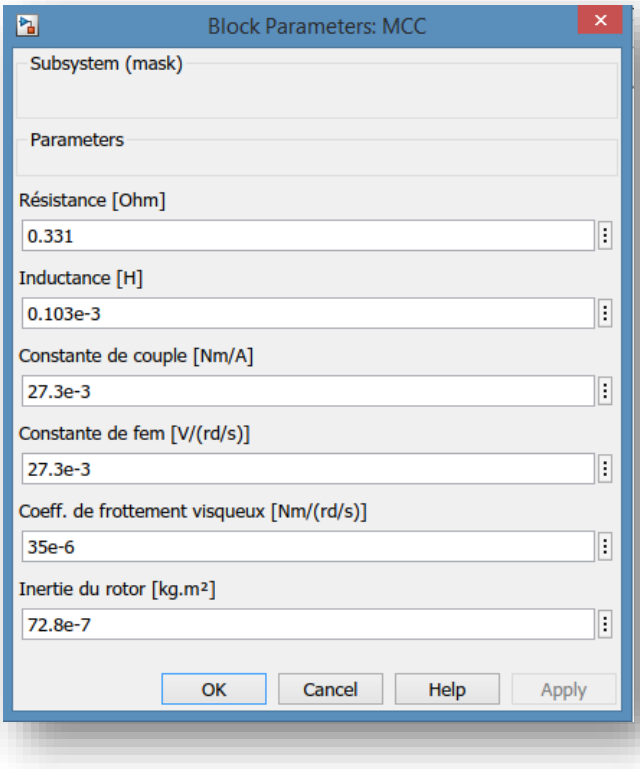
**Attention** : pour la simulation il faut changer de solveur car nous simulons le comportement d'un système continu.
Le solveur : « **discrete** » ne convient plus. On pourra prendre par exemple le solveur « **ode5** » avec un pas fixe de 0.001s :



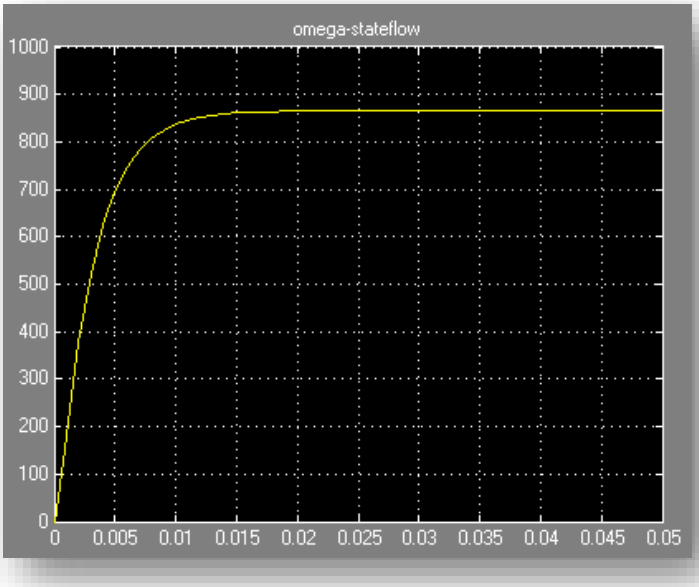

Prenons par exemple, le cas du moteur Maxxon DCX32 dont voici les caractéristiques (http://www.maxonmotor.com/maxon/view/catalog/):

| General data | |
|---|---|
| Status | provisional |
| **Values at nominal voltage** | |
| Nominal voltage | 24 V |
| No load speed | 8270 rpm |
| No load current | 164 mA |
| Nominal speed | 7700 rpm |
| Nominal torque (max. continuous torque) | 112 mNm |
| Nominal current (max. continuous current) | 4.26 A |
| Stall torque | 1980 mNm |
| Stall current | 72.5 A |
| Max. efficiency | 89 % |
| **Characteristics** | |
| Terminal resistance | 0.331 Ω |
| Terminal inductance | 0.103 mH |
| Torque constant | 27.3 mNm/A |
| Speed constant | 350 rpm/V |
| Speed / torque gradient | 4.24 rpm/mNm |
| Mechanical time constant | 3.24 ms |
| Rotor inertia | 72.8 gcm² |

La création d'un masque permet de saisir les caractéristiques du moteur choisi :




Pour un échelon de tension de 24V nous obtenons le résultat suivant:



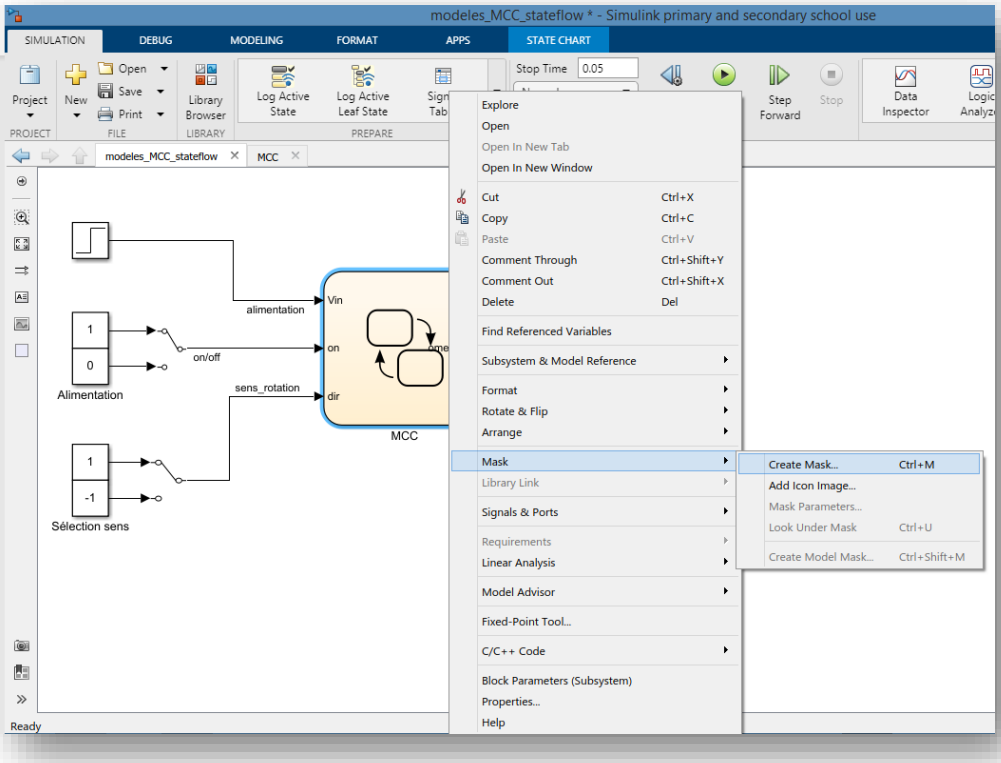

On retrouve bien les données du constructeur :

- en régime permanent le moteur atteint une vitesse de 866 rd/s soit 8270 tr/min.
- La constante de temps est de 3.2 ms.

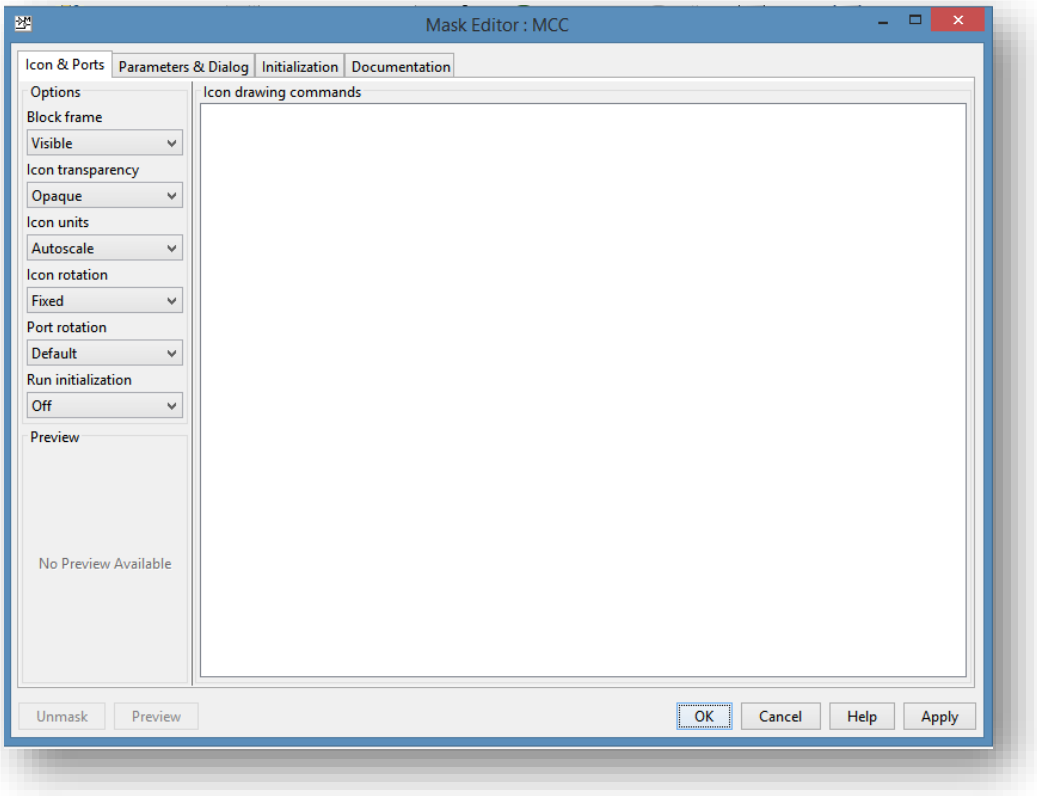
**Comment créer un masque :**

L'avantage du masque est l'éventuelle réutilisation du modèle mais pour un moteur qui possède des caractéristiques différentes.

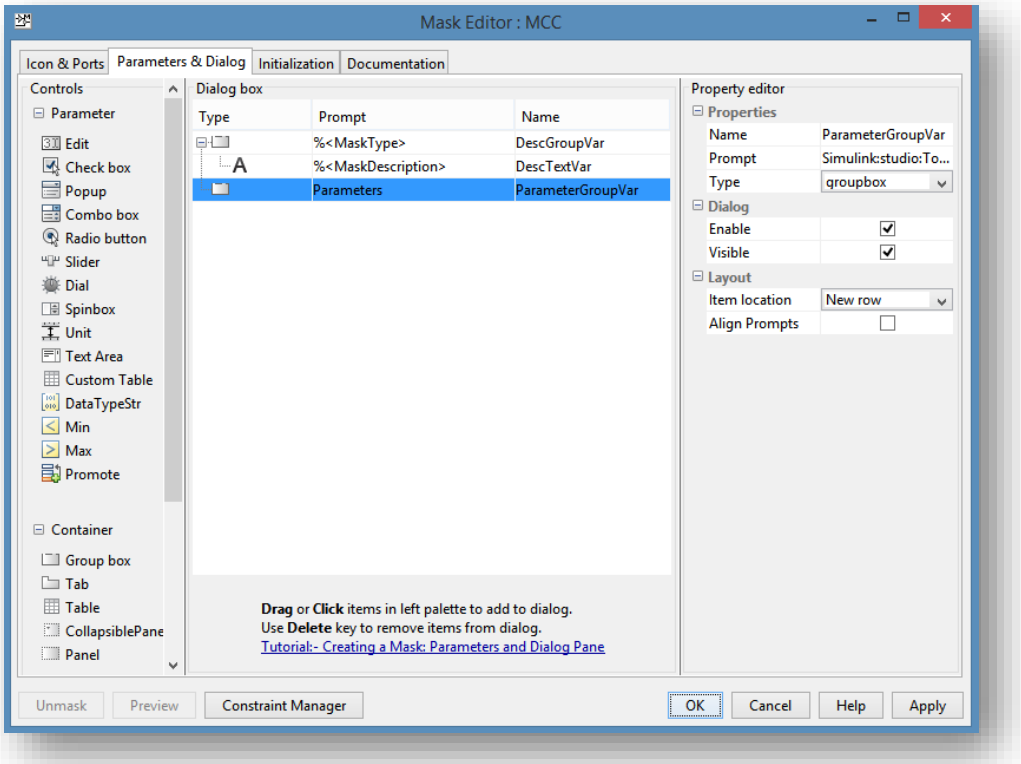
Sélectionner le « **Chart** », cliquer avec le bouton droit de la souris pour ouvrir le menu contextuel et sélectionner successivement les commandes « **Mask** » puis « **Create Mask** » :



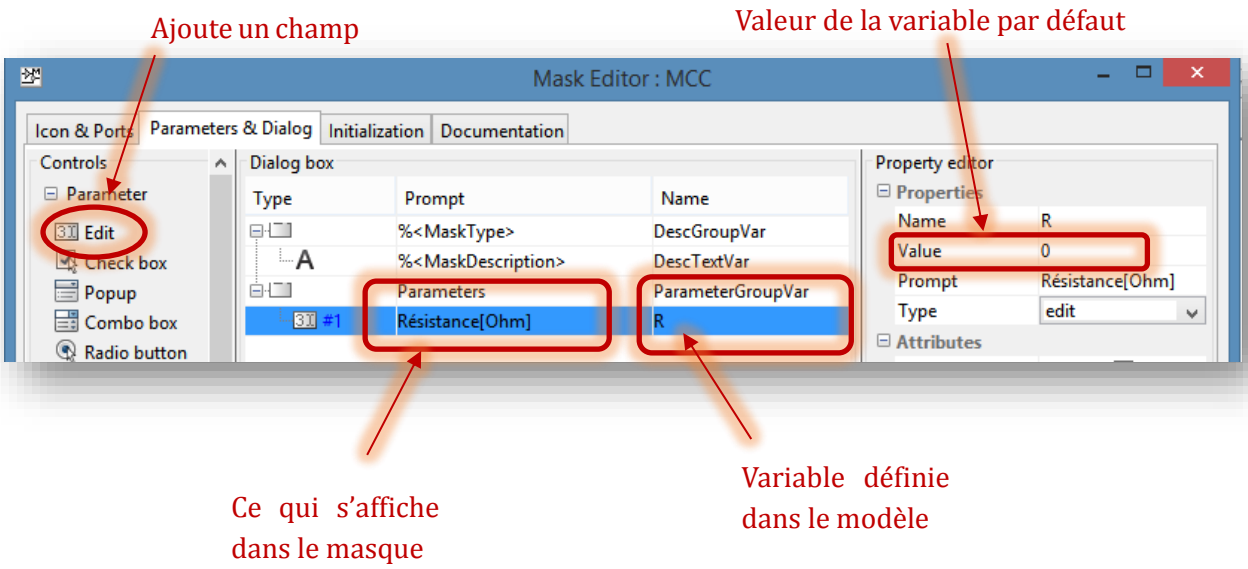
L'éditeur de masque s'affiche :



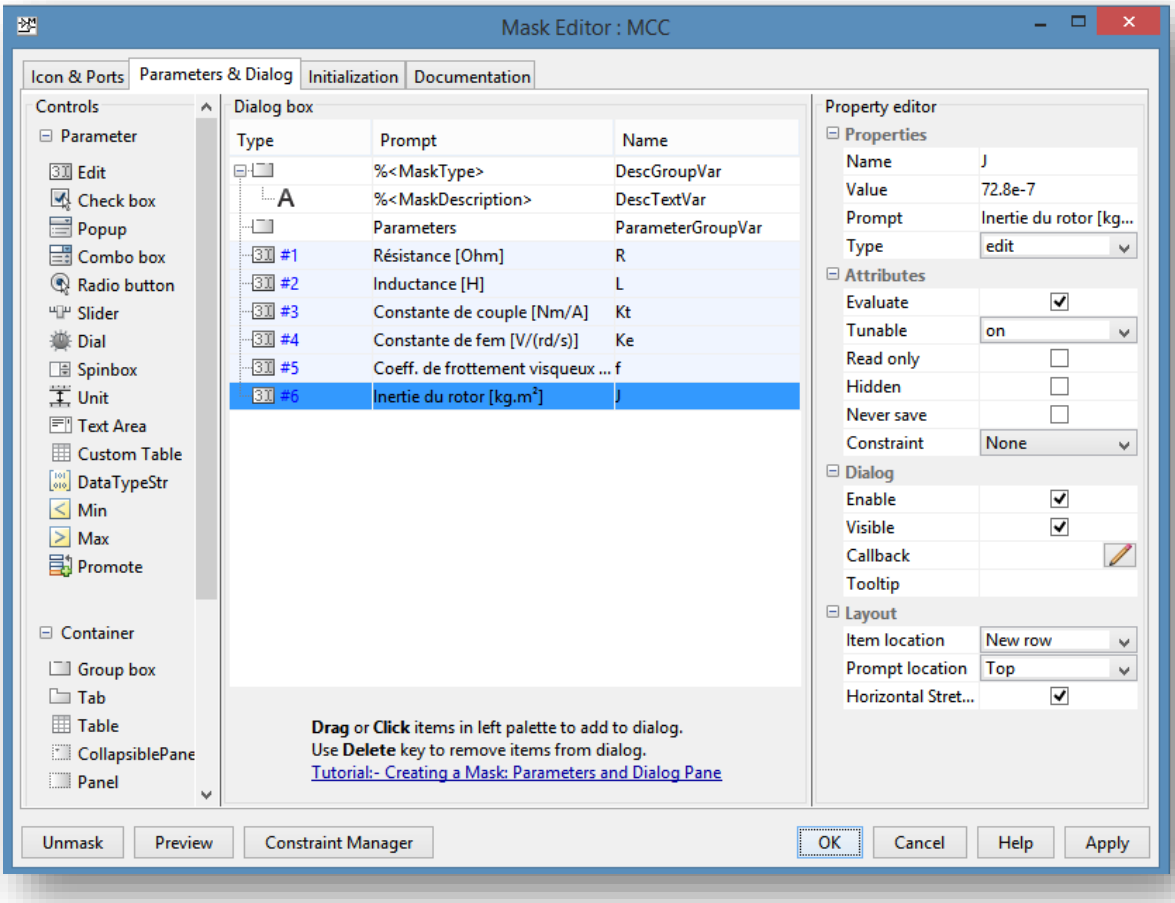

Sélectionner l'onglet « **Parameters & Dialog** » :



Saisir un premier champ du masque en cliquant sur l'icône « **Edit** » du panneau de contrôle :

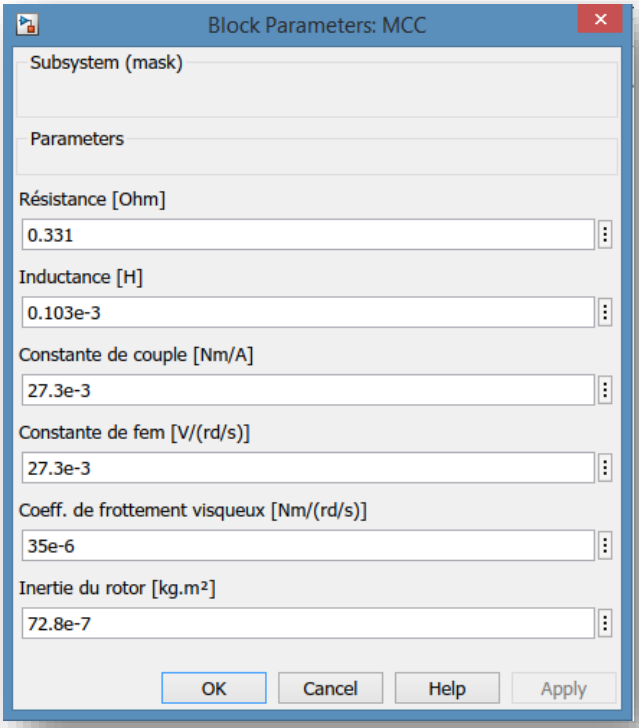



Les différents champs du masque sont créés en cliquant successivement sur « **Edit** ».

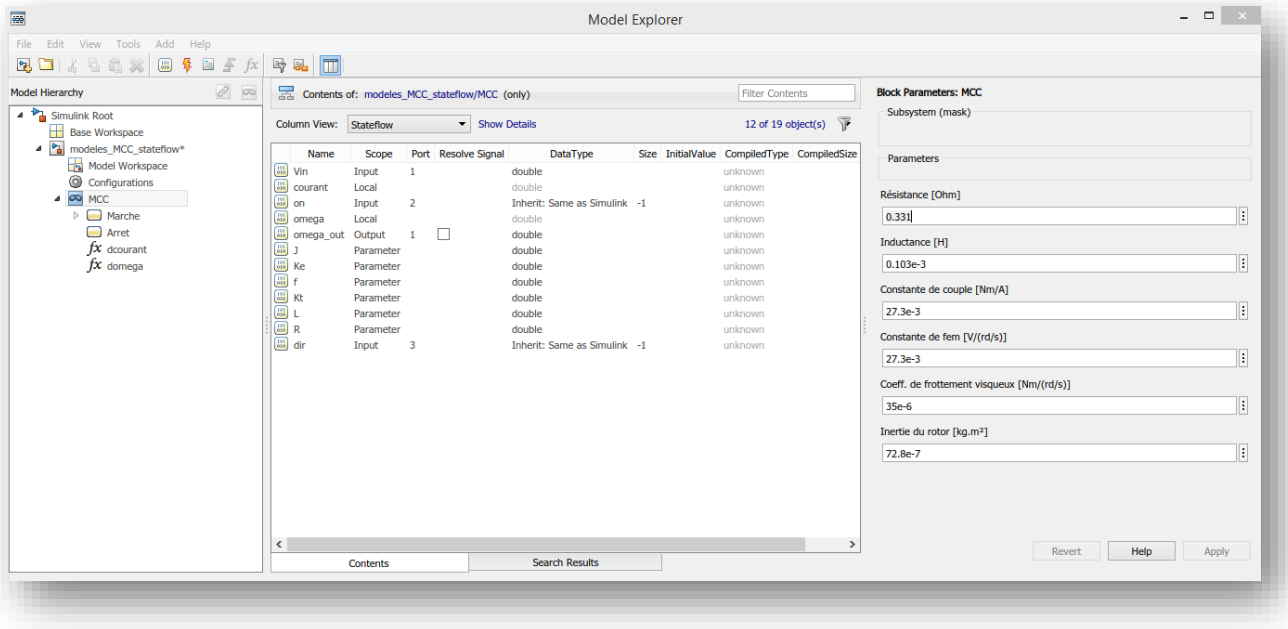



L'édition du masque est terminée lorsque les six champs de notre exemple sont déclarés.

En cliquant sur « **OK** » le masque est en place.

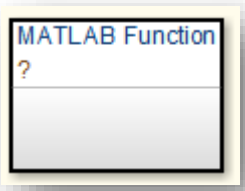



L'explorateur de modèle affiche une vision d'ensemble du modèle ainsi réalisé :



## 4.9 MATLAB FUNCTION

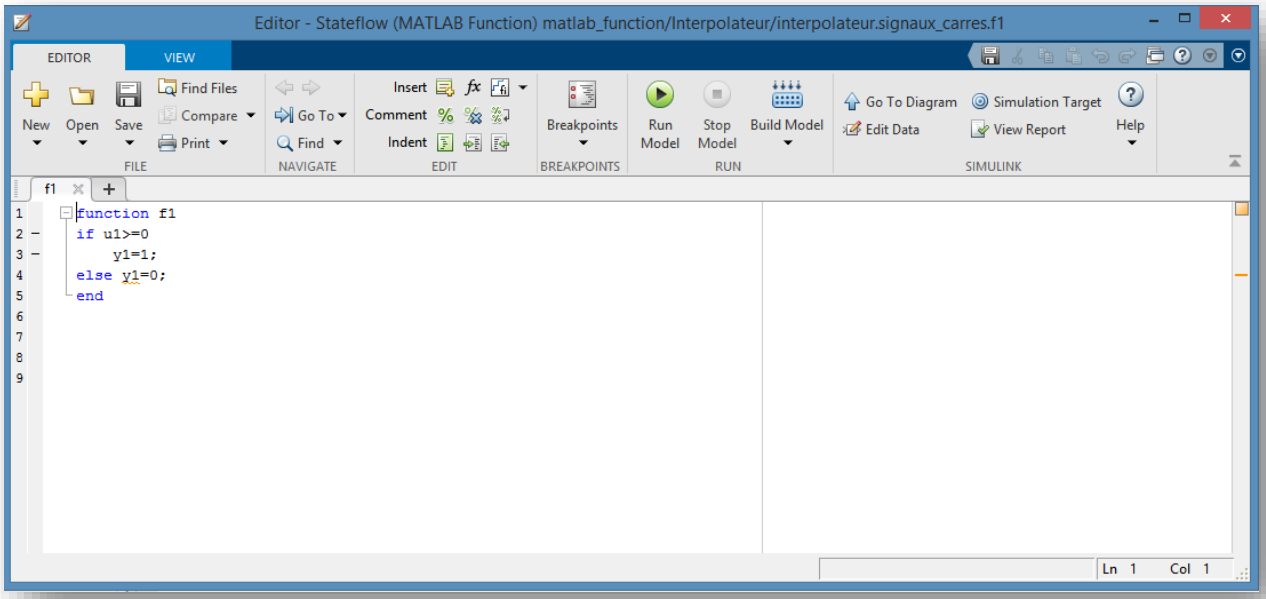
Comme nous venons de le voir dans le paragraphe précédent, il est possible dans un « **Chart** » de faire appel à une fonction graphique.




Nous allons voir maintenant qu'il est aussi possible d'appeler une fonction directement programmée en langage MATLAB.

Ces fonctions sont appelées « **Embedded MATLAB Functions** ». Elles sont représentées par un rectangle.

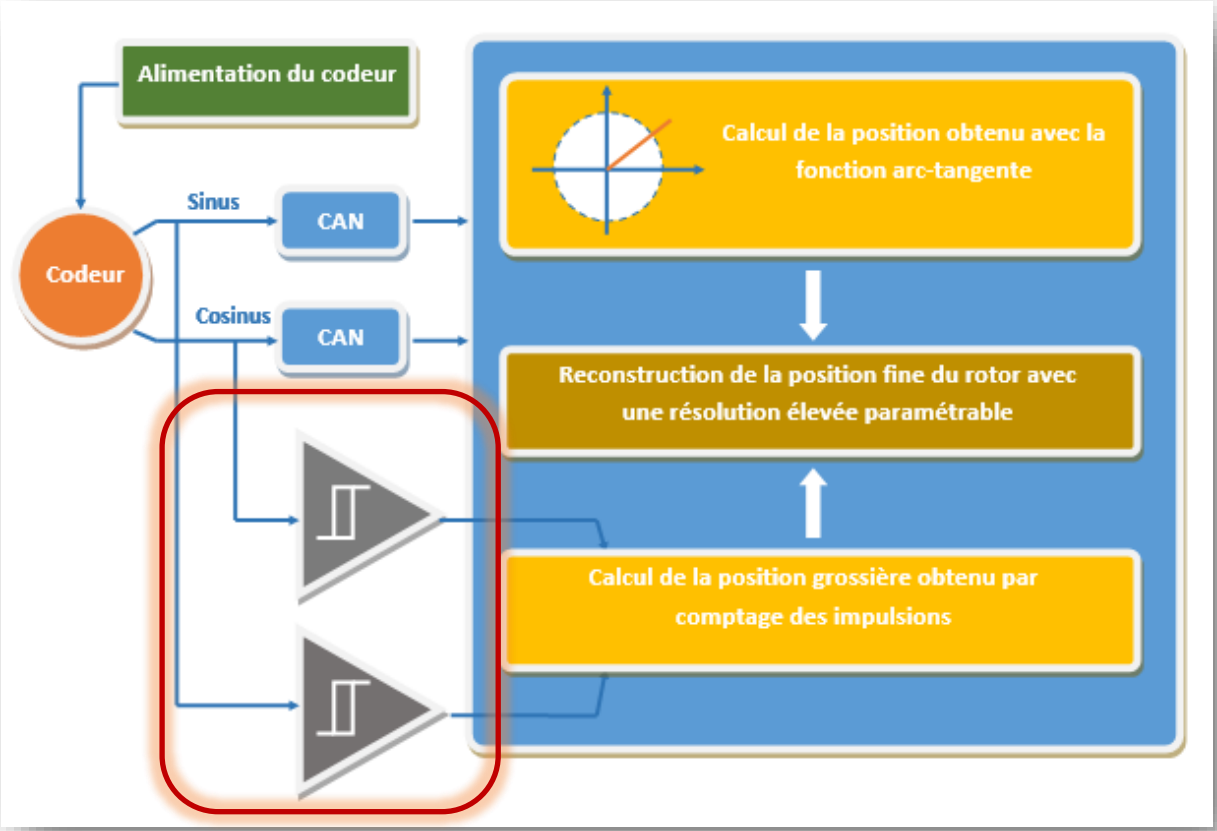
Un double-clic sur le bloc permet d'ouvrir l'éditeur (« **Editor** ») de fonction dans lequel il faut programmer la fonction désirée. Ici la fonction f1 dont on reparlera dans les lignes qui suivent :




Illustrons la méthode de mise en œuvre de ces fonctions à partir de l'exemple du traitement des signaux sinusoïdaux d'un codeur SinCos Hyperface®.

Ces capteurs optiques analogiques équipent les servomoteurs à dynamique élevée pouvant donc évoluer aussi bien à vitesse très lente qu'à vitesse très élevée.

L'information image de la position est constamment disponible. La valeur de la position est obtenue par le calcul numérique de la fonction arc-tangente après conversion des signaux sinus et cosinus. Ce calcul consiste en une interpolation fine dépendante de la position grossière du rotor :



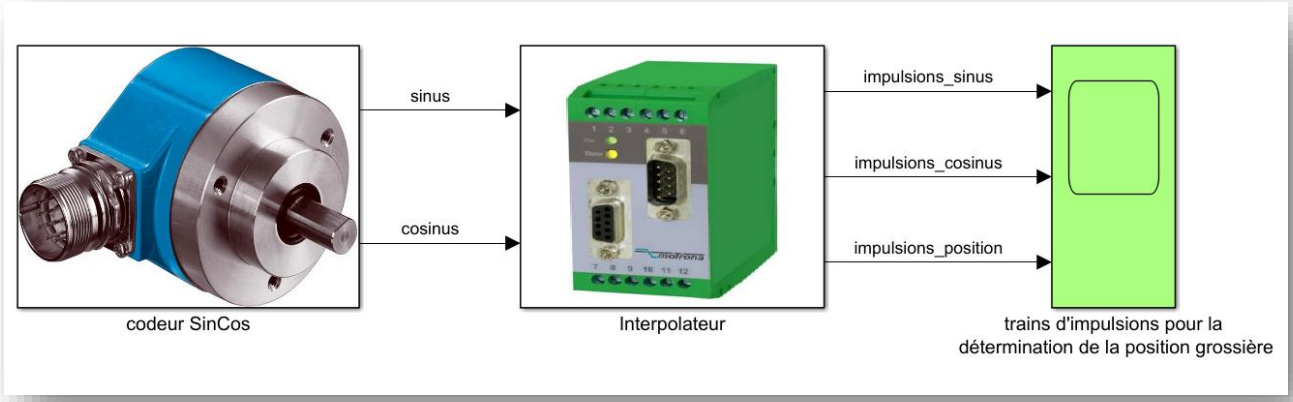

La position grossière du rotor correspond à sa position angulaire déterminée par le comptage d'impulsions de la même manière que pour un codeur incrémental.

L'objet de cette application est la mise en forme des trains d'impulsions nécessaires à la détermination de la position grossière.

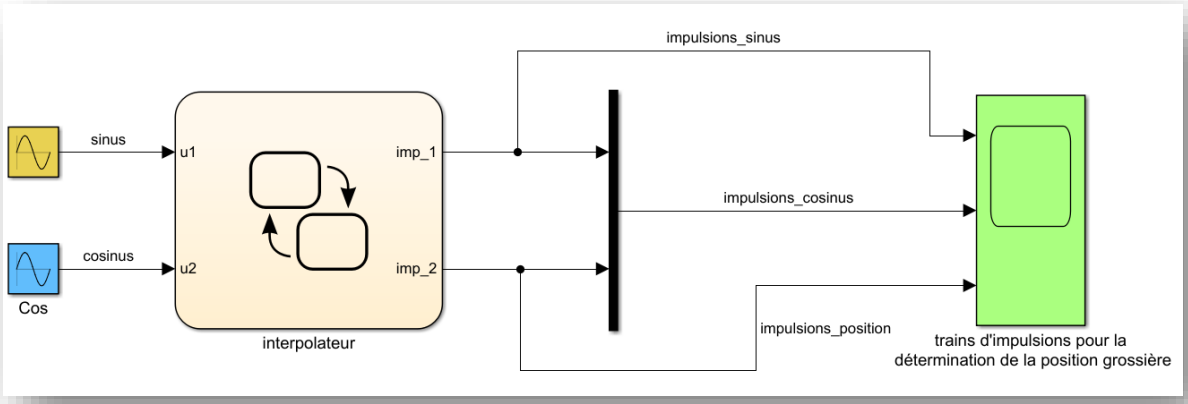
Pour ce faire nous allons construire un « **Chart** » comprenant quatre fonctions MATLAB.

Préparons l'interface Simulink®. Elle comprend deux éléments principaux :

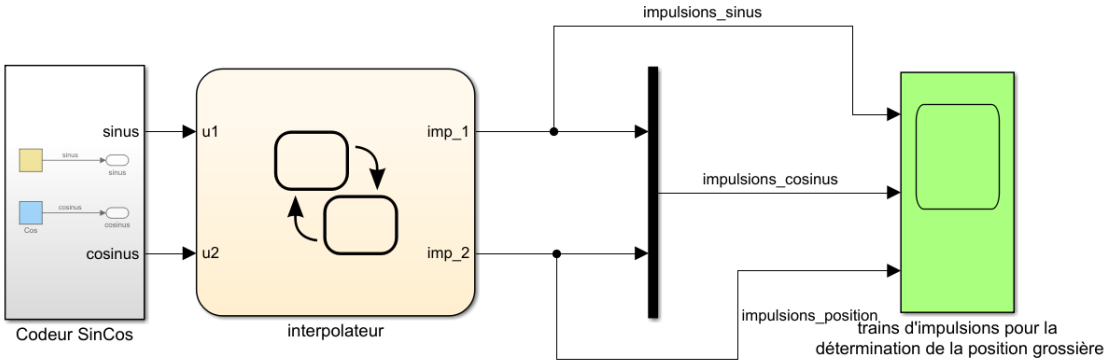
- Le codeur qui délivre les signaux sinus et cosinus,
- L'interpolateur qui génère le train d'impulsions nécessaire.



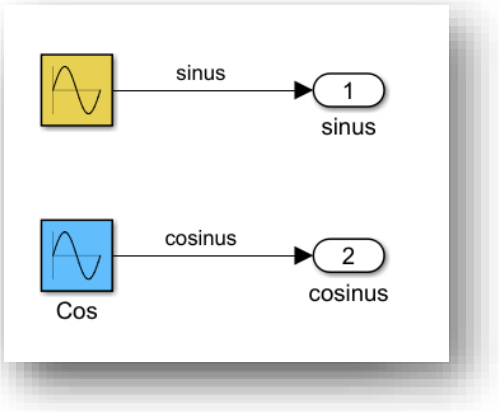

Mettons en place dans un premier temps l'interpolateur qui reçoit les signaux sinus et cosinus du codeur :



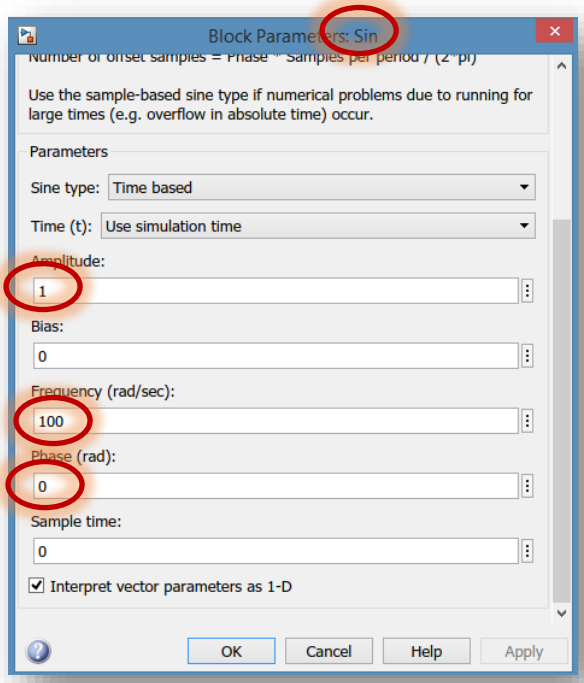

Pour mettre en place le bloc qui modélise le codeur nous allons créer un sous-système. Sélectionner les sources sinusoïdales, clic droit et « **Create Subsystem from Selection** » . On obtient une interface dont la forme est la suivante :

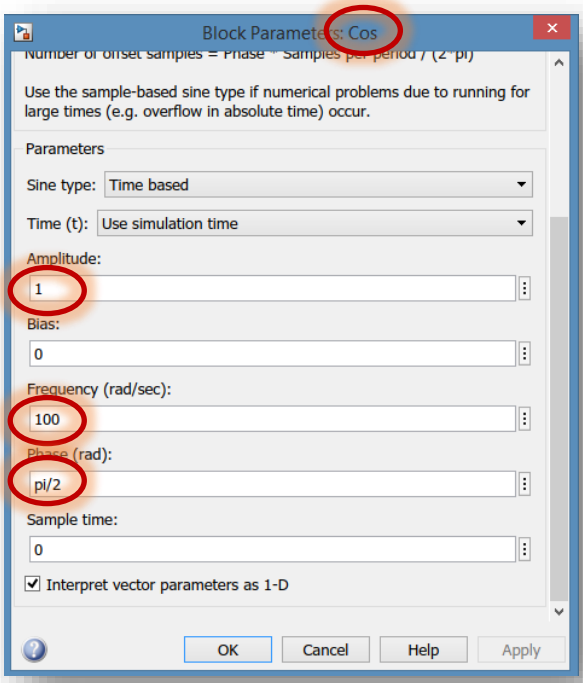


Un double clic sur le bloc « **CodeurSinCos** » fait apparaître les signaux préalablement saisis que nous allons configurer. On remarque que ces signaux apparaissent en transparence dans le bloc associé au sous-système.

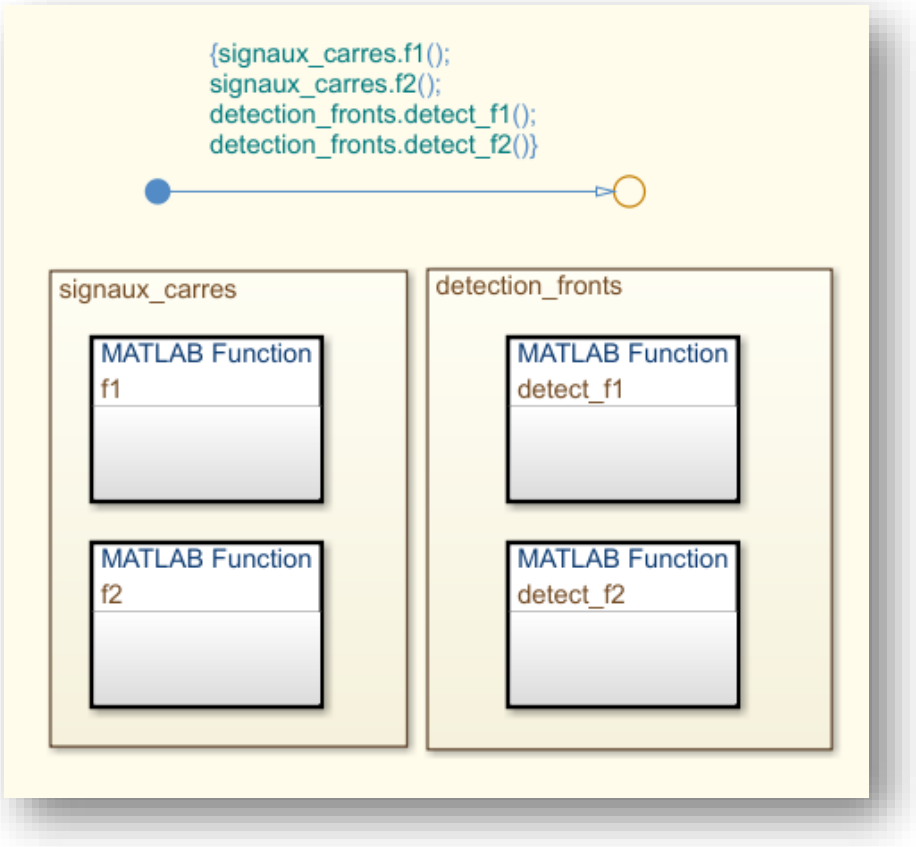



On considèrera dans l'exemple traité, un signal d'amplitude l'unité et de pulsation 100 rad/sec.




Un double-clic sur le « **Chart** » fait apparaitre le diagramme (déjà vu précédemment lorsque nous avons défini la « **Box** ») que nous allons maintenant construire et commenter :



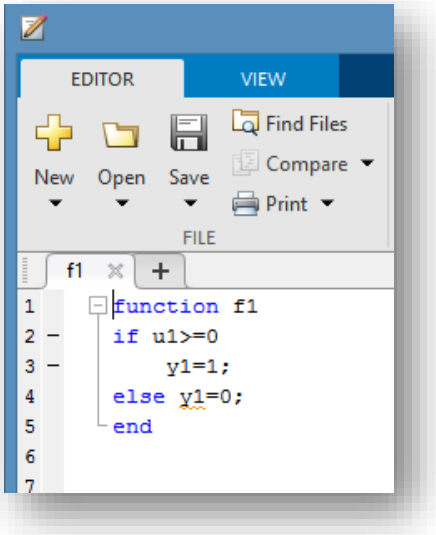

Le diagramme comporte une transition par défaut pointant vers une jonction. L'action associée fait appel à quatre fonctions MATLAB insérées dans deux « **Boxes** ». Attention a bien indiqué le chemin des fonctions en précisant le « label » de la « Box » qui la contient :

Dans la « **Box** » « **signaux_carres** » se trouvent les fonctions « **f1** » et « **f2** » qui vont construire les signaux carrés de même période que les signaux sinusoïdaux d'entrée.

Dans la « **Box** » « **detection_fronts** » se trouvent les fonctions « **detect_f1** » et « **detect_f2** »qui vont générer les trains d'impulsions qui correspondent aux changements de niveau logique des signaux carrés

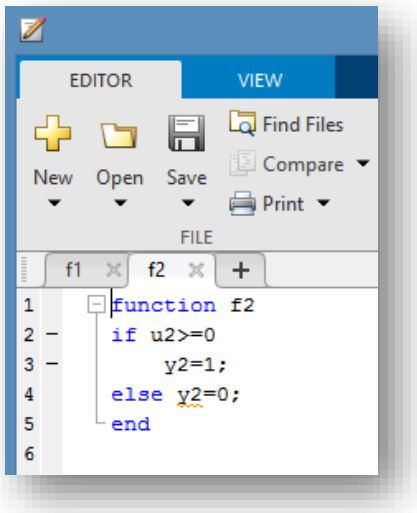
Les fonctions « **f1** » et « **f2** » s'appliquant aux deux entrées sinusoïdales sont similaires. De même que « **detect_f1** » et « **detect_f2** ».

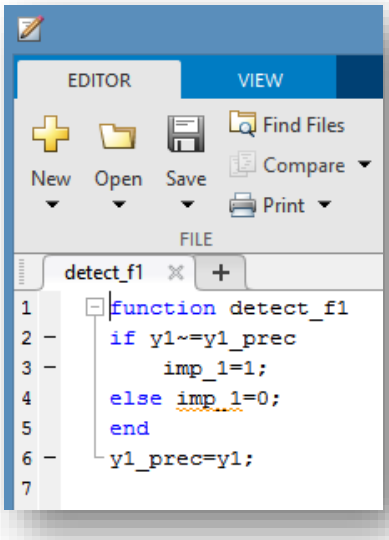
Un double-clic sur la fonction « **f1** » ouvre l'éditeur de fonction :




La fonction programmée affecte la valeur 1 à la variable locale « **y1** » si l'entrée « **u1** »est positive sinon la valeur 0.

De même pour « **f2** » :

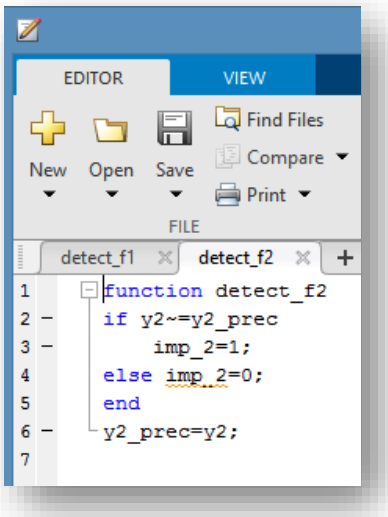


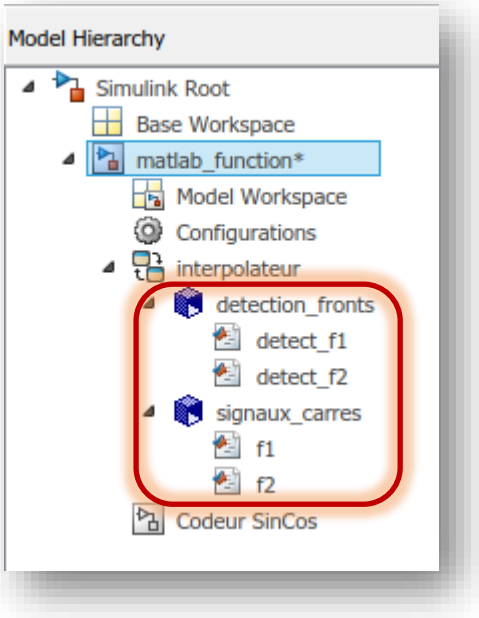




Concernant la détection des fronts, un double-clic sur la fonction « **detect_f1** » ouvre l'éditeur :

Si la valeur de la variable locale « **y1** » est différente de sa valeur au pas de calcul précédent alors la sortie « **imp_1** » prend la valeur 1 sinon 0. Après le test la variable locale « **y1_prec** » prend la valeur de la variable locale « **y1** ».

De même pour la fonction « **detect_f2** » :



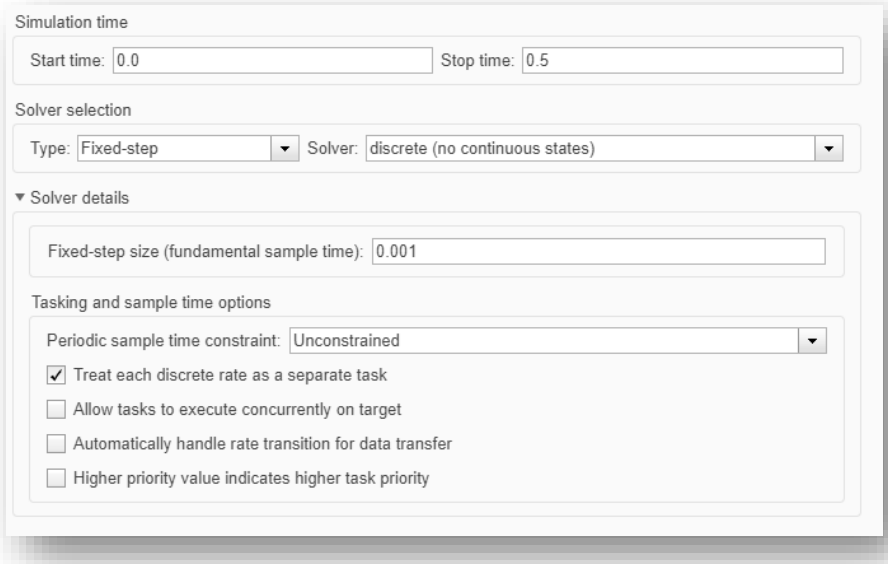



L'explorateur donne la hiérarchie du modèle construit. On y trouve :

- Les « **Boxes** »,
- Les fonctions MATLAB,

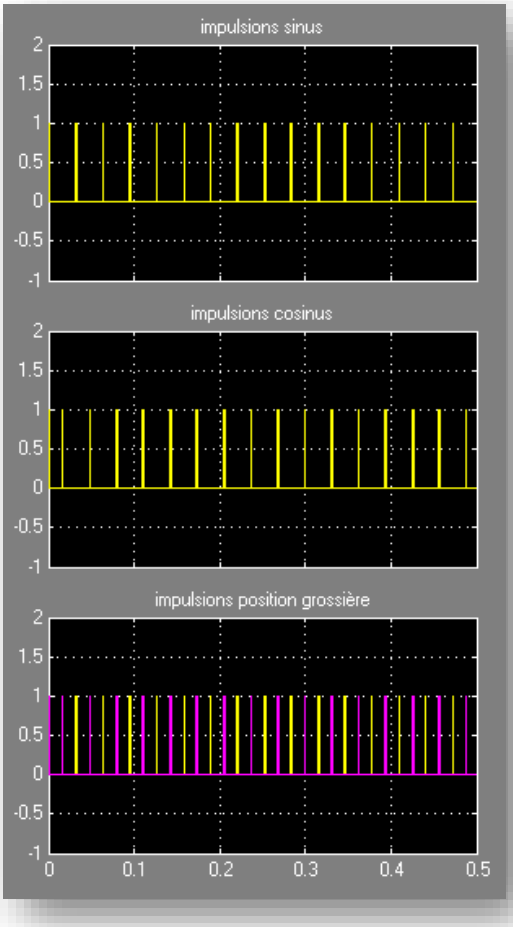
On y trouve aussi les caractéristiques des différentes variables :

| Name | Scope | Port | Resolve Signal | DataType |
|---|---|---|---|---|
| imp_1 | Output | 1 | ☐ | double |
| imp_2 | Output | 2 | ☐ | double |
| y1 | Local | | ☐ | double |
| y2 | Local | | ☐ | double |
| y1_prec | Local | | ☐ | double |
| y2_old | Local | | ☐ | double |
| u1 | Input | 1 | | double |
| u2 | Input | 2 | | double |
| detection_fronts | | | | |
| signaux_carres | | | | |

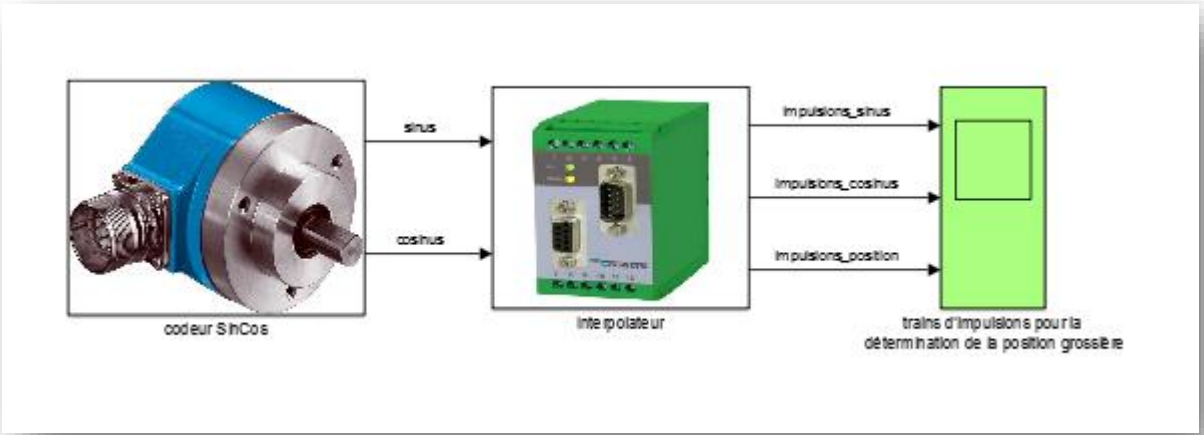



Pour cette application nous avons choisi le solveur « **discrete (no continuous states)** » avec un pas fixe de 0.001 s:La simulation sur 0.5 secondes affiche les résultats qui suivent.

On retrouve le train d'impulsions similaire à celui qu'on aurait obtenu avec un codeur incrémental.

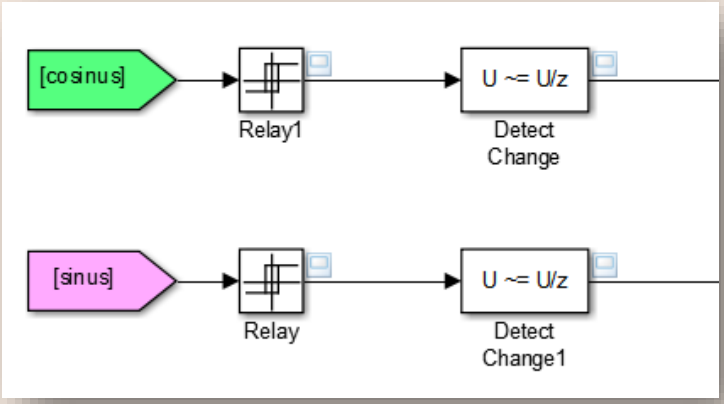



La présentation de l'ensemble peut être améliorée en créant un masque qui intègre une image dans les blocs qui modélisent le codeur et l'interpolateur (voir la mise en œuvre dans l'exemple du pont roulant au chapitre 5 consacré à la table de vérité) :

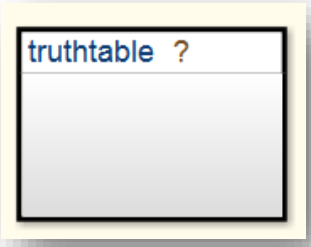


**Remarque** : un modèle équivalent pourrait être construit sous Simulink®. En effet les blocs dont nous venons de programmer la logique existent :

- le bloc « **Relay** » pour construire le signal carré (bibliothèque « **Discontinuities** »),
- le bloc « **Detect Change** » pour en extraire les fronts (bibliothèque « **Logic and Bit Operations** ».



## 4.10 TRUTH TABLE




Stateflow® offre la possibilité d'insérer dans un modèle une table de vérité qui va traduire le comportement **combinatoire** d'un système ou d'un sous-système. Cette table de vérité peut simplifier considérablement les diagrammes.

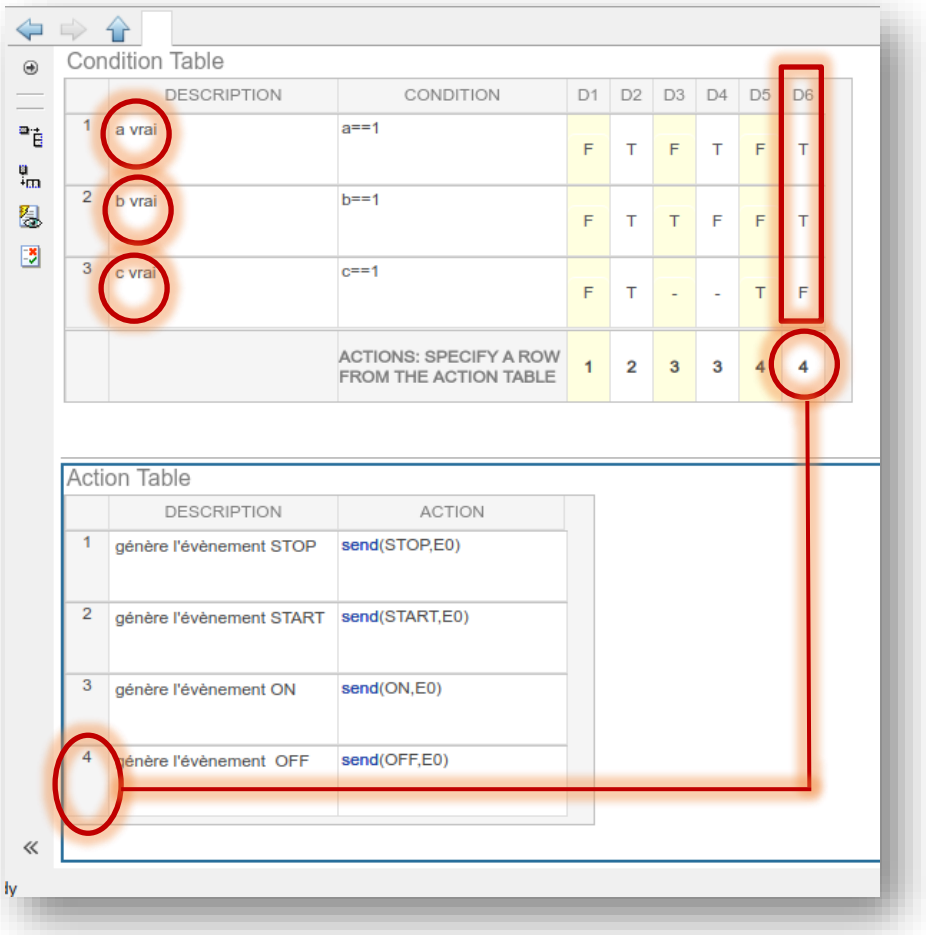
Une table de vérité est représentée par un rectangle.

Un double-clic sur ce rectangle ouvre une fenêtre dans laquelle apparaissent deux tableaux : le tableau des conditions (« **Condition Table** ») et le tableau des actions (« **Action Table** »).

Dans ces tables nous trouvons : des conditions, des décisions et des actions :


Chaque table possède aussi une colonne « **Description** ». Elle est destinée aux commentaires. Compléter ces champs n'est pas obligatoire.

Dans la seconde colonne de la « **Condition Table** » apparaissent les conditions auxquelles sont associées des actions. La condition peut être vraie (« **True :T** »), fausse (« **False :F** »), vraie ou fausse (' **-** ').

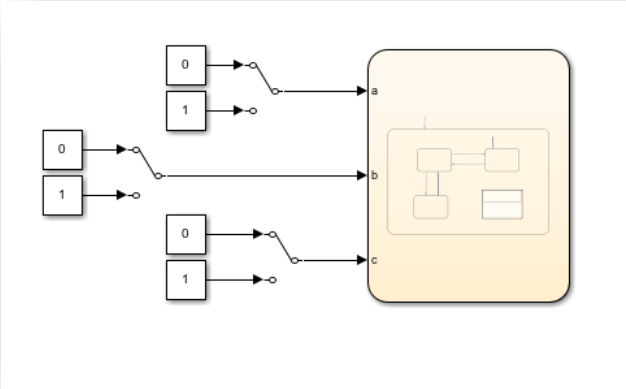



Une « **Decision** » (Di) correspond à une colonne dans laquelle on prend en compte l'état des conditions. A chaque décision correspond une action repérée par le numéro de ligne de l' « **Action Table** » liée à cette action.

Si nous prenons l'exemple ci-dessus, la décision D6 considère (a_vraie ET b_vraie ET c_fausse). Dans ce cas l'action associée est définie à la quatrième ligne de l' « **Action Table** » : send(OFF,E0), c'est-à-dire qu'on déclenche l'évènement OFF qui se trouve dans l'état E0 etc…

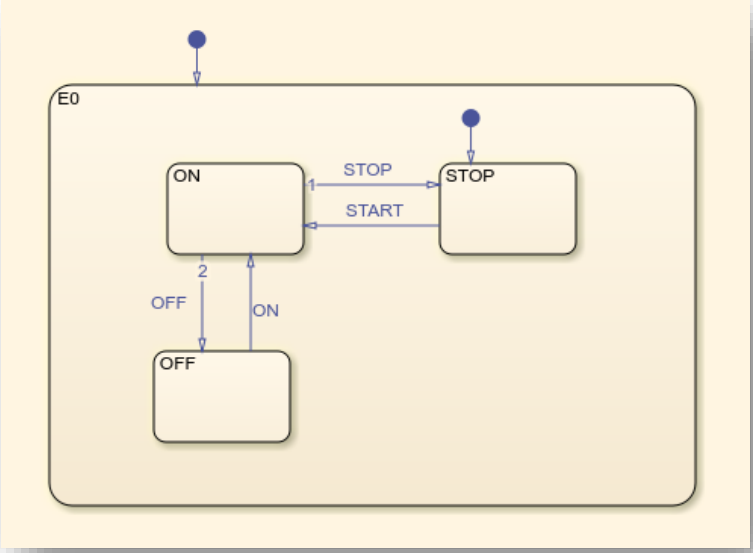
**Une première application…**

Développons davantage l'exemple présenté plus haut.




Dans Simulink® nous allons saisir le modèle ci-contre constitué d'un « **Chart** », de trois « **Switches** » affectant la valeur 0 ou 1 aux trois variables binaires a, b et c.

Le diagramme d'état obtenu en double-cliquant sur le « **Chart** » est le suivant :




E0 est un état composite dans lequel se trouvent les trois états ON, OFF et STOP.

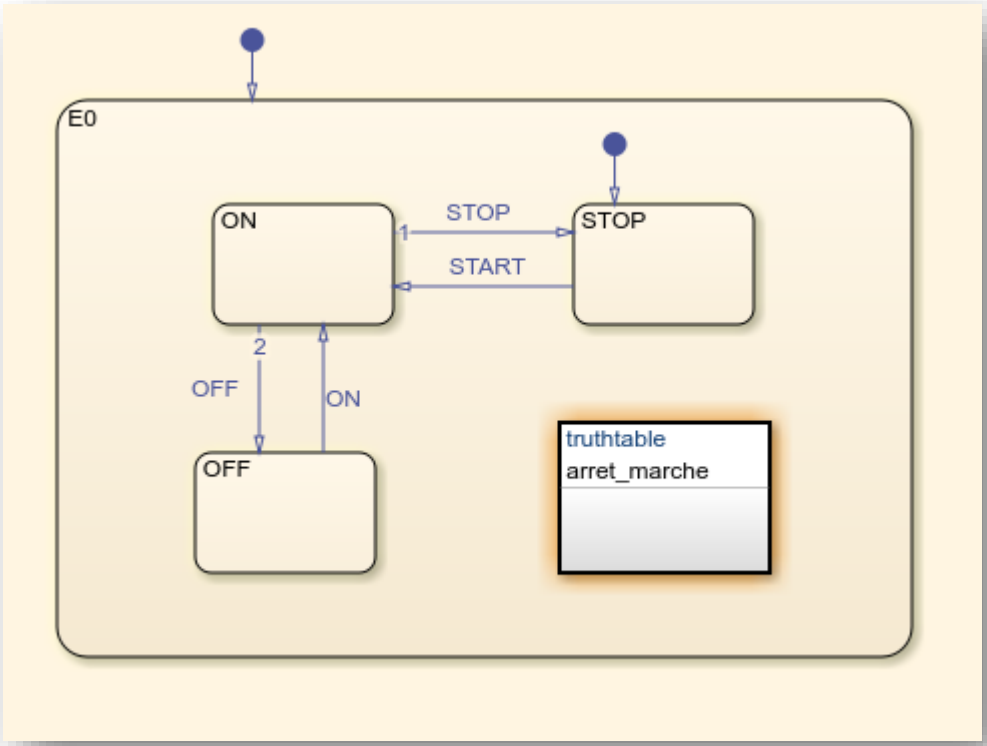
Afin de limiter le nombre de transitions et donc de simplifier le diagramme il est intéressant d'utiliser une table de vérité qui renvoie ici les évènements de type « **local** » *start*, *stop*, *on* et *off* .

Le fonctionnement est décrit dans la table de vérité qui suit. Il est propre à l'état E0. La table sera donc insérée dans l'état E0.

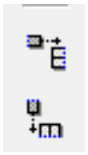
On considère la table de vérité suivante pour traduire le comportement logique du système :

| c | b | a | Evènement associé | Etat pointé |
|---|---|---|---|---|
| 0 | 0 | 0 | stop | E0.STOP |
| 0 | 0 | 1 | on | E0.ON |
| 0 | 1 | 0 | on | E0.ON |
| 0 | 1 | 1 | off | E0.OFF |
| 1 | 0 | 0 | off | E0.OFF |
| 1 | 0 | 1 | on | E0.ON |
| 1 | 1 | 0 | on | E0.ON |
| 1 | 1 | 1 | start | E0.ON |

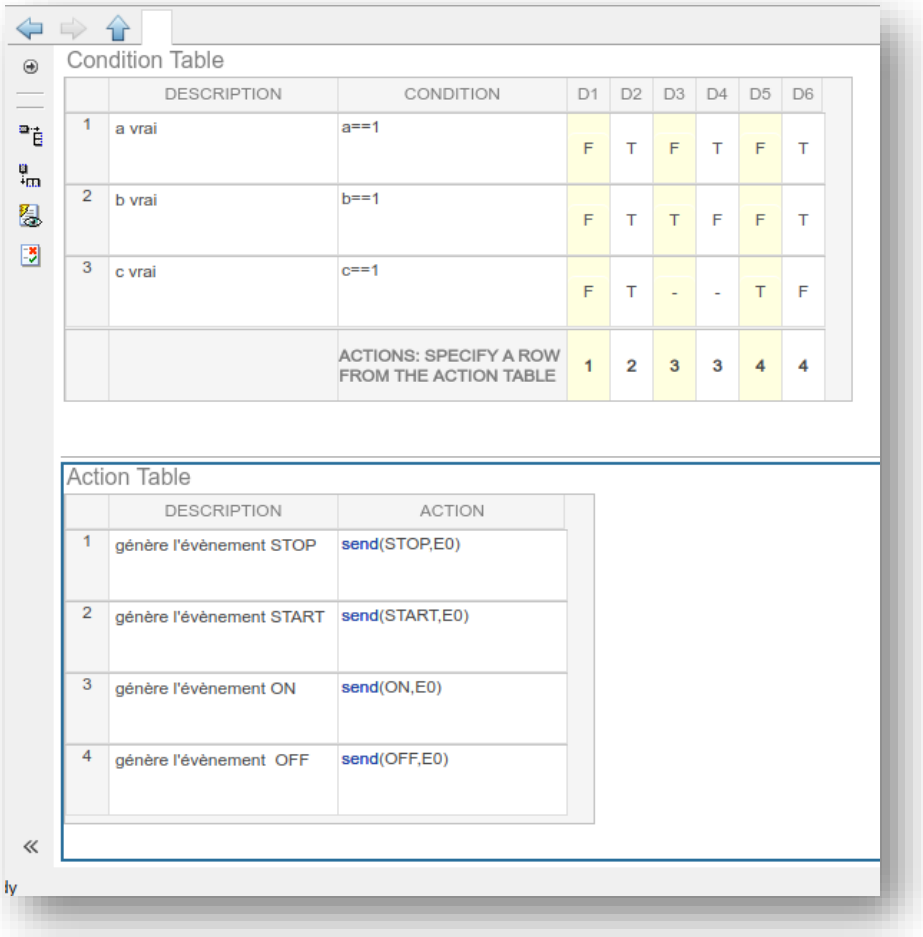
Après avoir glissé-déposé la table de vérité, nous allons saisir son étiquette (« **label** ») : « **arret_marche »** à la place du point d'interrogation :




Un double-clic sur le rectangle ouvre la table de vérité qu'il faut compléter conformément au mode de fonctionnement normal de notre système. Nous devons ajouter des lignes et des colonnes :



Ces icônes du menu latéral permettent de le faire.



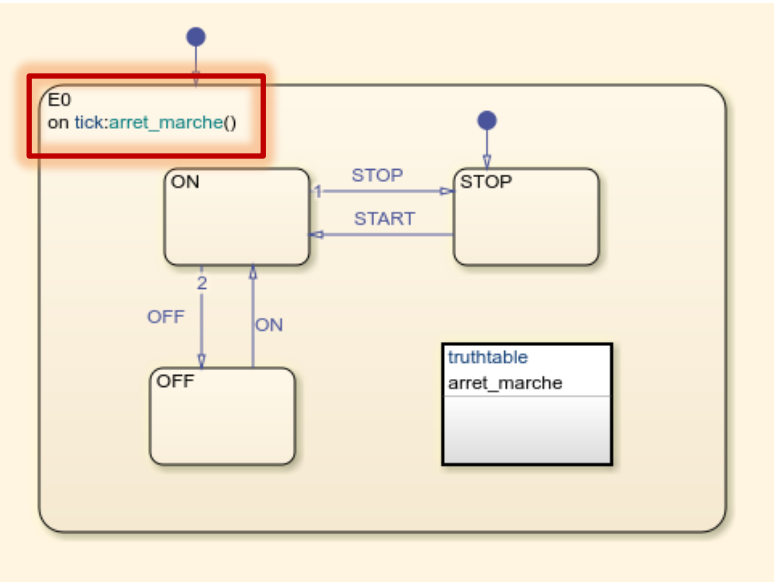

La table de vérité est maintenant remplie.

Or pendant la simulation il faut appeler la table de vérité alors que l'état E0 est actif.

On utilise ici le mot clé « **on *évènement* : action ;** ».

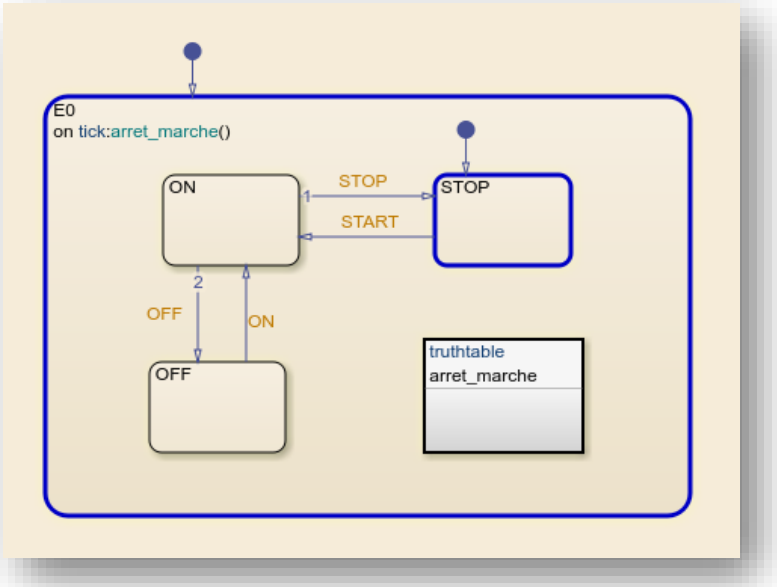
La table de vérité sera appelée à chaque pas de calcul.

L'évènement déclencheur sera donc « **tick** ».




La simulation est désormais prête à être lancée. L'explorateur de modèle donne la structure suivante :

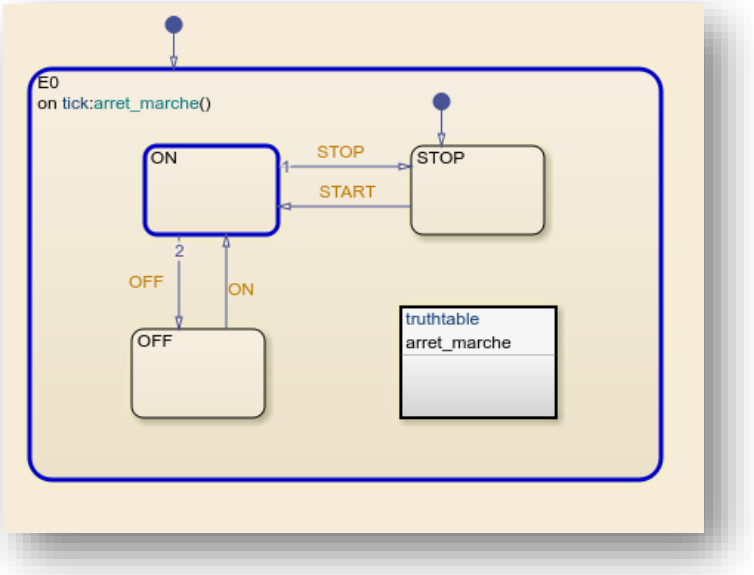
| Name | Scope | Port | Resolve Signal | DataType | Size |
|---|---|---|---|---|---|
| E0 | | | | | |
| a | Input | 1 | | Inherit: Same as Simulink | -1 |
| b | Input | 2 | | Inherit: Same as Simulink | -1 |
| c | Input | 3 | | Inherit: Same as Simulink | -1 |
| STOP | Local | | | | |
| OFF | Local | | | | |
| START | Local | | | | |
| ON | Local | | | | |

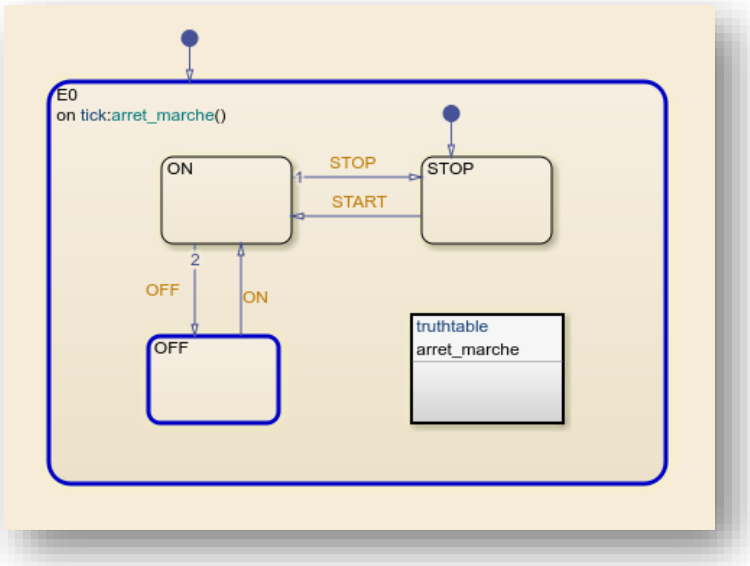



Au démarrage l'état STOP est actif.

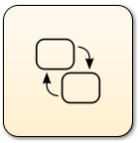
L'état ON sera activé à l'occurrence de l'évènement *start,* c'est-à-dire pour a, b, et c vrais.

L'état OFF sera activé à l'occurrence de l'évènement *off*, c'est-à-dire pour a et b vrais puis c faux.




Ainsi de suite…

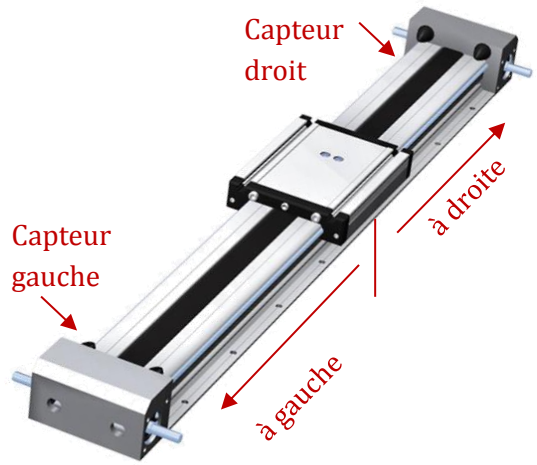


# Chapitre 5

# *D'autres exemples d'application du« Chart »*

# Chapitre 5 - D'autres exemples d'application du « Chart »…

## 5.1- INITIALISATION D'UN AXE LINEAIRE




Nous allons considérer l'axe linéaire ci-contre.

Deux capteurs TOR de fin de course sont montés sur cet axe asservi en position: un à l'extrémité gauche et un à l'extrémité droite. Ils sont disposés juste avant les butées élastiques gauche et droite.

Les deux capteurs sont utilisés pour le lancement d'une routine de sécurité afin que le moteur ne soit jamais alimenté rotor bloqué sur les butées. De plus, le capteur de gauche est utilisé pour l'initialisation du compteur associé au codeur qui assure l'asservissement en position.

Dans cette application, on s'intéresse à la phase d'initialisation de l'axe.

L'initialisation doit respecter le cahier des charges suivant :

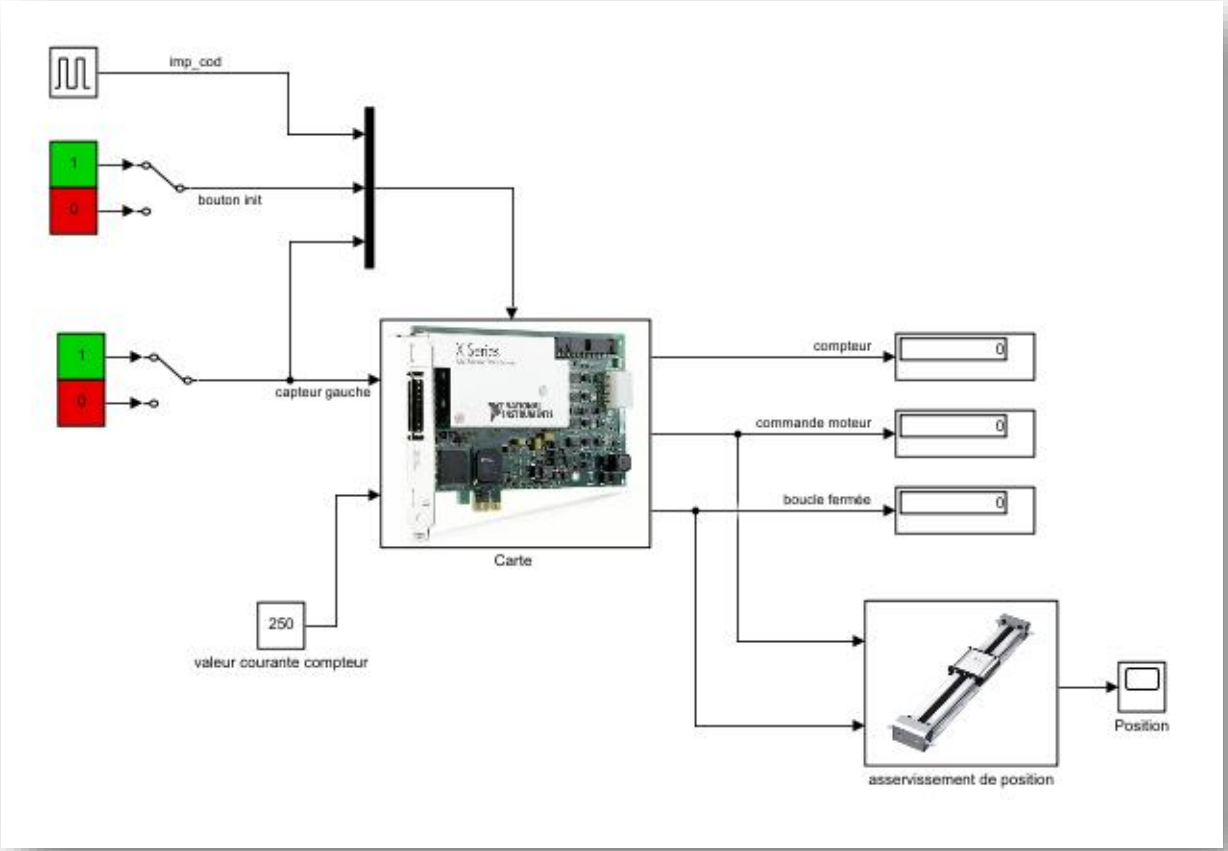
Après appui sur le bouton d'initialisation si le capteur gauche n'a pas détecté la présence du chariot celui-ci se translate à vitesse rapide en boucle ouverte sur la gauche. Dès que la présence du chariot est détectée à gauche il repart à droite à vitesse lente toujours en boucle ouverte jusqu'à ce que le capteur gauche commute à nouveau.

Si au moment du relâchement du bouton d'initialisation, le chariot est à gauche et donc que le capteur gauche a détecté sa présence, il se déplace vers la droite à vitesse lente en boucle ouverte jusqu'à ce que le capteur gauche commute.

Au moment de la commutation du capteur gauche lorsque le chariot se déplace vers la droite, la boucle en position est fermée et la consigne de position imposée nulle. Le compteur associé au codeur est simultanément initialisé.

Le compteur est incrémenté pour un déplacement vers la droite et décrémenté pour un déplacement vers la gauche.

Mettons l'interface Simulink® en place :




Au dessus du « **Chart** » (Carte) nous retrouvons les évènements : init, detec_cg et imp_cod.

En entrée nous retrouvons la variable associée au capteur fin de course gauche, cg ainsi que la valeur du compteur d'impulsions issues du codeurs avant initialisation.

L'initialisation sera lancée lorsqu'après appui sur le bouton poussoir on le relachera (front descendant).

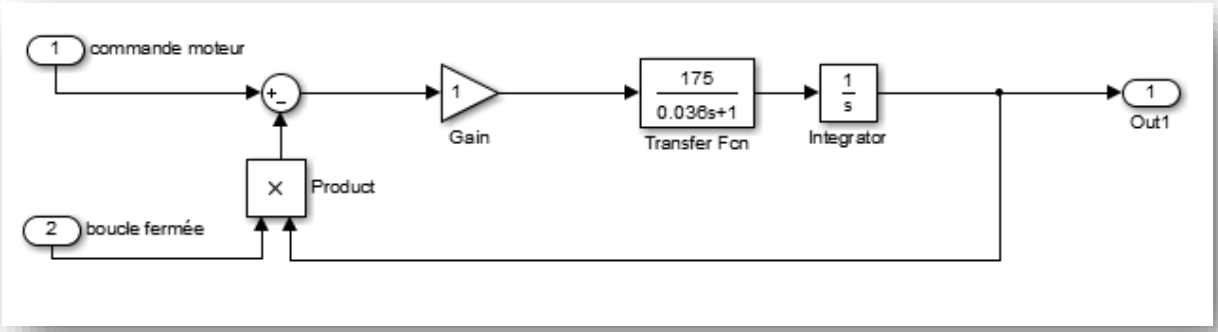
L'évènement detect_cg correspond à un front montant ou un front descendant de la variable cg.

L'évènement imp_cod correspond au train d'impulsions délivré par le codeur. Ces impulsions sont modélisées par les fronts montants et descendants d'un signal carré.

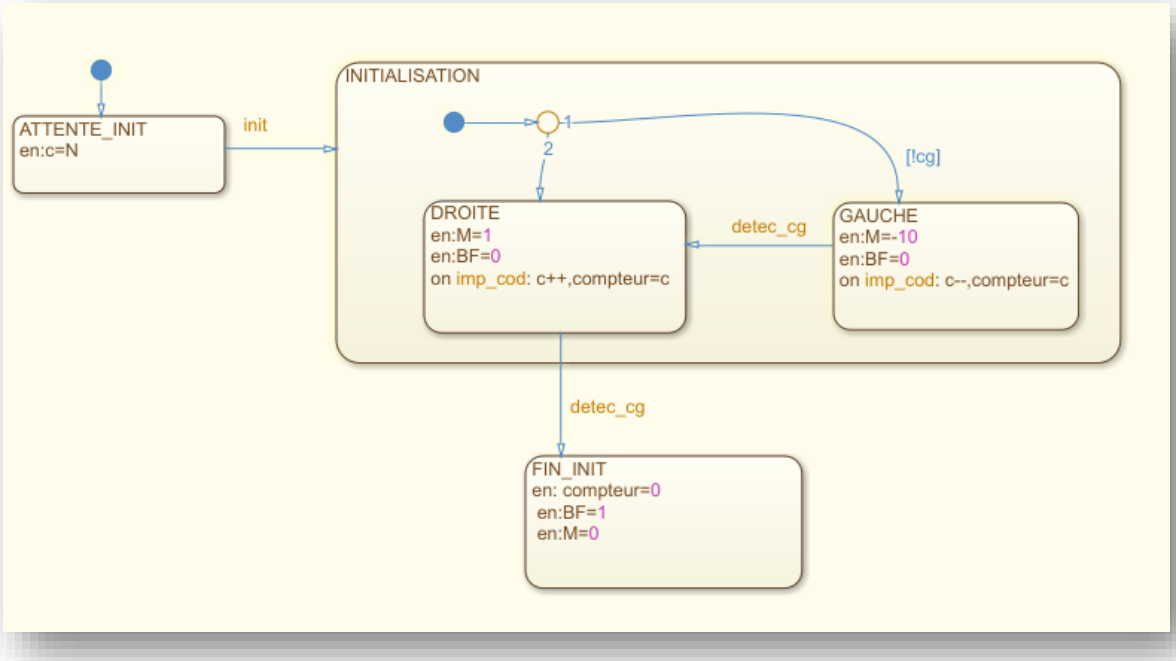
En sortie sont affichées la commande du moteur (rotation rapide dans un sens : M=-10, lente dans l'autre : M=1, arrêt : M=0), l'état d'une variable binaire de gestion de la

boucle d'asservissement (boucle fermée : BF=1, boucle ouverte BF=0), et la valeur courante du compteur.

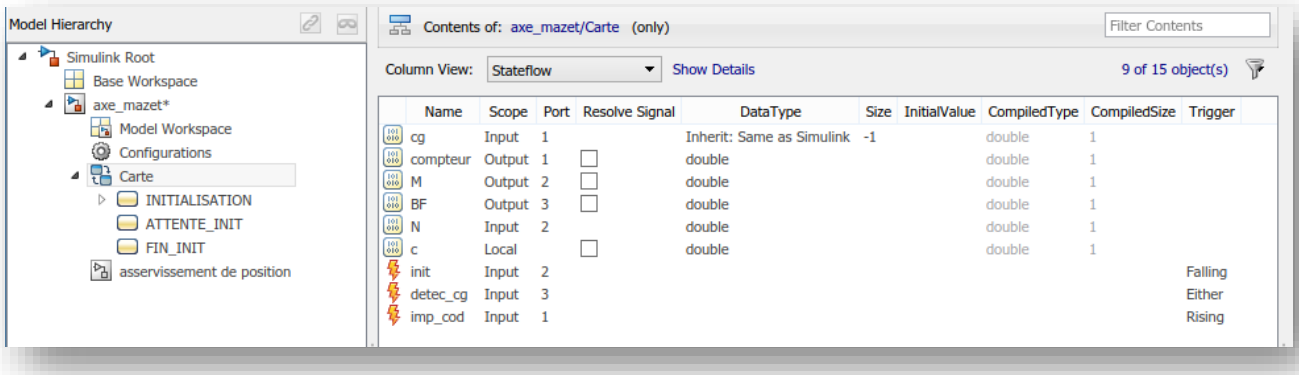
L'asservissement en position est défini par le schéma-blocs :



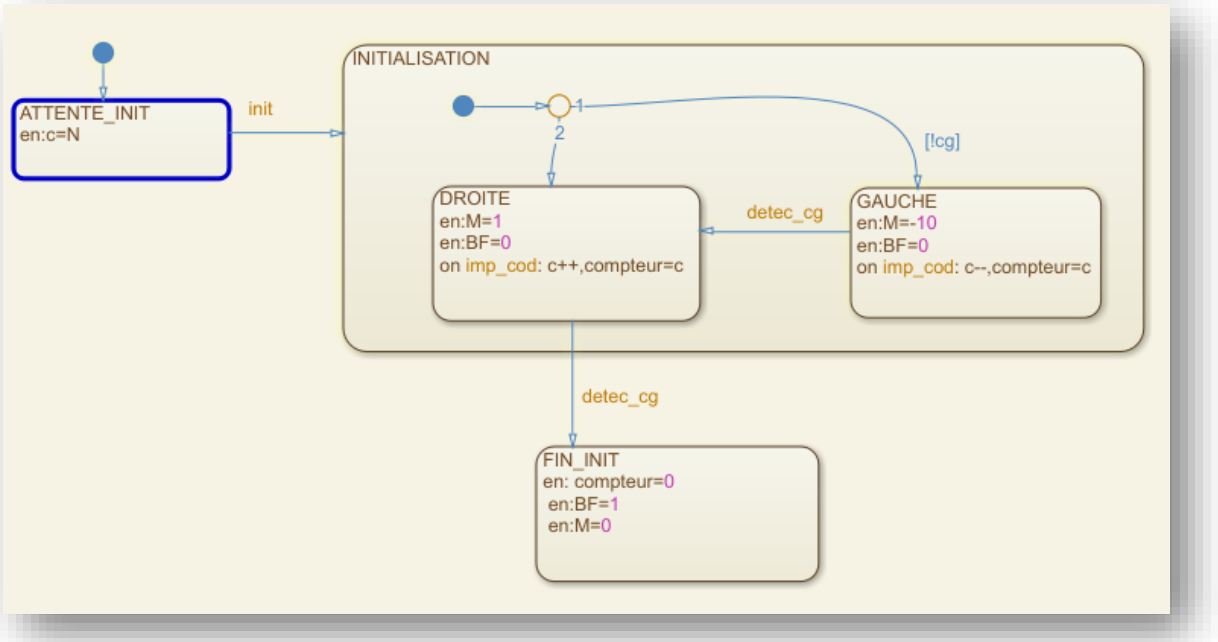

Une solution possible pour la phase d'initialisation qui reprend le cahier des charges peut être celle qui suit :



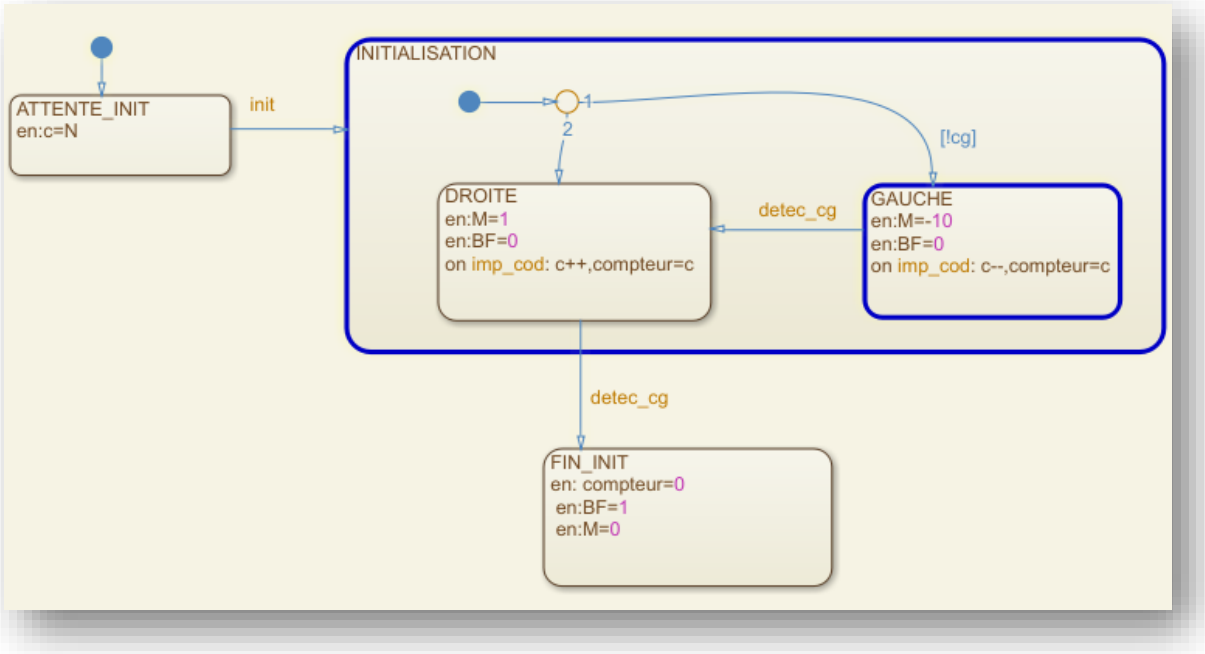

l'explorateur de modèle précise les entrées et sorties :



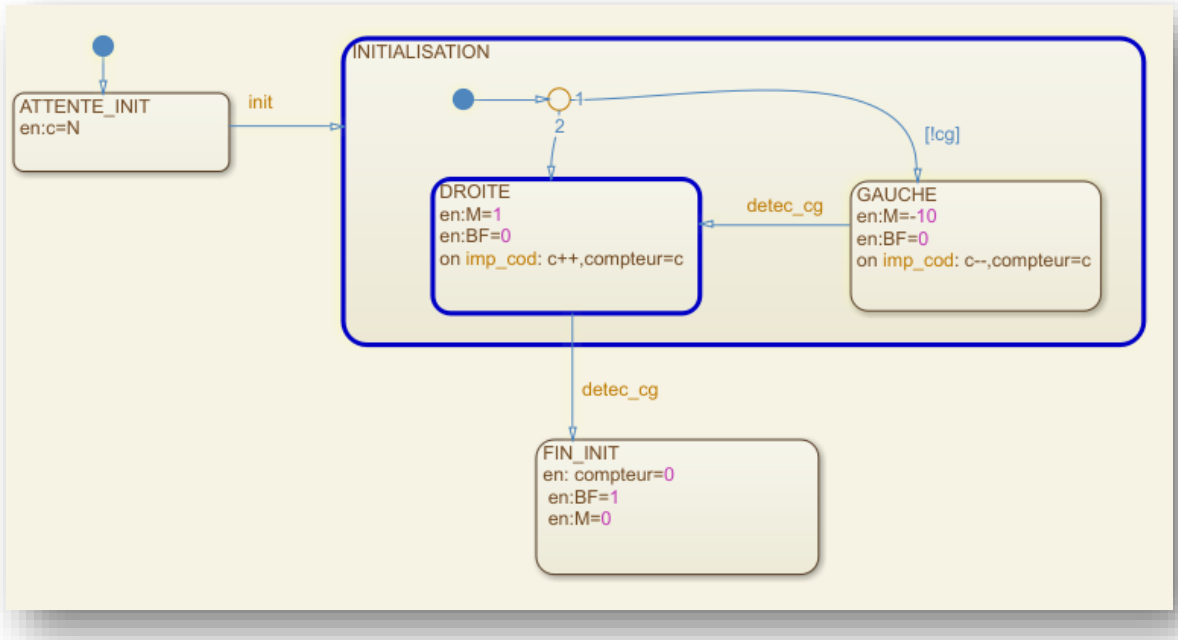

Analysons les résultats de la simulation. En attente de l'initialisation, l'état ATTENTE_INIT est actif :



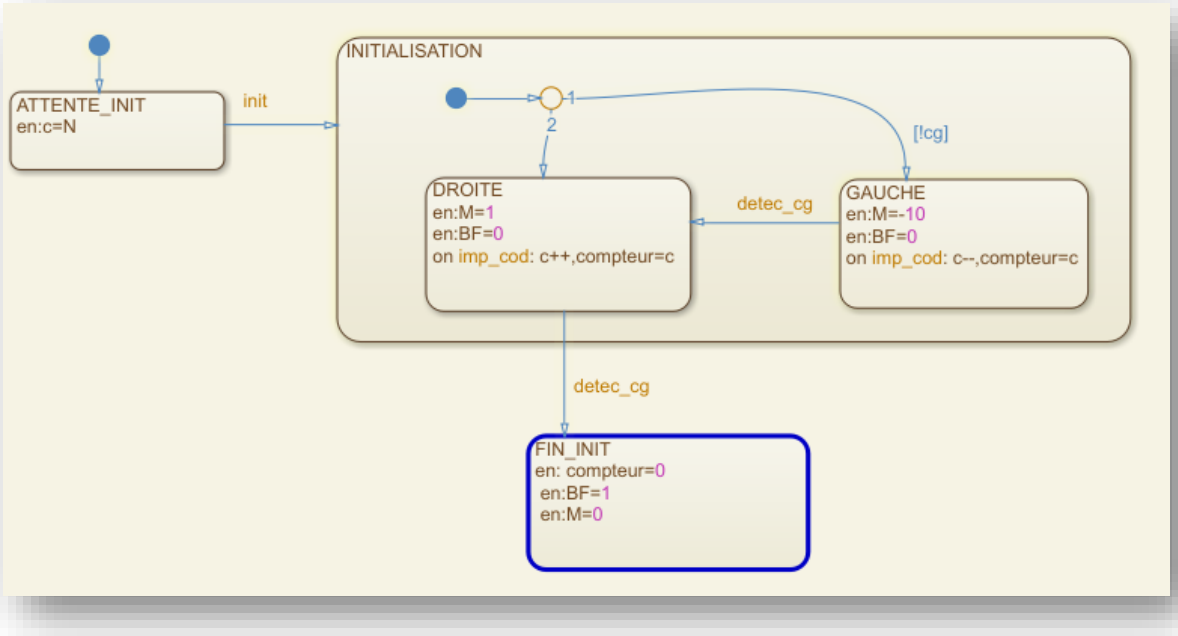

On appuie sur le bouton d'initialisation pui on le relâche. Un front descendant sur init est alors détecté, l'état ATTENTE_INIT est désactivé et l'état INITIALISATION est activé. Dans le cas ci-dessous, le capteur gauche n'est pas commuté, [!cg] est donc vrai. L'état GAUCHE est donc activé. Le chariot se déplace vers la gauche à vitesse rapide en boucle ouverte aussi le compteur d'impulsions est décrémenté :




Lorsque le capteur gauche commute, on détecte alors un front montant. L'état GAUCHE est désactivé et l'état DROITE est activé. Le chariot se déplace vers la droite à vitesse lente en boucle ouverte aussi le compteur d'impulsions est incrémenté :

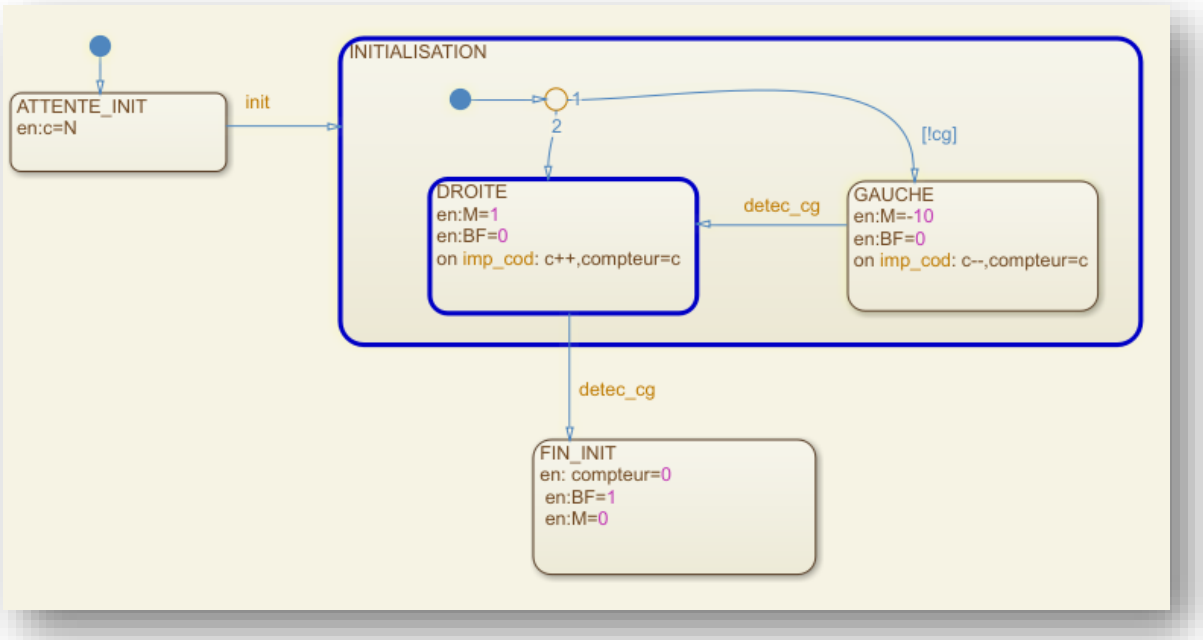



En se déplaçant vers la droite le chariot va « libéreré » le capteur gauche qui va commuter. Un front descendant est alors détecté juste après le front montant précédent. L'état DROITE est désactivé et l'état FIN_INIT est activé. Le compteur d'impulsion est remis à zéro, le moteur est arrêté et la commande bascule en boucle fermée.




Cet état final caractérise la fin de l'initialisation de l'axe.

Si au moment du relâchement du bouton d'initialisation, le chariot est à gauche et donc que le capteur gauche a détecté sa présence, l'état DROITE est activé. Le chariot se déplace alors vers la droite à vitesse lente en boucle ouverte jusqu'à ce que le capteur gauche commute, l'état FIN_INIT est alors activé caractérisant ainsi la fin de l'initialisation.



## 5.2- PILOTAGE D'UNE PLATEFORME OMNIDIRECTIONNELLE

La société française Sovam installée à Parthenay met au point et fabrique des équipements destinés à la maintenance aéronautique.

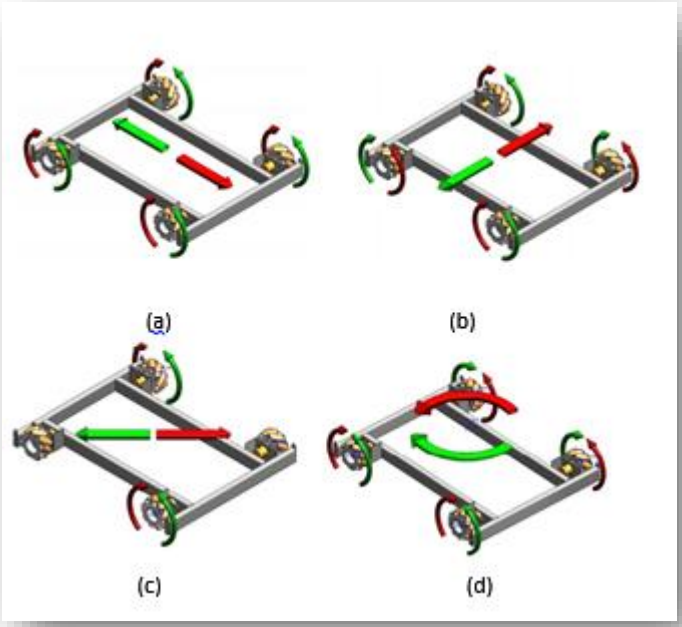
La plateforme omnidirectionnelle Easymov'x en fait partie et peut être utilisée dans des ateliers de peinture par exemple.

Sa base est équipée de quatre roues Mecanum (ou suédoises) indépendamment motorisées.

Des batteries permettent l'alimentation électrique des différents composants et en particulier des moteurs. Une centrale hydraulique permet l'alimentation des deux vérins permettant à la plateforme d'évoluer verticalement d'une hauteur de 1m à 8m par rapport au sol.



Le pilotage de la plateforme se fait en agissant sur deux joysticks ; un pour commander la montée et la descente de la plateforme et le second pour commander le déplacement de la base.



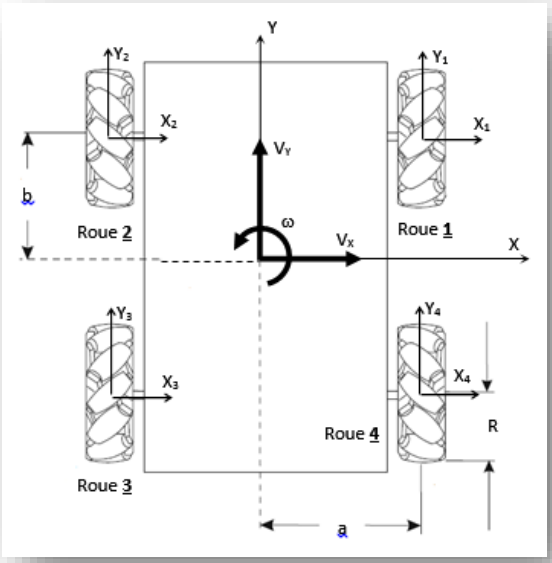

Le déplacement de la base est omnidirectionnel, c'est à dire qu 'elle peut se déplacer suivant toutes les directions et en particulier latéralement ou en diagonal, ou bien encore en rotation autour d'un axe vertical passant par le centre de la base. Toutes les combinaisons sont possibles sans roues directrices.

Cette base est mue par quatre roues Mecanum motorisées. Elles sont constituées de deux flancs solidaires du moyeu entre lesquels des galets orientés à 45° par rapport à l'axe de la roue peuvent tourner librement par rapport à leur axe longitudal.

Le but de cette application est d'élaborer un modèle de commande des moteurs pour les déplacements longitudinaux, latéraux, diagonaux et de rotation autour de l'axe vertical.

Auparavant, une étude cinématique doit être menée pour établir le modèle cinématique inverse qui met en relation le déplacement de la plateforme avec la rotation des moteurs qui entrainent les roues.

Le paramétrage choisit pour écrire ce modèle est le suivant :




En formulant l'hypothèse de roulement sans glissement des galets par rapport au sol, on montre que le modèle inverse s'écrit :

$$\begin{pmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{pmatrix}=\frac{1}{R}\begin{pmatrix}1 & -1 & -(a+b)\\ -1 & -1 & a+b\\ 1 & -1 & a+b\\ -1 & -1 & -(a+b)\end{pmatrix}\begin{pmatrix}V_x\\ V_y\\ \omega\end{pmatrix}$$

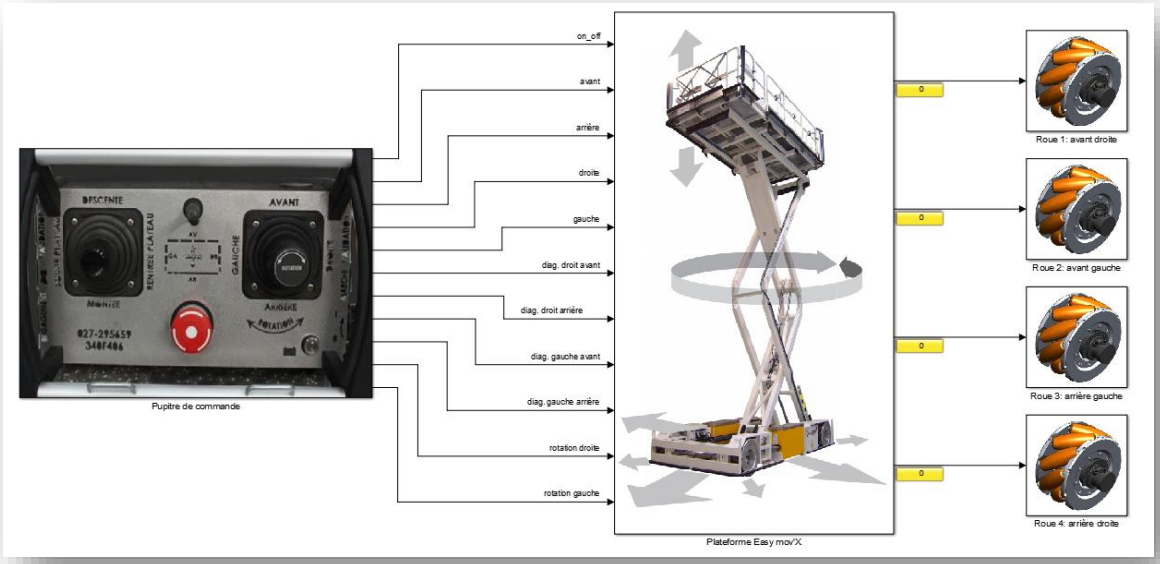
Intéressons-nous maintenant à la commande des moteurs pour différents déplacement de la plateforme.

Compte tenu du paramétrage précédent, nous affecterons la valeur 1 à la variable Mi associée à la rotation d'une roue i lorsque celle-ci tourne dans le sens direct, -1 pour une rotation dans le sens indirect, et 0 lorsque la roue est à l'arrêt.

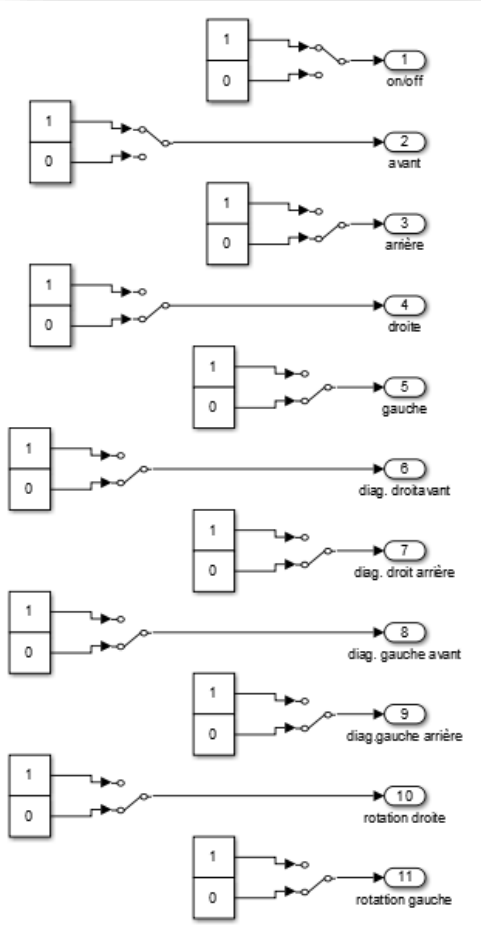
Ainsi la plateforme avancera suivant Y si : M1 = M2 = M3 = M4 = -1

D'autre part on considèrera que les moteurs, lorsqu'ils tournent, le font à la même vitesse.

L'interface Simulink® se présente de la manière suivante :



On y retrouve les modèles associés au pupitre de commande, et à la commande des moteurs.



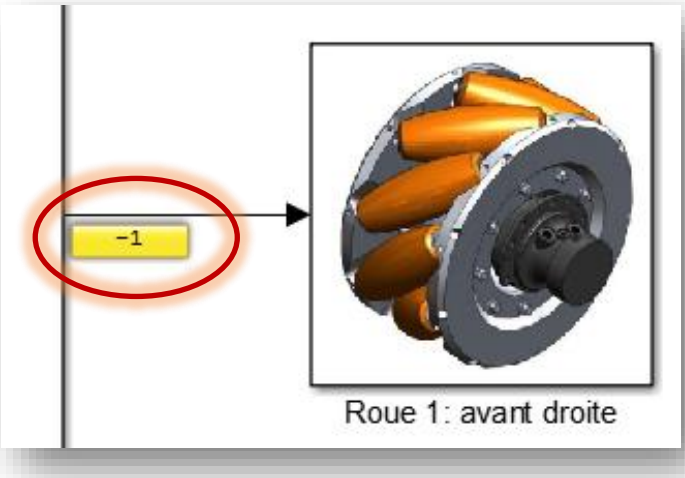

Un double-clic sur le bloc « **Pupitre de commande** » ouvre la fenêtre ci-contre:

La sortie 1 correspond à la mise en service de la commande des déplacements.

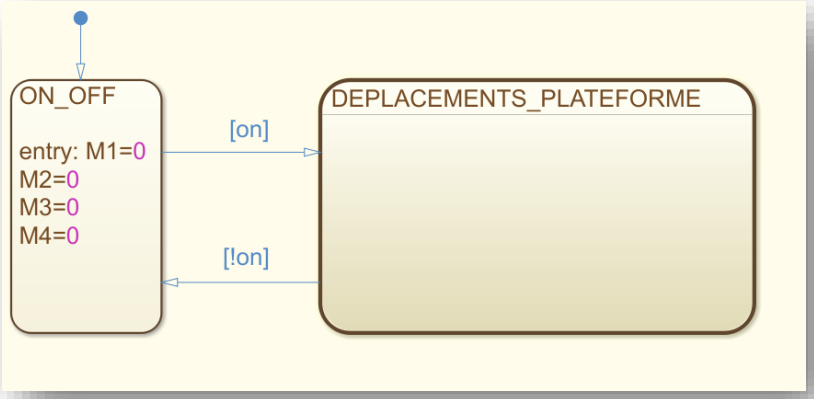
Les sorties repérées 2 à 11 autorisent les dix mouvements possibles de la plateforme : longitudinaux, latéraux, diagonaux et de rotation.

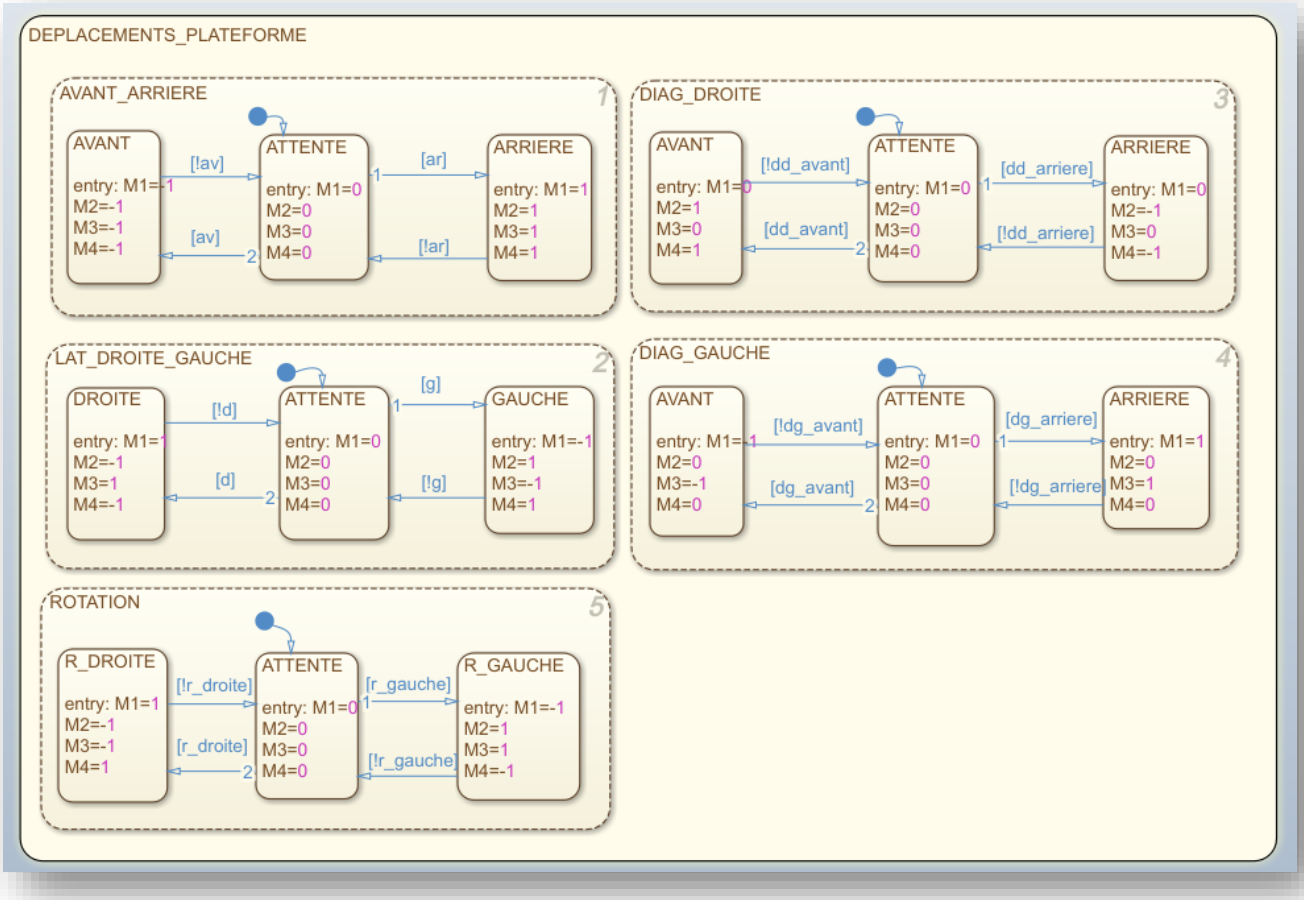
En sortie du bloc « **Plateforme Easymov'X** » on retrouve les quatre roues motorisées indépendamment les unes des autres.

La valeur de la variable associée au sens de rotation des roues donc des moteurs s'affiche dans le rectangle jaune. Pour ce faire, il faut sélectionner la liaison entre la sortie du bloc « **Plateforme Easymov'X** » et l'entrée du bloc « **Roue…** » puis clic droit et sélectionner la ligne du menu contextuel : « **Show Value Label of Selected Port** ».

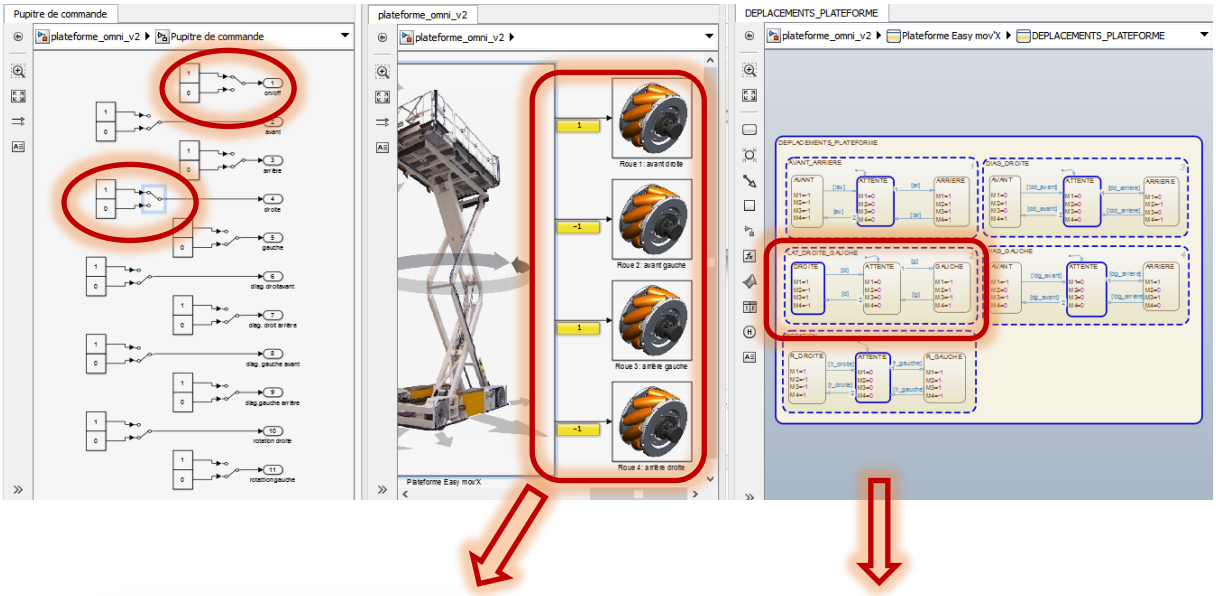


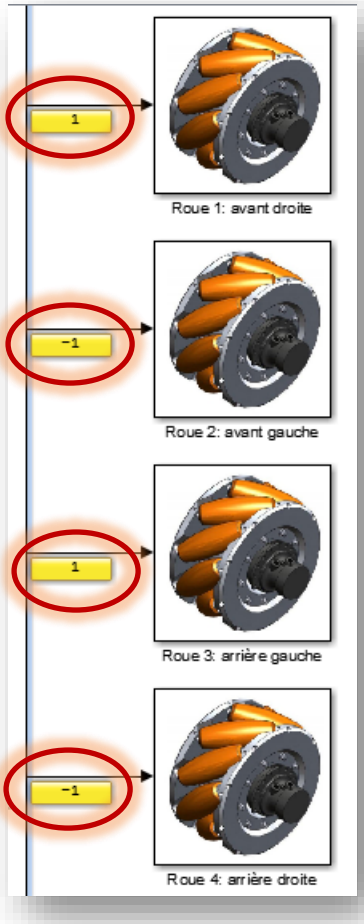
Concernant maintenant la commande des moteurs, un double-clic sur la bloc « **Plateforme Easymov'X** » permet d'accéder au « **Chart** » suivant :

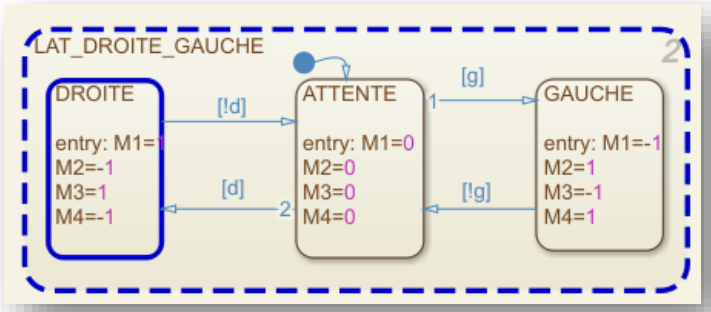


La simulation pour un déplacement latéral droit donne les résultats suivants :




Pour compléter cet exercice, il peut être intéressant d'ajouter la gestion de la montée et de la descente de la plateforme en ajoutant le deuxième joystick du pupitre de commande et en combinant les différents mouvements alors envisageables.

## 5.3- TRAITEMENT DES INFORMATIONS DELIVREES PAR UN CODEUR SINCOS

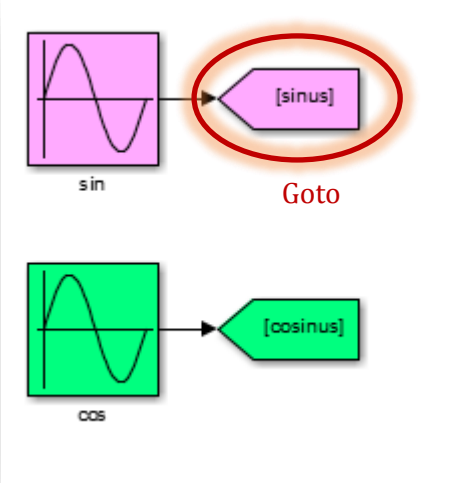
Reprenons l'exemple du codeur SinCos Hiperface® déjà présenté au chapitre 4 lorsque nous avons introduit la fonction MATLAB.

Nous avons vu comment générer le train d'impulsions servant à la détermination de la position angulaire grossière du rotor du capteur.

Nous allons maintenant construire les diagrammes qui donneront à la fois la position grossière et la position fine pour un seul sens de rotation.

La position fine est obtenue au moyen de la fonction arc-tangente interpolée par un algorithme entre deux positions grossières consécutives. Cette interpolation n'est pas prise en compte ici.

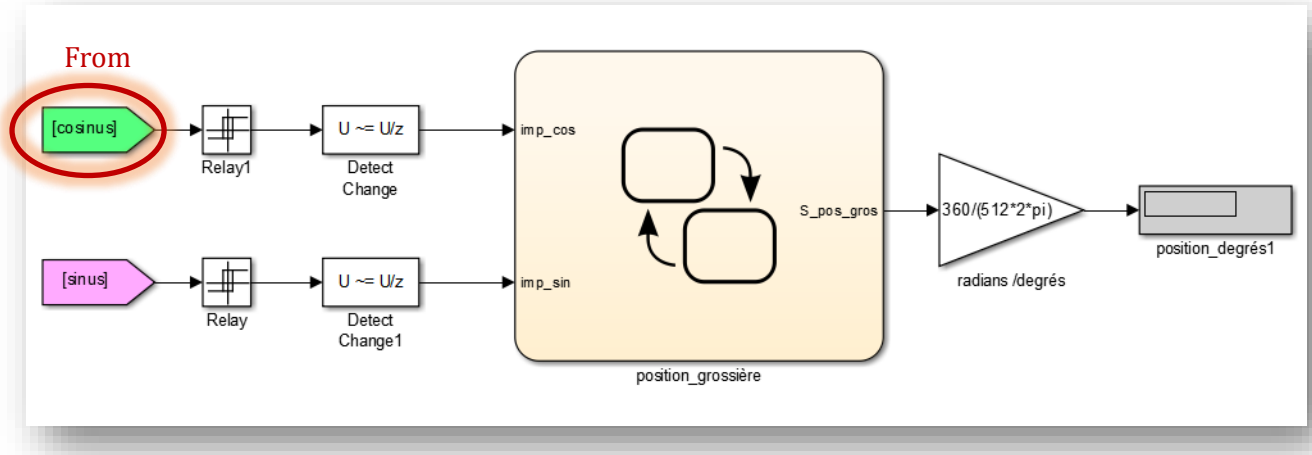
Intéressons-nous à la détermination de la position grossière et préparons l'interface Simulink® :




Mettons tout d'abord en place les signaux sinusoïdaux délivrés par le codeur. Les sorties des blocs sont reliées à des blocs « **Goto** » pour alléger le modèle en limitant le nombre de liaisons. Ce bloc permet de transmettre le signal à un bloc « **From** » qui portera le même nom.

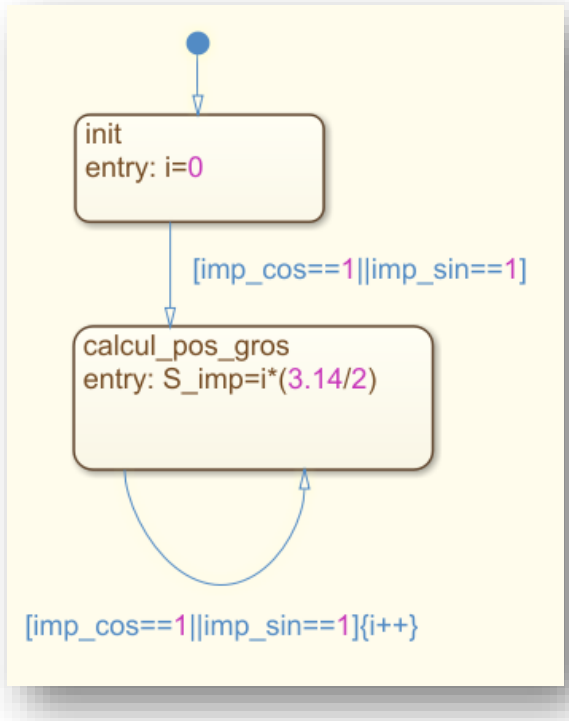
Ces deux blocs se trouvent dans la bibliothèque « **Signal Routing** » de Simulink®.

Ensuite vient le « **Chart** » qui reprend la structure vue au chapitre 4:




Nous disposons à l'entrée du « **Chart** » des trains d'impulsions « **imp_cos** » et « **imp_sin** » qui vont permettre d'obtenir en sortie la position angulaire « **S_pos_gros** » en radians qui sera convertie en degrés. Le codeur délivre 512 périodes. Une impulsion correspond à un quart de période comme nous l'avons déjà vu.

D'où le diagramme :

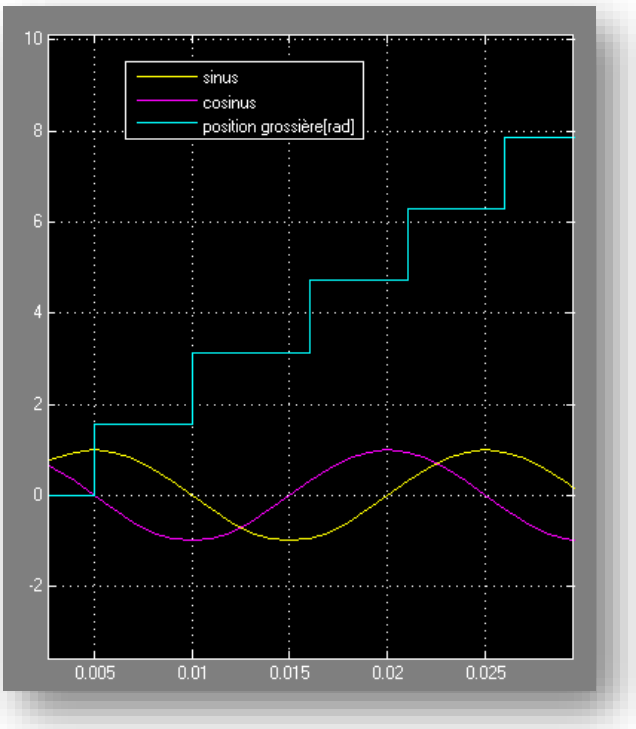



On initialise la variable de comptage « **i** ».

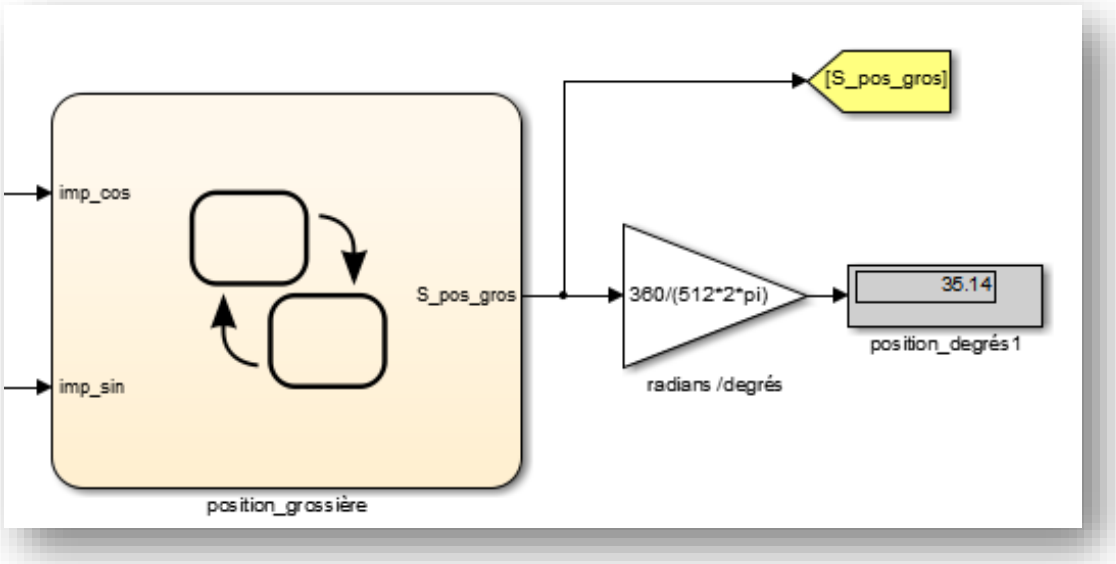
Après détection d'une impulsion on affiche la rotation en radians du rotor du codeur.

A chaque impulsion le compteur est incrémenté d'une unité.

La simulation d'une durée de 0.03s donne les résultats suivants :

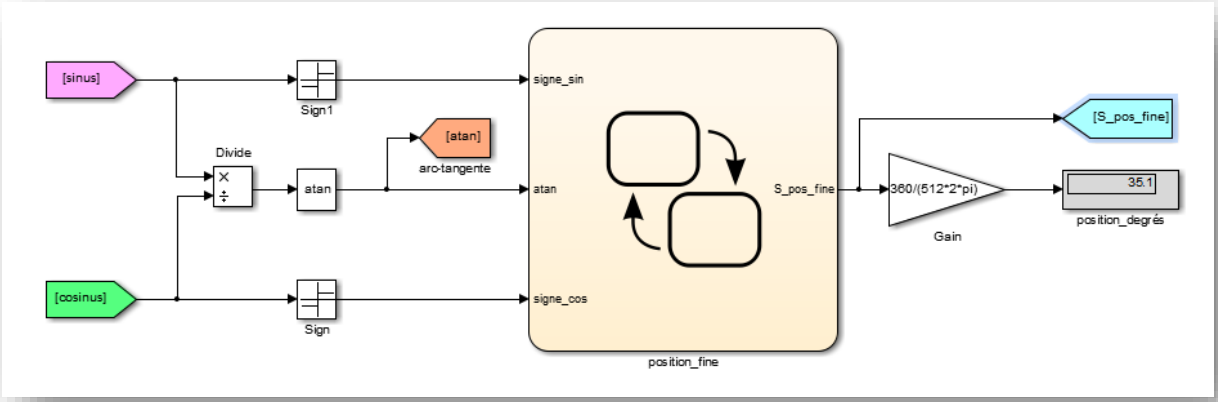



Pour une pulsation des signaux sinusoïdaux de 100π rad/s le rotor du codeur a atteint ici une position angulaire de 35,14°.



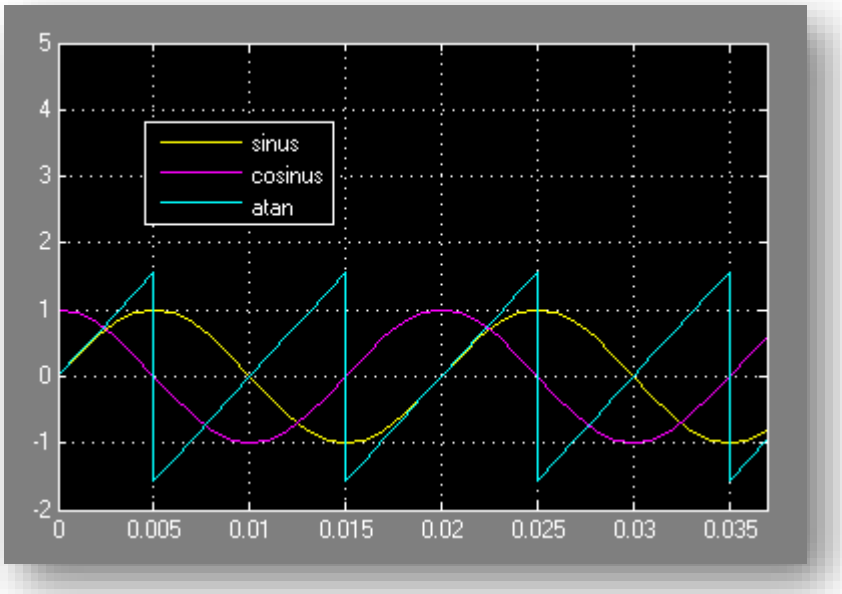
Intéressons-nous maintenant à la détermination de la position fine à partir de la fonction arc-tangente.

Préparons l'interface Simulink® correspondante:




Pour afficher la position angulaire fine du rotor du capteur il est nécessaire :

- de diviser le sinus par le cosinus (bloc « **Divide** » dans la bibliothèque **« Math Operations** » de Simulink®) et d'extraire l'angle avec la fonction arc-tangente (bloc « **atan** » dans la bibliothèque « **Math Operations** » de Simulink®),
- de connaître le signe du sinus et du cosinus pour connaitre le quadrant dans lequel se situe l'angle à afficher (bloc « **Sign** » dans la bibliothèque « **Math Operations** » de Simulink®).

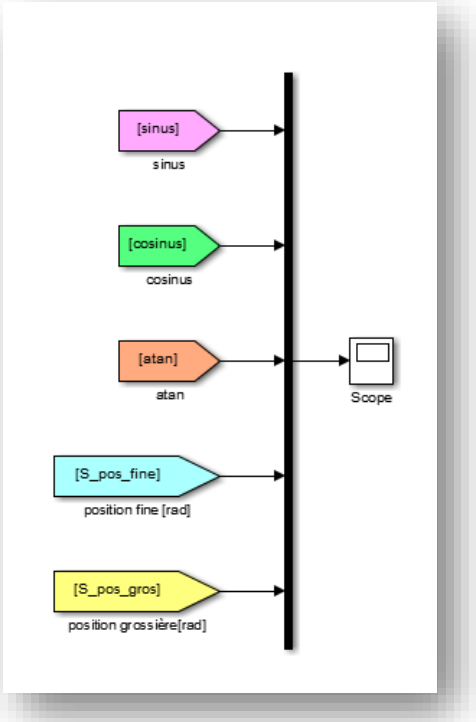



Le résultat que donne la fonction « **atan** » met clairement en évidence son changement de signe à chaque quart de période des signaux d'entrée, tous les $\pi/2$ radians. On note aussi une discontinuité lorsque le cosinus est nul.

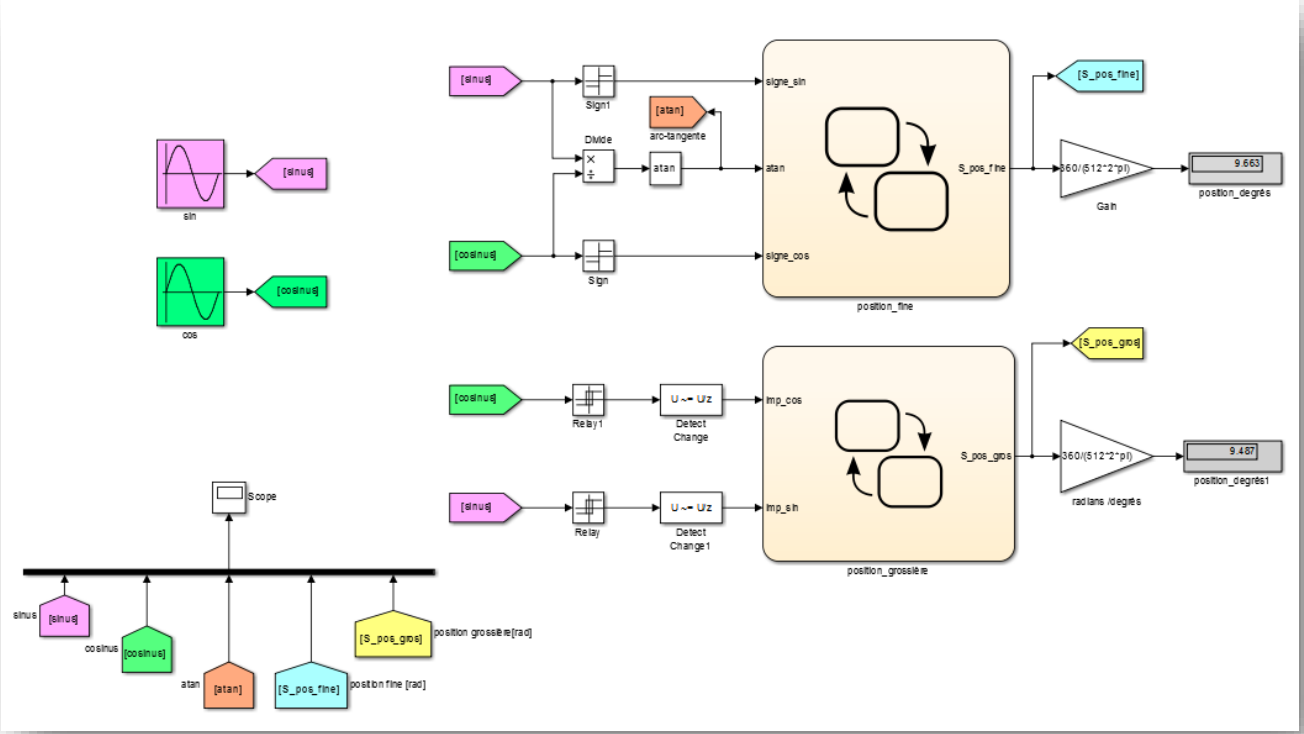
Par conséquent la position fine dépend du signe du cosinus et du signe du sinus qui définit ainsi le quadrant d'évolution de la position angulaire du rotor.

La sortie du bloc « **Sign** »vaut 1 si son entrée est positive, -1 si elle est négative, sinon elle vaut 0.

Pour compléter l'interface Simulink nous allons synthétiser les résultats de la simulation dans un même « **Scope** » en utilisant le bloc « **Mux** » :

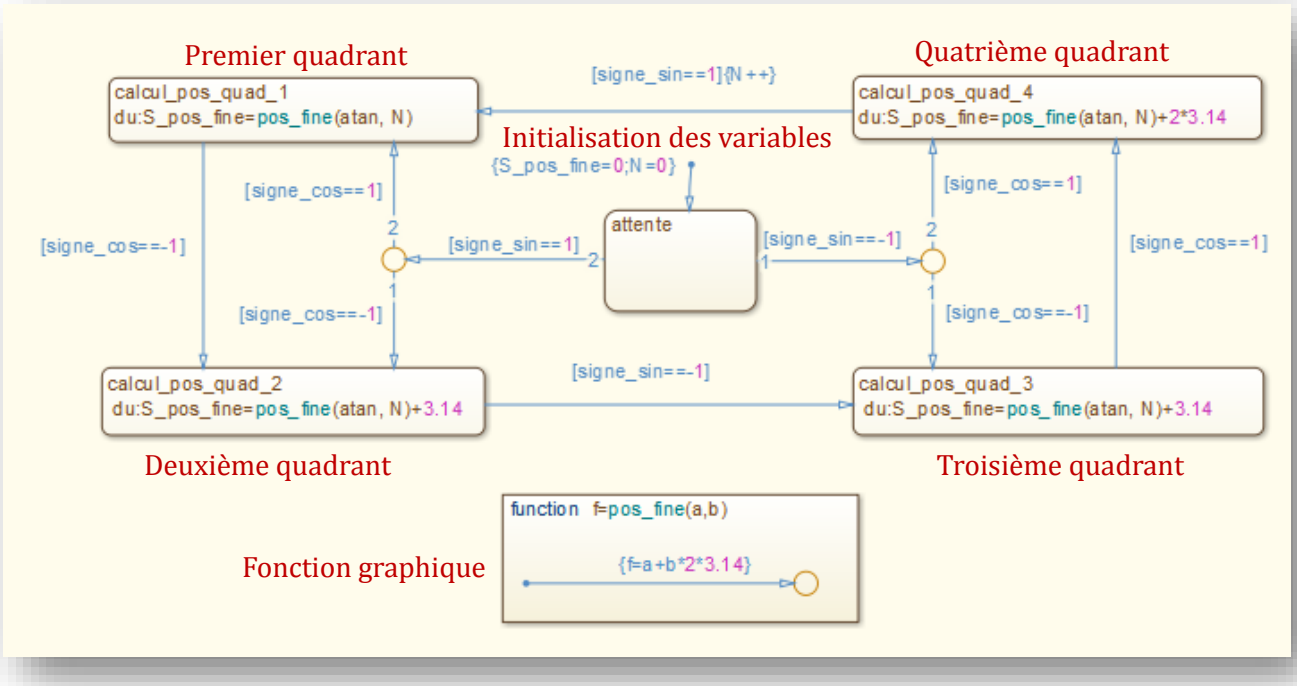



L'interface globale aura alors l'allure suivante :



L'interface Simulink® mise en place nous allons tracer le diagramme qui permet d'obtenir la position fine. Comme nous l'avons vu précédemment il faut identifier les quatre quadrants et adapter la position donnée par la fonction arc-tangente pour chacun des quadrants. L'opération consiste à « translater » les valeurs retournées de π ou 2π suivant le cas.

Une solution possible est donnée ci-dessous, elle fait appel à une fonction graphique :




Apportons quelques commentaires sur ce diagramme :

Si le sinus et le cosinus sont positifs la position évolue dans le premier quadrant et vaut la valeur retournée par le bloc « **atan** » jusqu'à ce que le cosinus change de signe.

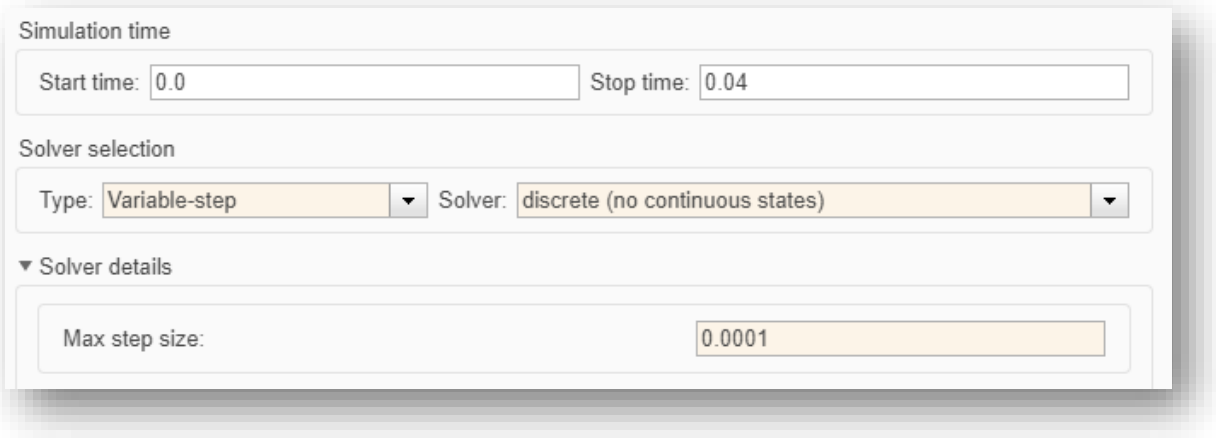
A l'instant où le cosinus change de signe, la position évolue dans le second quadrant. Il faut alors ajouter π à la valeur retournée par la fonction arc-tangente jusqu'à ce que le sinus change de signe.

A l'instant où le sinus change de signe, la position évolue dans le troisième quadrant. Il faut alors ajouter π à la valeur retournée par la fonction arc-tangente jusqu'à ce que le cosinus change de signe.

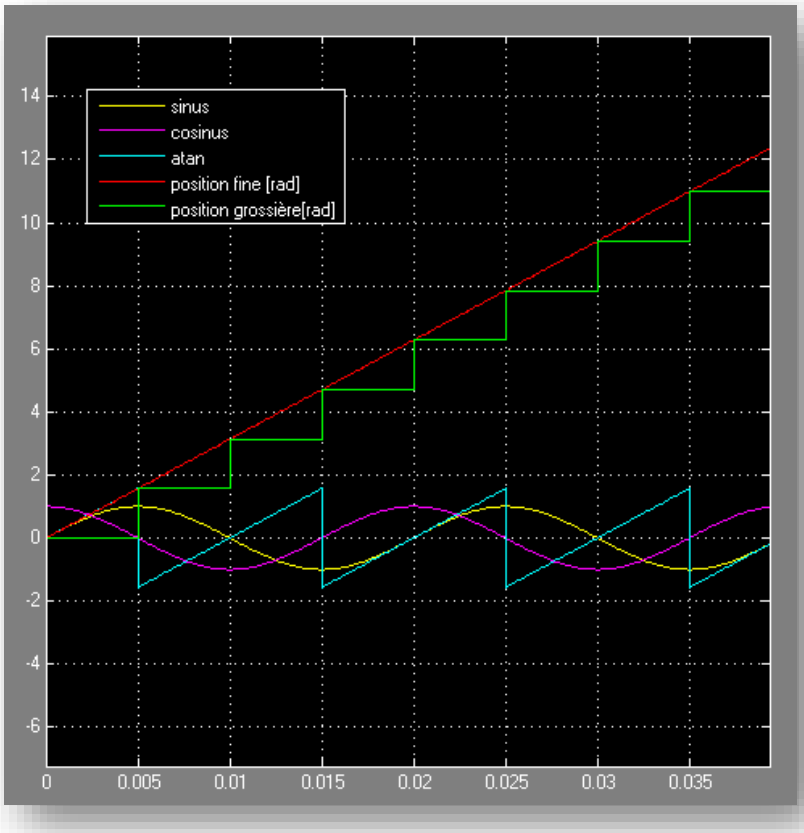
A l'instant où le cosinus change de signe, la position évolue dans le quatrième quadrant. Il faut alors ajouter 2π à la valeur retournée par la fonction arc-tangente jusqu'à ce que le sinus change de signe.

Les quatre quadrants étant parcourus, la variable N est incrémentée d'une unité etc…

Pour la simulation nous allons choisir un solveur « **Discrete** » à pas variable avec un pas maximal de 0.0001s. Ce solveur est adapté lorsque le passage d'un état à l'autre doit se faire rapidement. De plus il diminue le temps de calcul de la simulation :

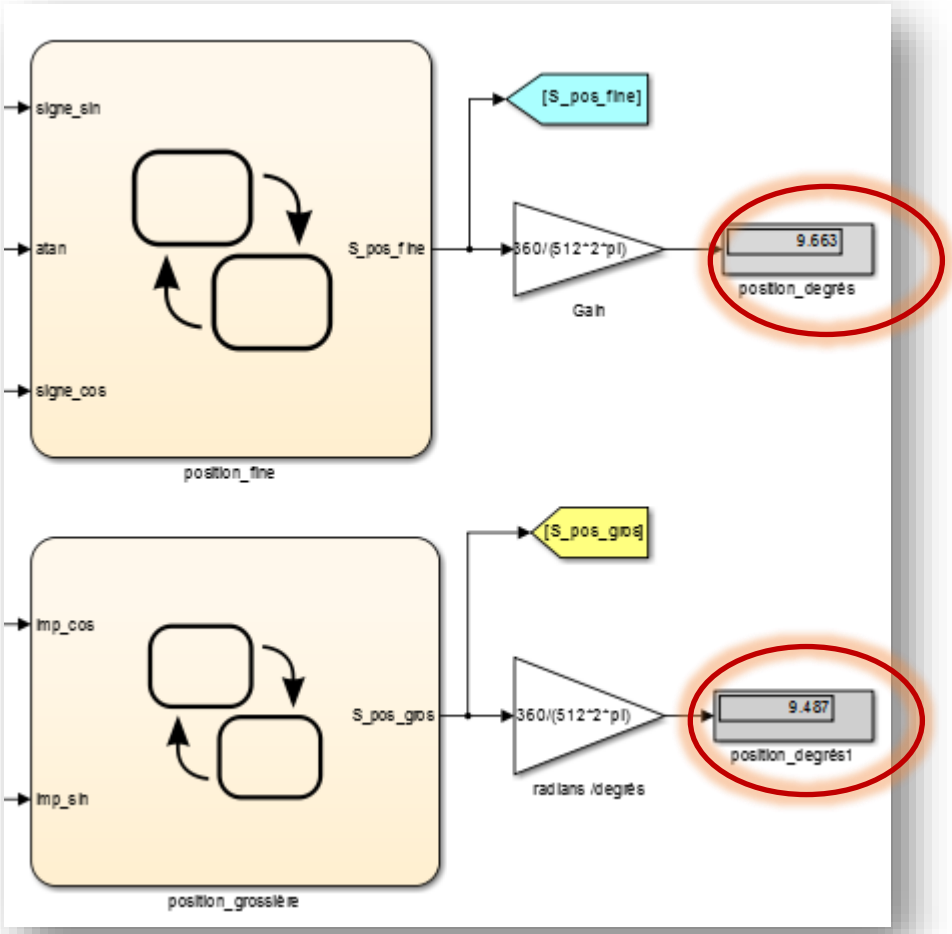



La simulation donne les résultats suivants :

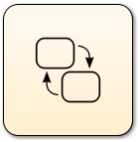



Pour cette simulation nous disposons entre deux positions grossières consécutives de cinquante positions fines intermédiaires.

Les positions grossière et fine du rotor du codeur affichées en degrés sont données ci-dessous :

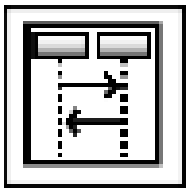



**Remarque** : la démarche serait identique pour le traitement des signaux sinusoïdaux de sortie d'un résolveur.



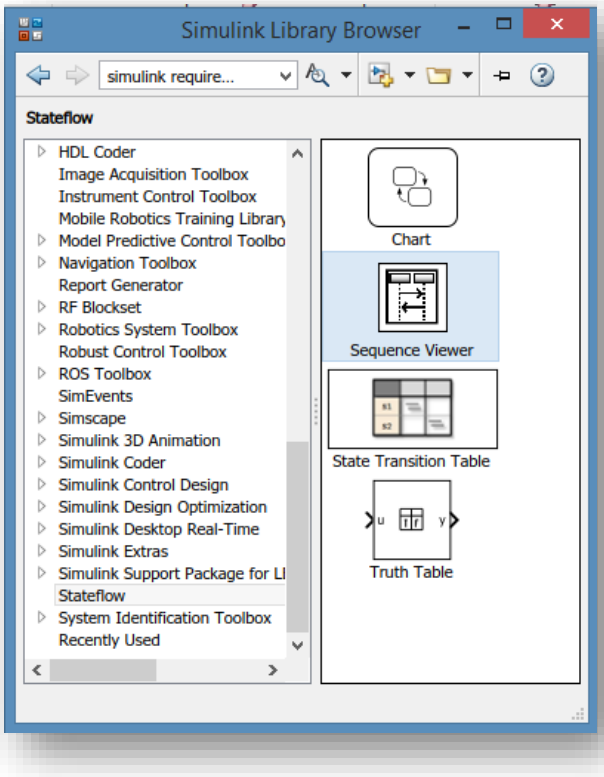
# Chapitre 6

# Le « Sequence Viewer»



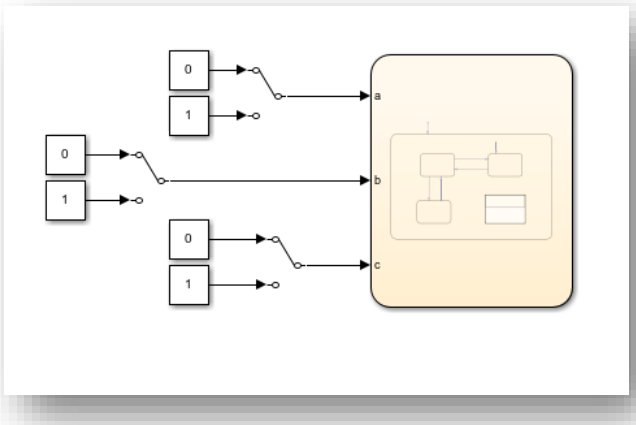
Sequence Viewer

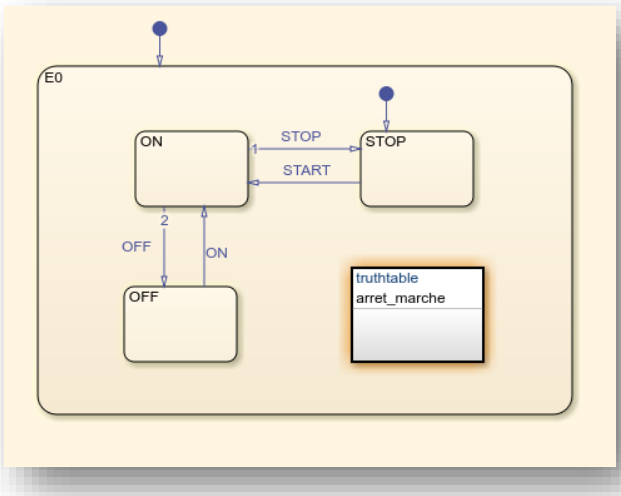
# Chapitre 6 - Le « Sequence Viewer »…




Le « **Sequence Viewer** » comme son nom l'indique permet de visualiser et d'analyser une séquence complète suite à une simulation.

Reprenons un exemple simple, celui du paragraphe 9, du chapitre 2, consacré à la table de vérité :

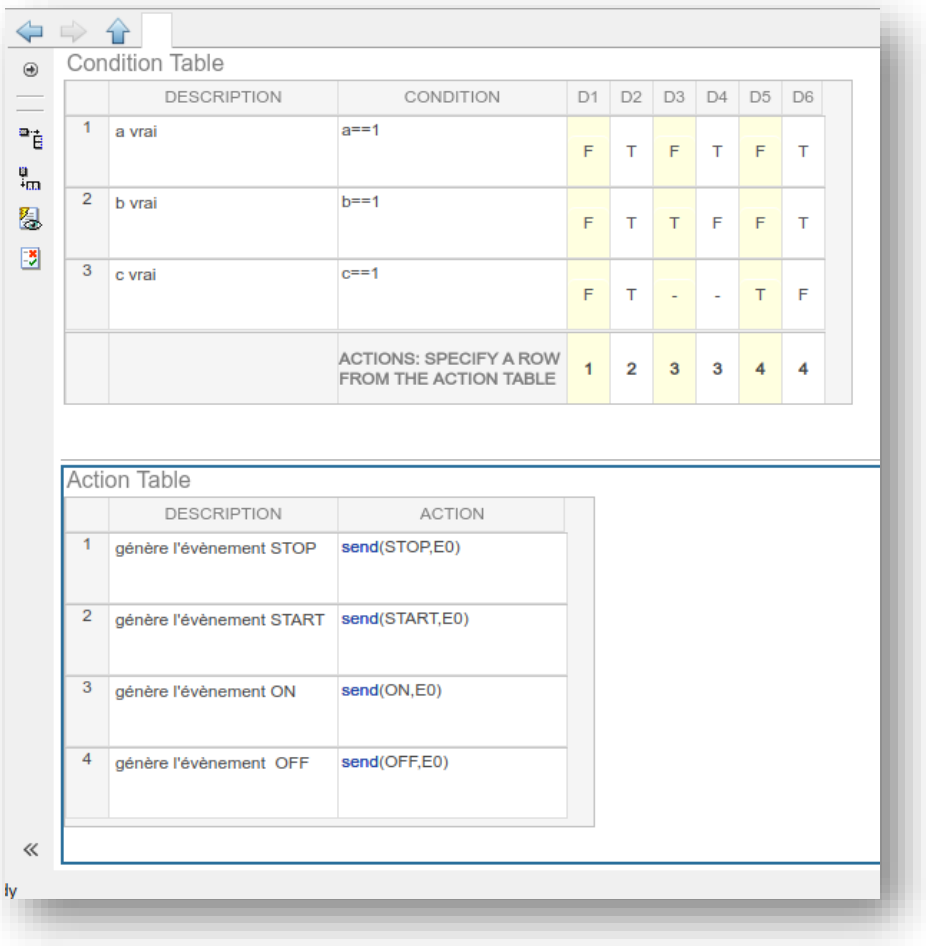


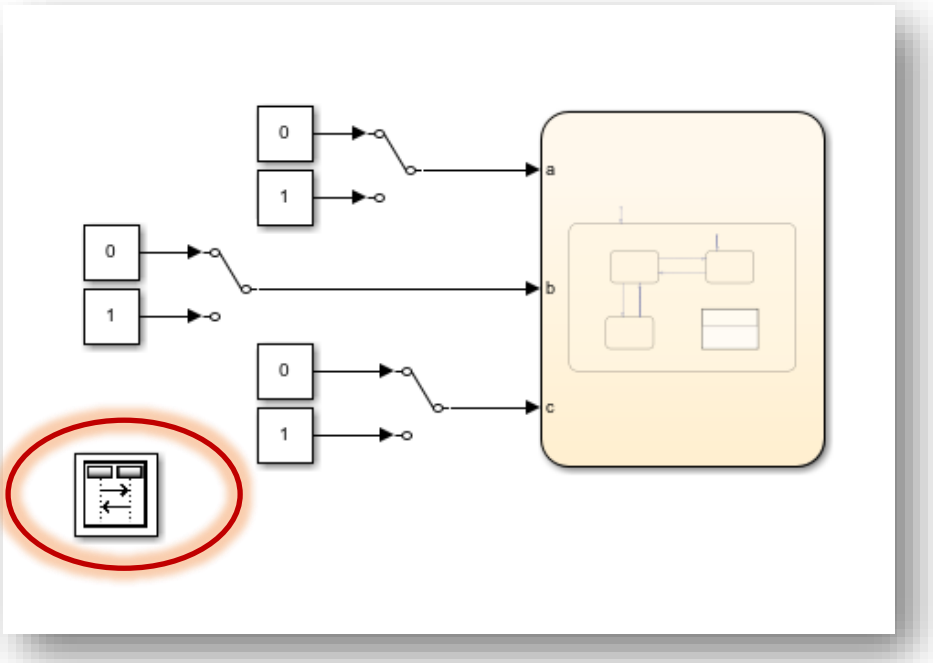

Nous allons réaliser la séquence suivante :

STOP → ON → OFF → ON → STOP

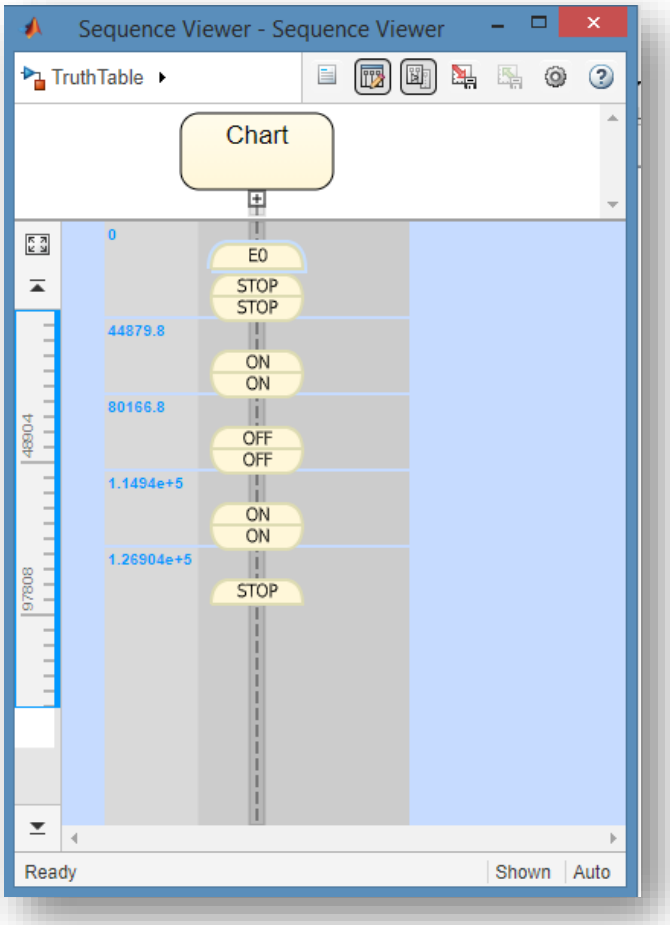
Le changement d'état aura lieu à l'occurrence des évènements, respectivement : START, OFF, ON, STOP. Ces évènements sont générés par la table de vérité.




Il convient maintenant de déposer un bloc « **Sequence Viewer** » :



Lancer la simulation en respectant la séquence choisie. Puis, ouvrir le bloc « **Sequence Viewer** » après la simulation :

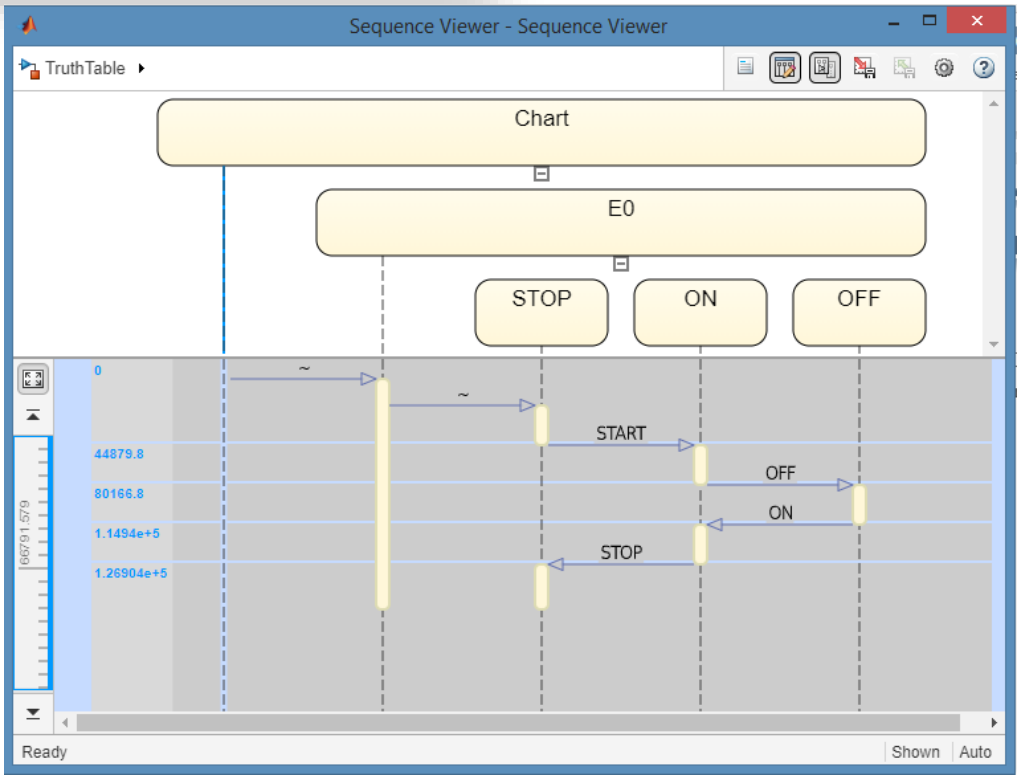



La fenêtre ci-contre s'ouvre et laisse apparaître la séquence.

Pour une meilleure lisibilité du diagramme, il est possible de cliquer sur le « + » situé au-dessous du Chart.

Le contenu du Chart est alors affiché et l'analyse de la séquence peut être menée.

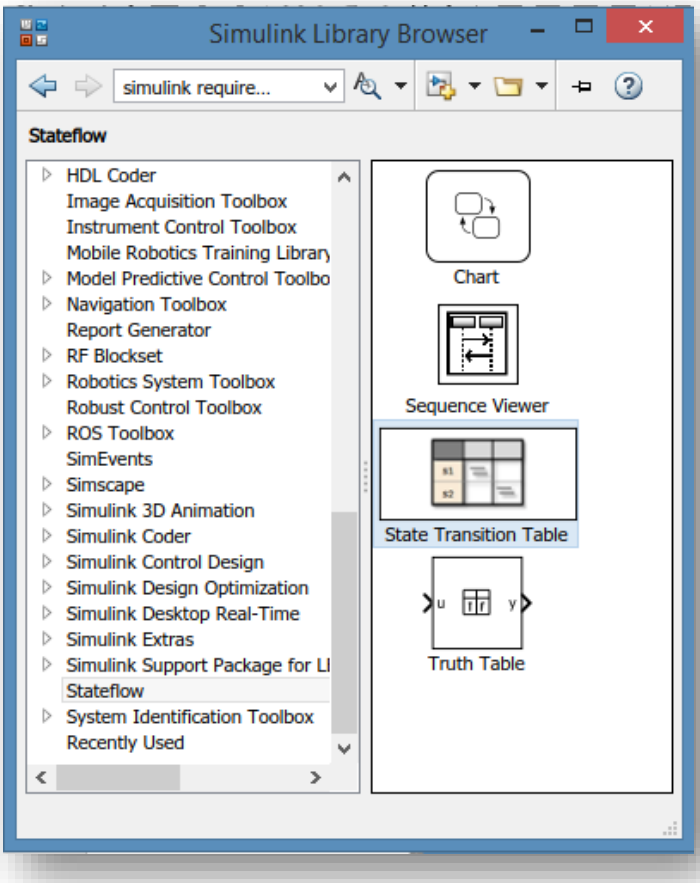
Le passage d'une ligne de vie à l'autre se fait à l'occurrence des évènements précités.



# Chapitre 7

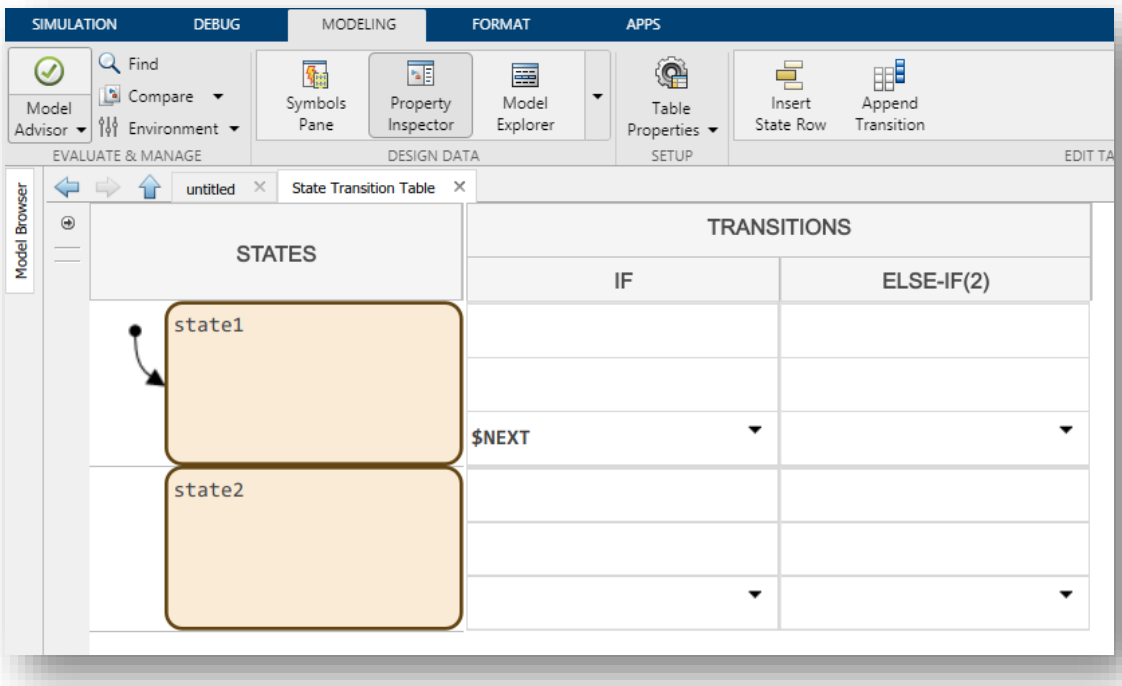
# La « State Transition Table»

# Chapitre 7 - La « State Transition Table »…



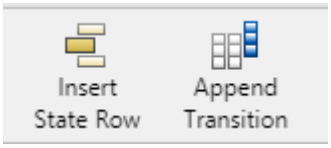

Intéressons-nous maintenant à la table état-transition.

Après un glissé-déposé, un double-clic sur le bloc « **State Transition Table** » ouvre la fenêtre suivante :

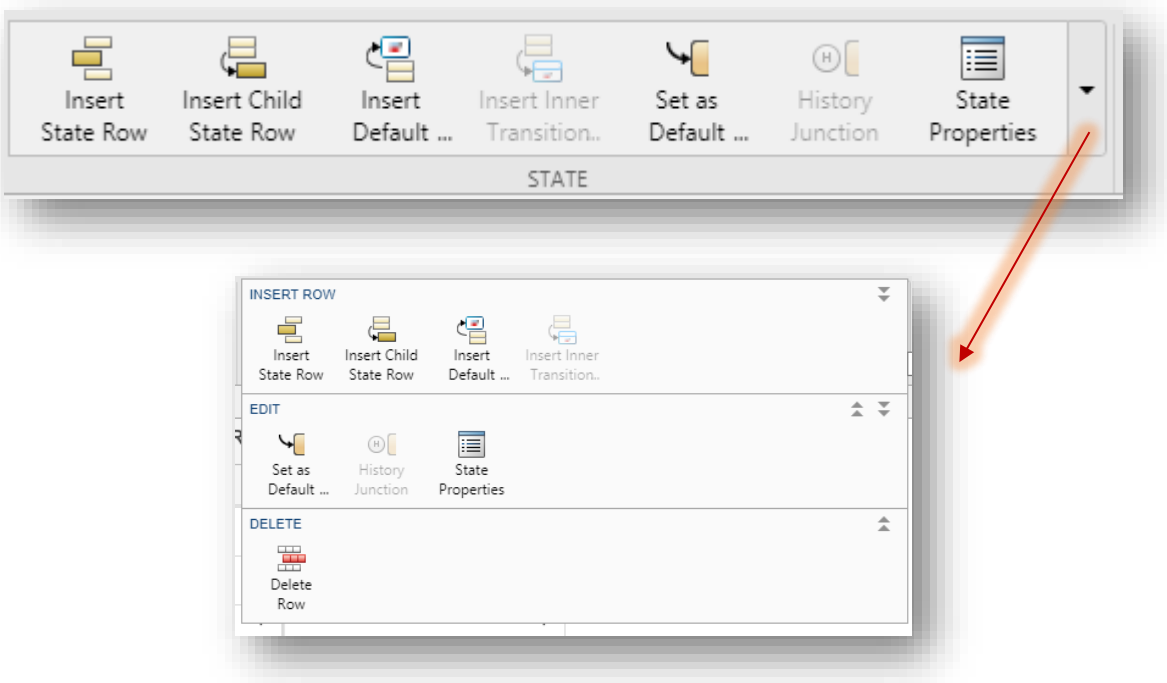


Par défaut, la table comporte deux lignes et trois colonnes. Dans la première colonne sont représentés les états. Dans la seconde et la troisième seront définies les transitions.

Il est possible d'ajouter des lignes et des colonnes en cliquant sur les icônes ci-contre du menu « **Edit Table** » de l'onglet « **MODELING** » du bandeau supérieur .

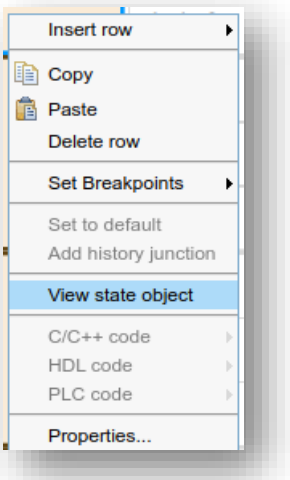



Lorsqu'on insère une ligne, le menu « **Edit Table** » change de nom et s'appelle « **STATE** » :

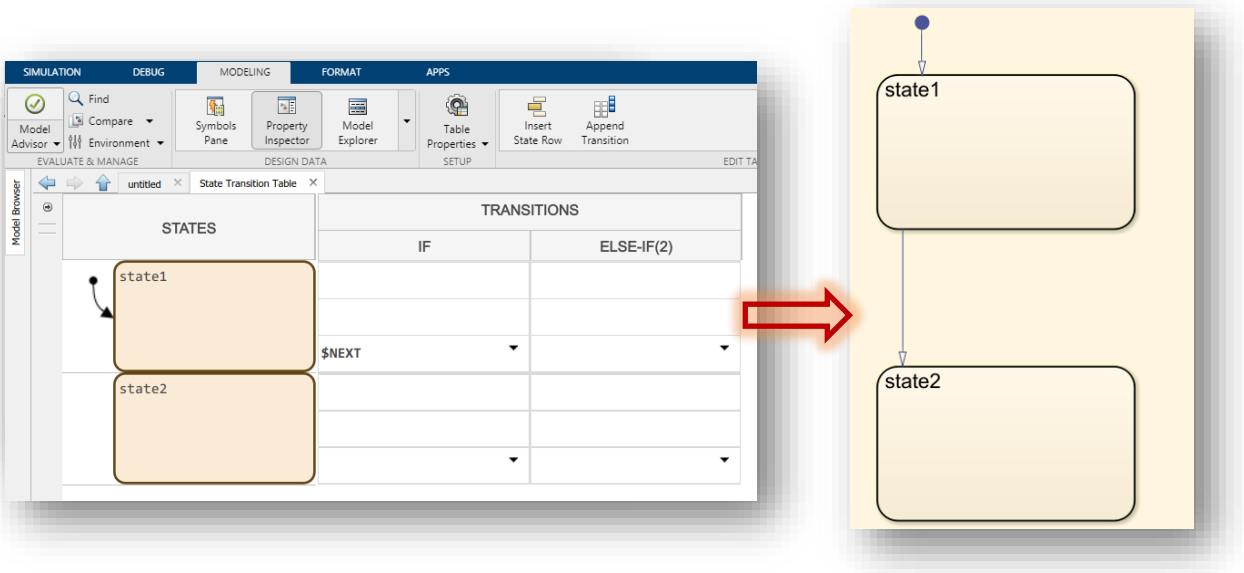



Il est possible de prendre en compte la hiérarchie des états insérés, en insérant par exemple un état enfant « **Insert Child State Row** ».

Un clic gauche sur un des états de la table ouvre le menu contextuel. La commande « **View State Object** » du menu permet la visualisation du diagramme tracé.

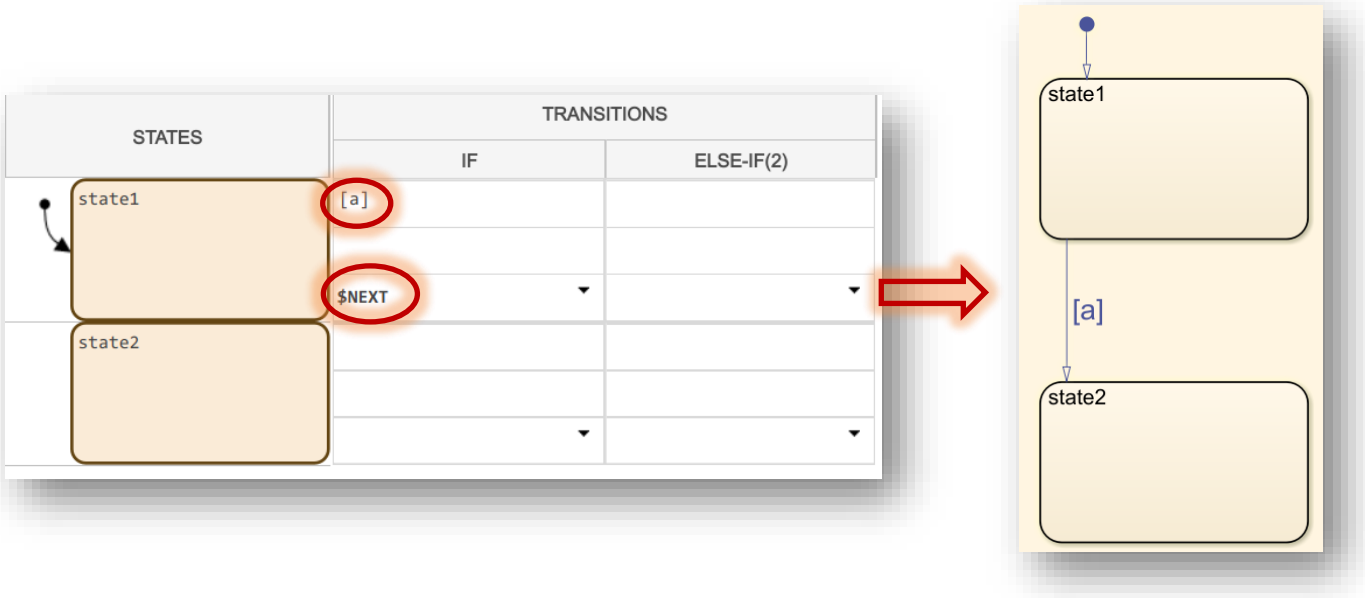


Par défaut :

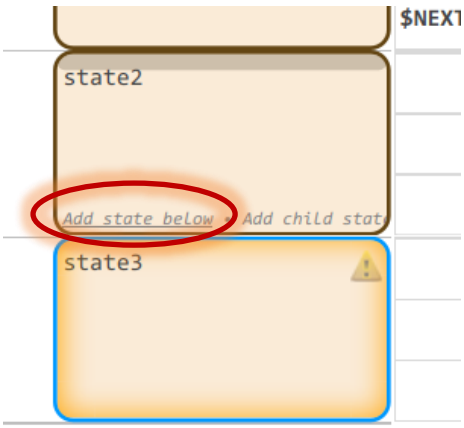


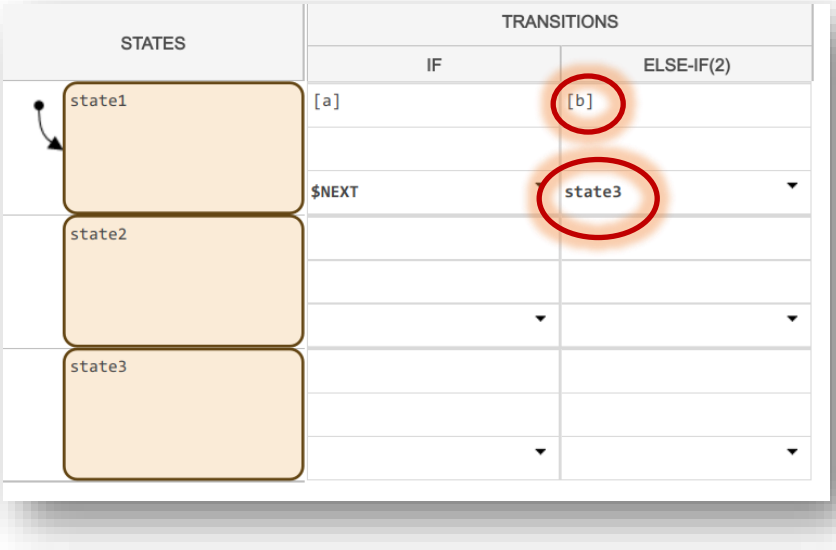

Pour ajouter une transition entre ces deux états, il faut remplir la case adjacente au premier état :

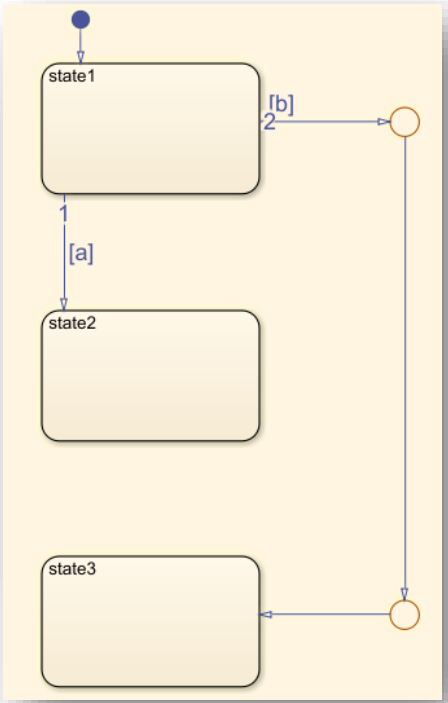
Si [a] est vraie, la transition pointe vers l'état suivant « **$NEXT** ».




Complétons cette table en introduisant une deuxième transition : si [a] est vraie alors la transition pointe l'état 2, sinon si [b] est vraie la transition pointe l'état 3 qu'il faut ajouter («**Insert State Row** ») ou en cliquant sur « **Add state below** » qui s'affiche en gris dans la partie basse de l'état State2 :

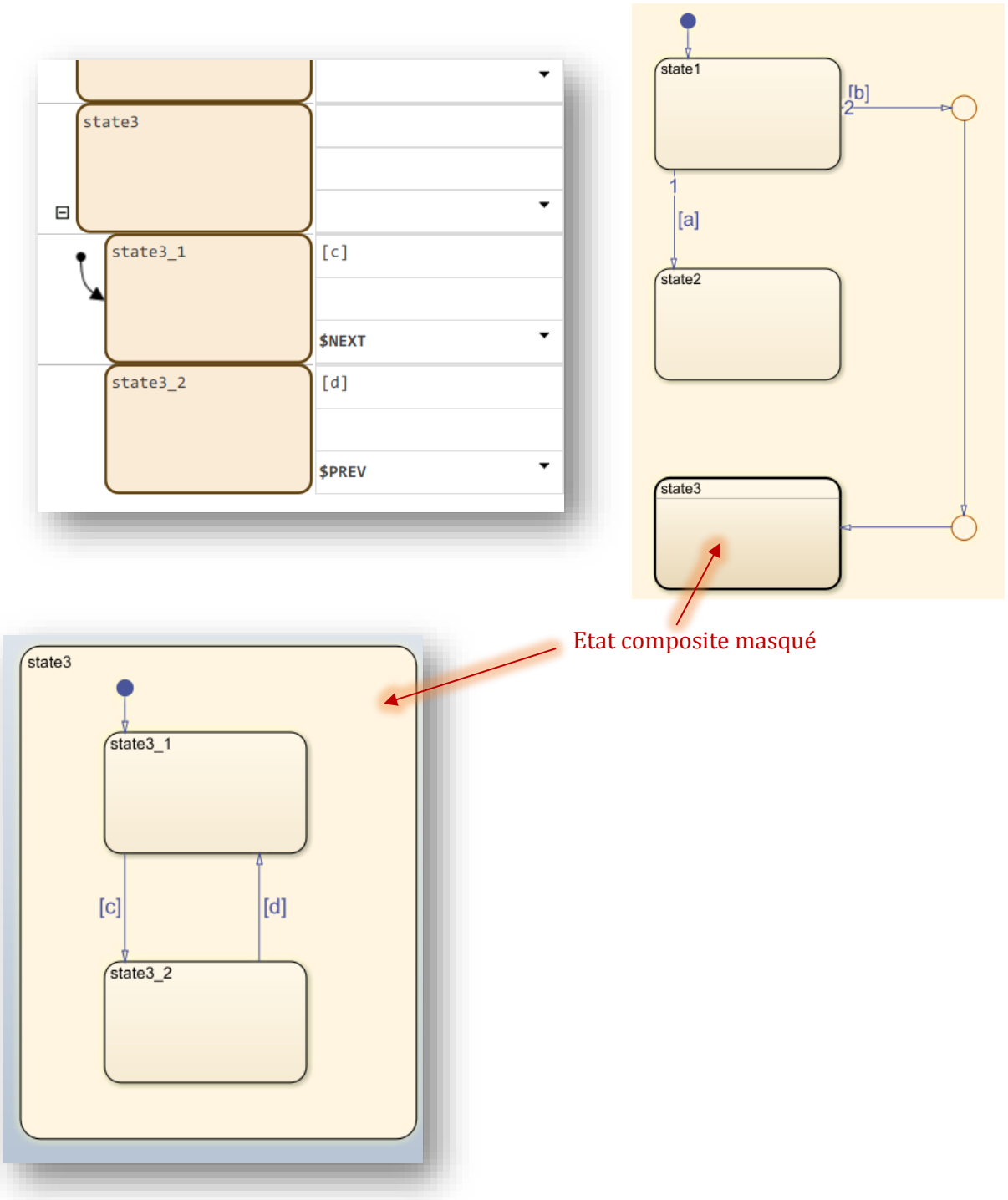


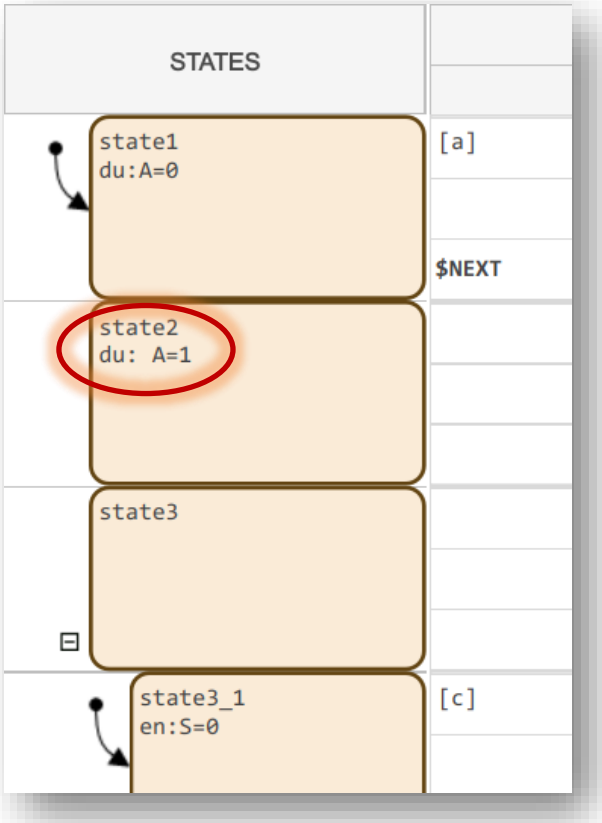



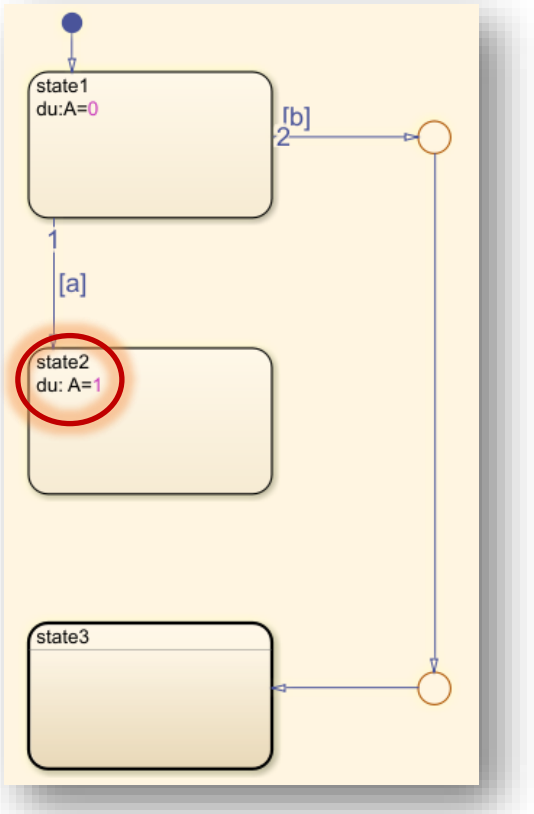

Pour construire un état composite, il faut introduire des états enfants en cliquant sur « **Insert Child State Row** » dans le menu « **STATE** » de l'onglet « **MODELING** » ou en cliquant sur « Add child state » qui s'affiche en gris dans la partie basse de l'état State3.

Insérer deux états enfants et les transitions qui suivent :




Les actions associées aux états se saisissent directement dans les états en utilisant les mots-clés correspondants :




Nous venons de voir succinctement les principes de base de la construction d'une table état-transition.

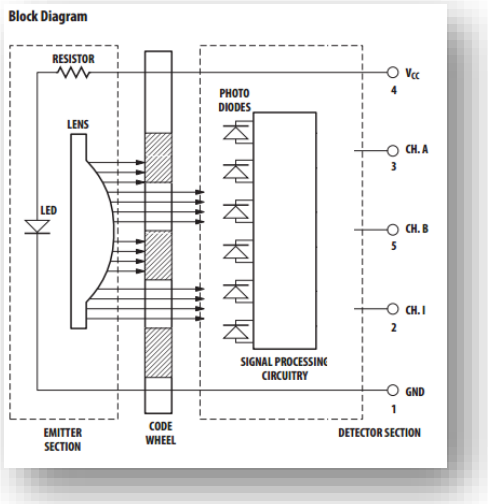
Concentrons-nous désormais sur l'application qui suit…

## 7.1- EXEMPLE DU CODEUR INCREMENTAL

Nous allons considérer le codeur incrémental AEDB-9140-F12 fabriqué par la société AVAGO Technologies.

Ce codeur possède trois voies : une voie A et une voie B en quadrature, une voie Z permettant le comptage des tours du disque.

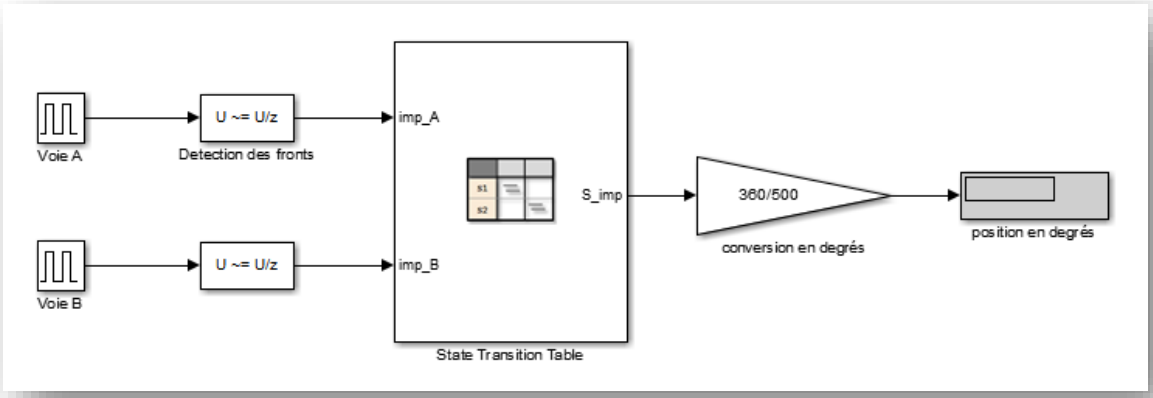
La résolution de ce codeur est de 500 points par tour.




L'objectif de cette application est de déterminer la position angulaire du disque en fonction des signaux A et B reçus. On considère que ces signaux sont des signaux carrés.

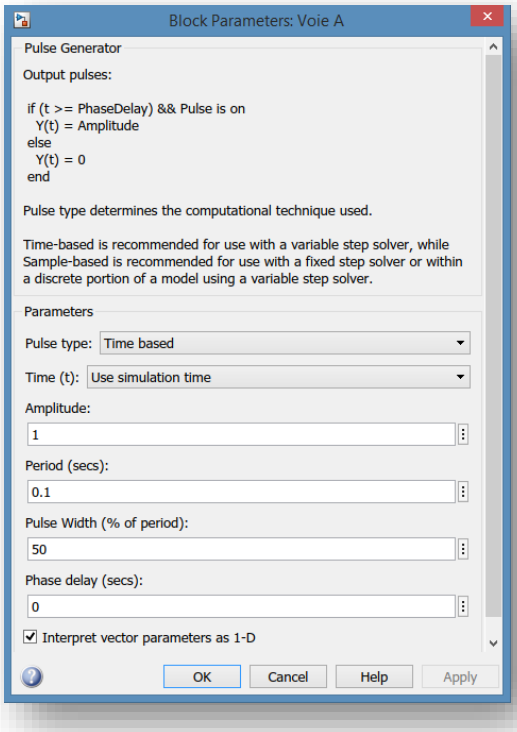
Le décompte du nombre de tours n'est pas abordé ici.

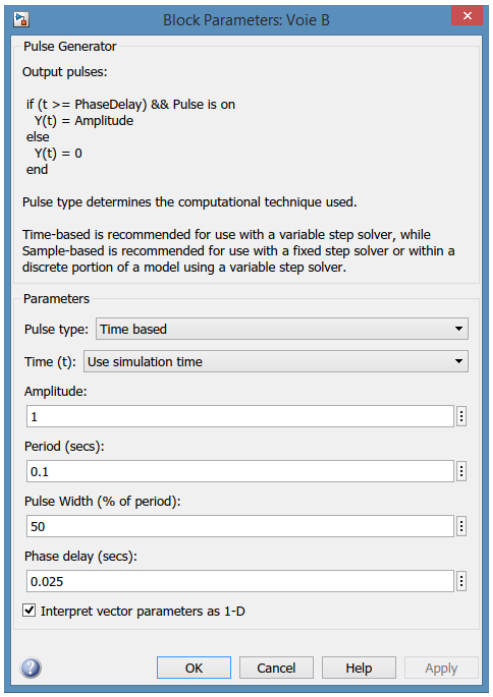
Préparons l'interface Simulink® :



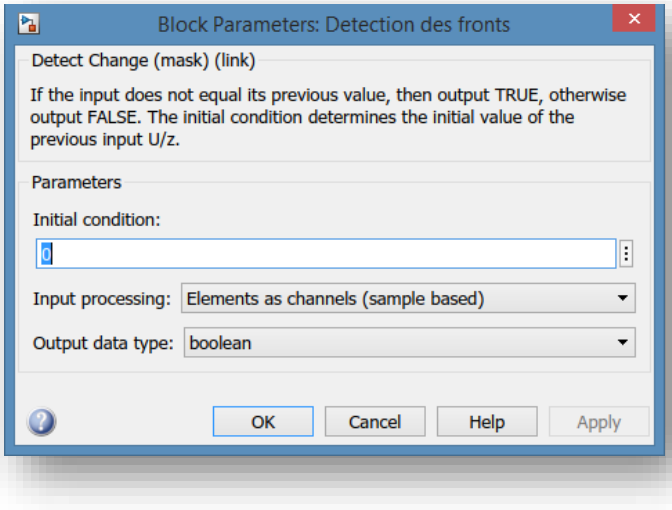

Ce modèle comprend deux signaux A et B en quadrature, des blocs permettant la détection des fronts montants et descendants sur ces signaux, la table état-transition, un convertisseur du nombre d'impulsions en degrés et l'affichage de l'angle de rotation du disque.

Définition des signaux A et B (bloc « **Pulse Generator** » dans le menu « **Sources** » de Simulink®) :



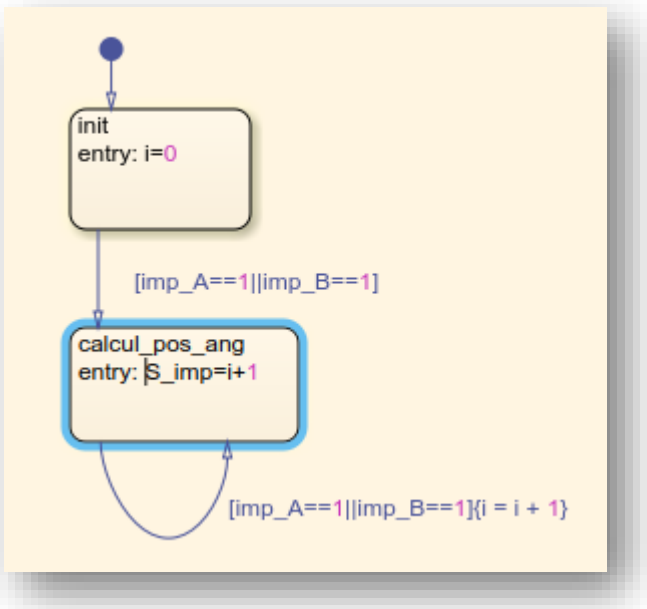




Les signaux ont une amplitude unitaire et une période de 0.1 s. Le signal B est déphasé de 0.025 s (« **Phase delay** »), soit un quart de période par rapport au signal A.




La détection des fronts (bloc « **Detect Change** » dans le menu « **Logic and Bit Operations** » de Simulink® ):

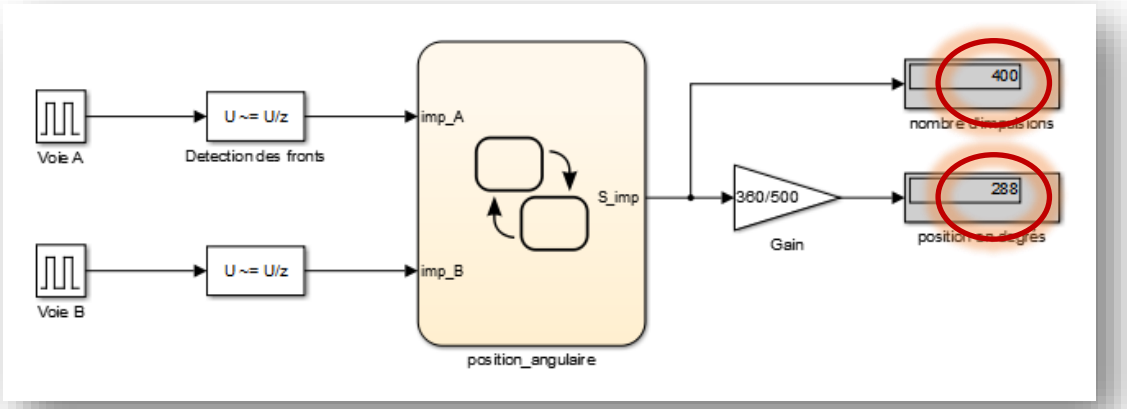
Pour la détermination de l'angle de rotation du disque on propose le diagramme état-transition suivant :




Il s'agit de mettre en place un compteur en introduisant la variable i qui est incrémentée à la détection d'un front sur A ou B.

Un calcul rapide permet de valider la simulation. En effet la période des signaux étant de 0.1 s une simulation sur 10 s génère 4 impulsions fois 100, soit 400 impulsions. En convertissant le nombre d'impulsions en degrés on trouve une rotation du disque qui vaut 288°.

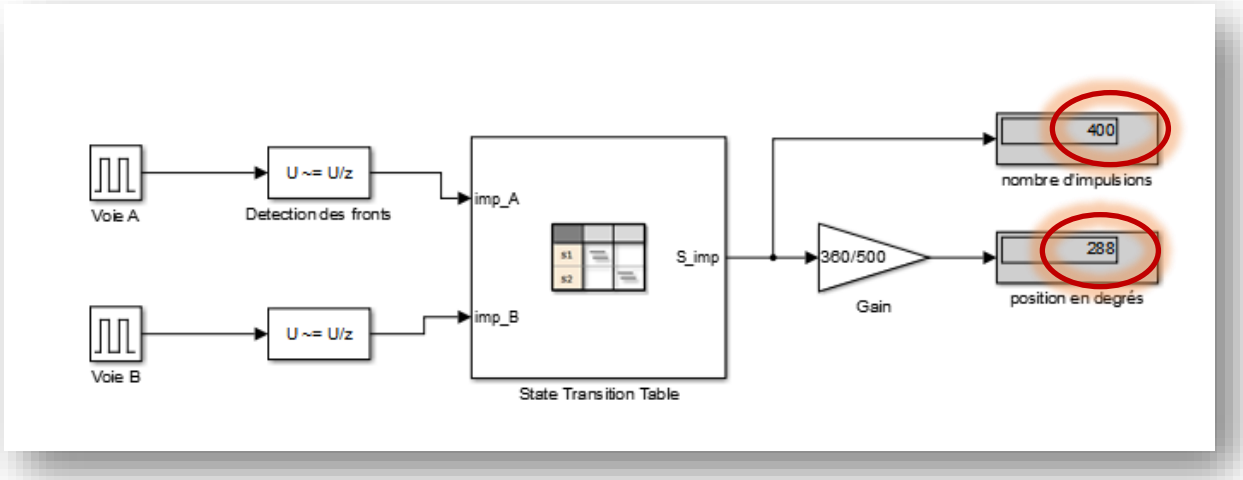
Ce que donne bien la simulation avec le diagramme précédent :

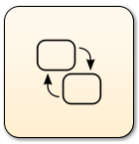



Mettons maintenant en place la table état-transition :

| STATES | TRANSITIONS | |
|---|---|---|
| | IF | ELSE-IF(2) |
| init<br>entry: i=0 | [imp_A==1\|\|imp_B==1] | |
| | | |
| | $NEXT | |
| calcul_pos_ang<br>entry: S_imp=i+1 | [imp_A==1\|\|imp_B==1] | |
| | {i = i + 1} | |
| | $SELF | |

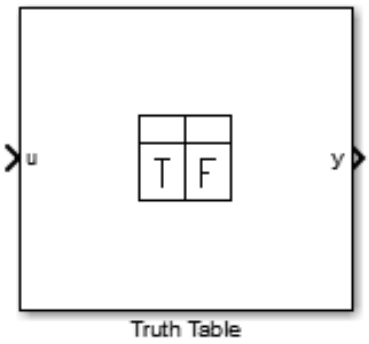
La simulation donne bien le résultat attendu :





# Chapitre 8

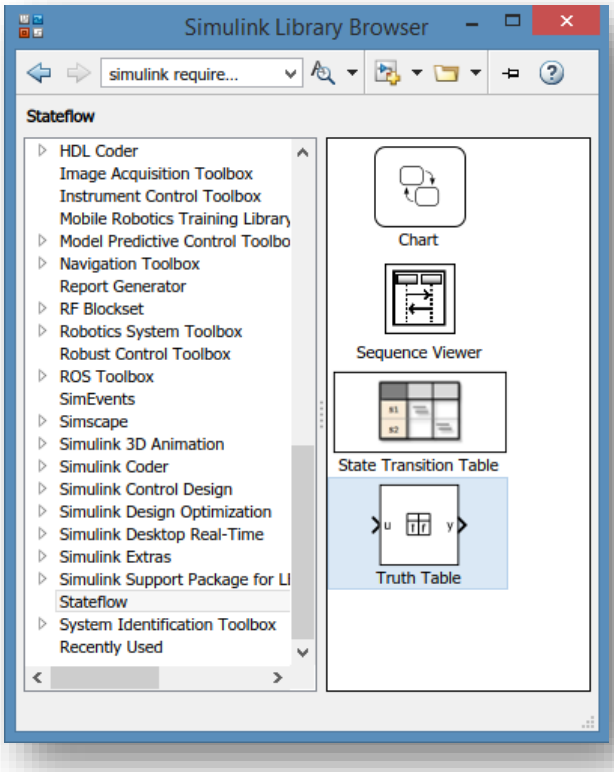
# La « Truth Table »



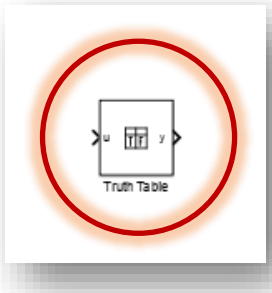
# Chapitre 8 - La « Truth Table »…

Si le système dont nous voulons simuler le comportement est un système combinatoire nous pouvons insérer dans le modèle Simulink® une table de vérité sans construire de machine d'état, c'est-à-dire sans créer de « **Chart** ».

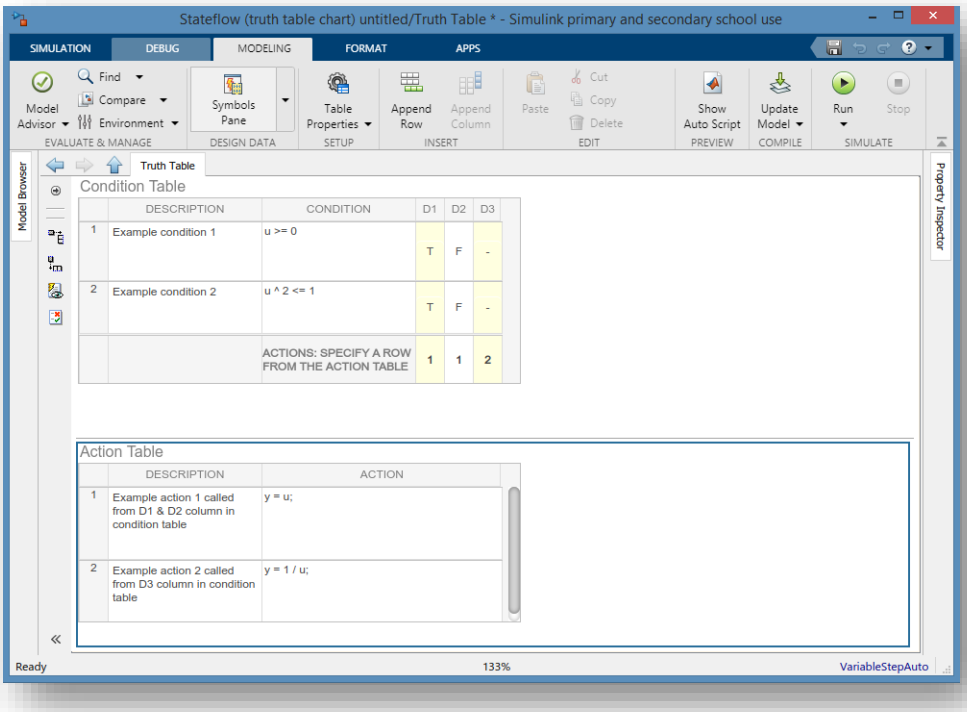
En effet dans la bibliothèque de Stateflow® apparait l'icône d'une table de vérité (« **Truth Table** »).




Un simple glissé-déposé fait apparaitre la table de vérité dans le modèle Simulink.




Un double clic sur le composant ouvre la table de vérité (elle est pré-remplie à titre d'exemple). Celle-ci est proche de celle dont il a été donné une description précédemment dans le chapitre 2.



C'est l'occasion de traiter une nouvelle application….

## 8.1- EXEMPLE DE LA COMMANDE D'UN PONT ROULANT

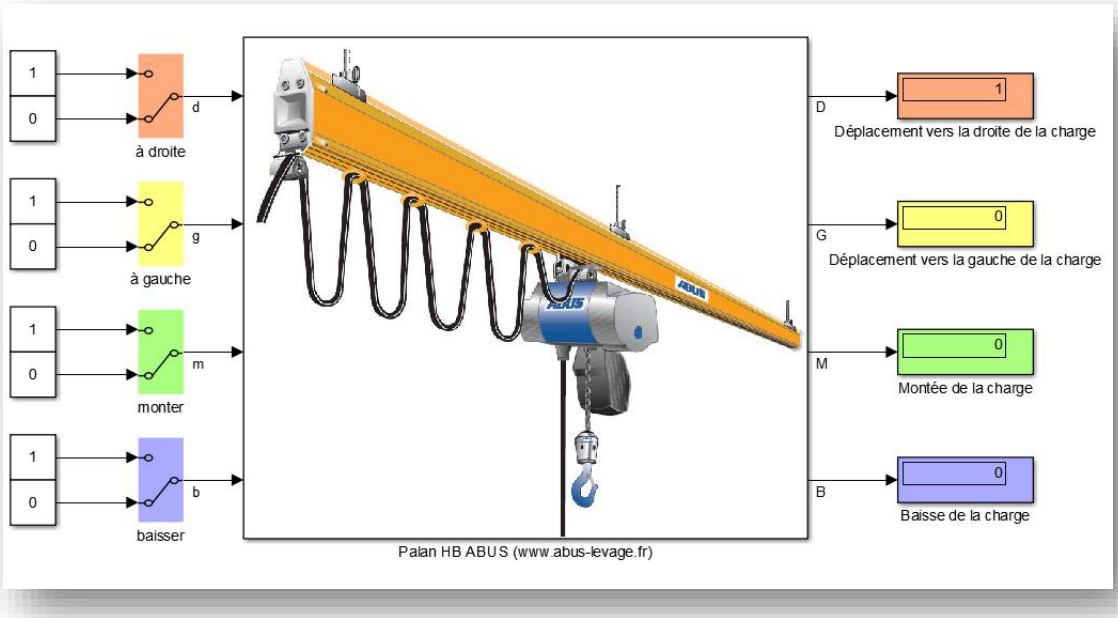
Nous allons construire un modèle de la commande d'un pont roulant pour le déplacement de faibles charges. (2t maxi)

La charge se déplace dans un plan vertical. Elle peut monter, descendre, se déplacer à droite, se déplacer à gauche.

Il s'agit d'un palan ABUS de la gamme HB (www.abus-levage.fr)

L'opérateur dispose d'une commande filaire à quatre boutons pour déplacer la charge.

Nous allons donc élaborer étape par étape le modèle suivant :




Dans cet exercice nous ne prenons pas en compte l'effet des capteurs fin de course haut, bas, droite et gauche. Vous pourrez ultérieurement compléter le modèle si vous le souhaitez.

Les déplacements de la charge sont assurés par une commande filaire à quatre boutons :

- « **d** », commande le sens D,
- « **g** », commande le sens G,
- « **m** », commande la montée M,
- « **b** », commande la descente B.

Des conditions particulières sont envisagées :

- si par erreur on actionne simultanément « **d** » et « **g** » la priorité est accordée au sens gauche G,
- si par erreur on actionne simultanément « **m**» et « **b** » la priorité est accordée à la montée de la charge,
- si les quatre boutons sont appuyés toutes les commandes sont annulées.

La table de vérité ci-contre définit l'ensemble les conditions d'exploitation du pont roulant.

Le codage binaire réfléchi est ici adopté pour faciliter la simulation qui suivra. En effet, un seul « **Switch** » sera manipulé à chaque changement de ligne de la table au cours de la phase de test.

| | d | g | m | b | D | G | M | B |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 8 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 9 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 11 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 12 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 13 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 14 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 15 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 16 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

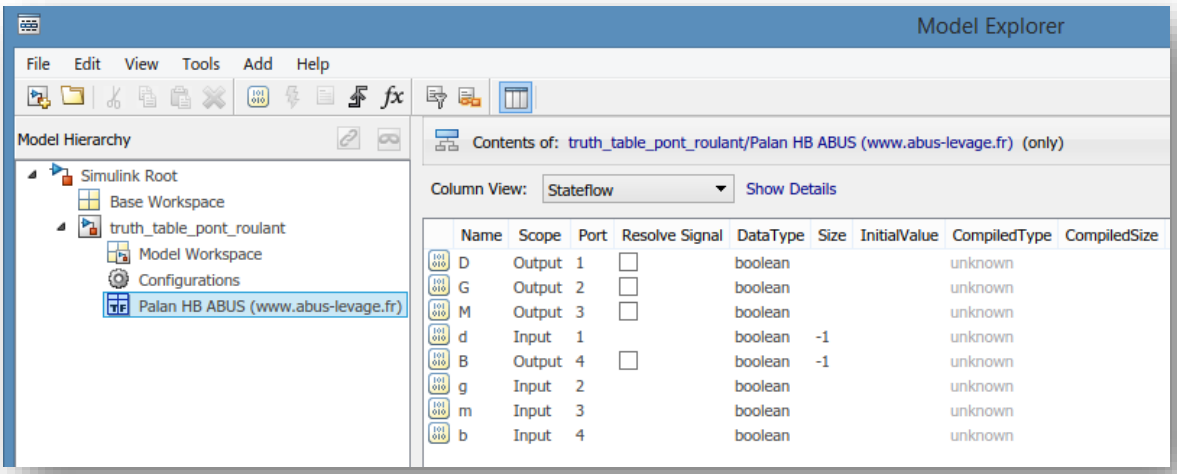
En lisant la table de vérité qui est donnée, nous pouvons nous apercevoir qu'à des combinaisons des variables d'entrées différentes peuvent correspondre une même action. C'est le cas par exemple pour les lignes 8 et 9 ou 1 et 11… Cette remarque devra être prise en compte au moment de la création de la table de vérité sous Stateflow®.

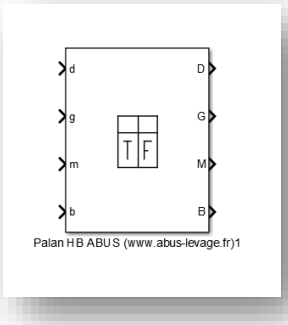
Préparons tout d'abord l'interface Simulink®.

La table de vérité dispose de quatre variables d'entrée et de quatre variables de sortie pouvant prendre la valeur 0 ou 1.

A partir de l'explorateur de modèle il convient d'ajouter ces entrées et ces sorties au bloc préalablement mis en place dans le modèle.

L'explorateur est accessible à partir de l'onglet « MODELING » dans le bandeau supérieur.

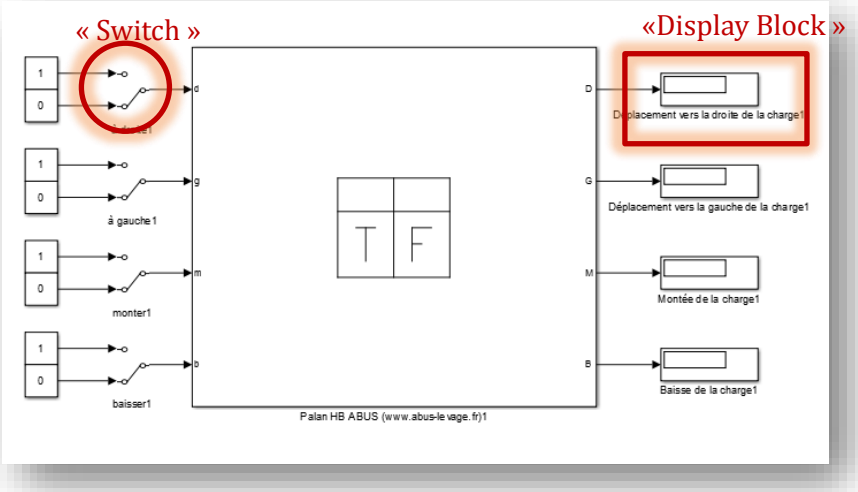


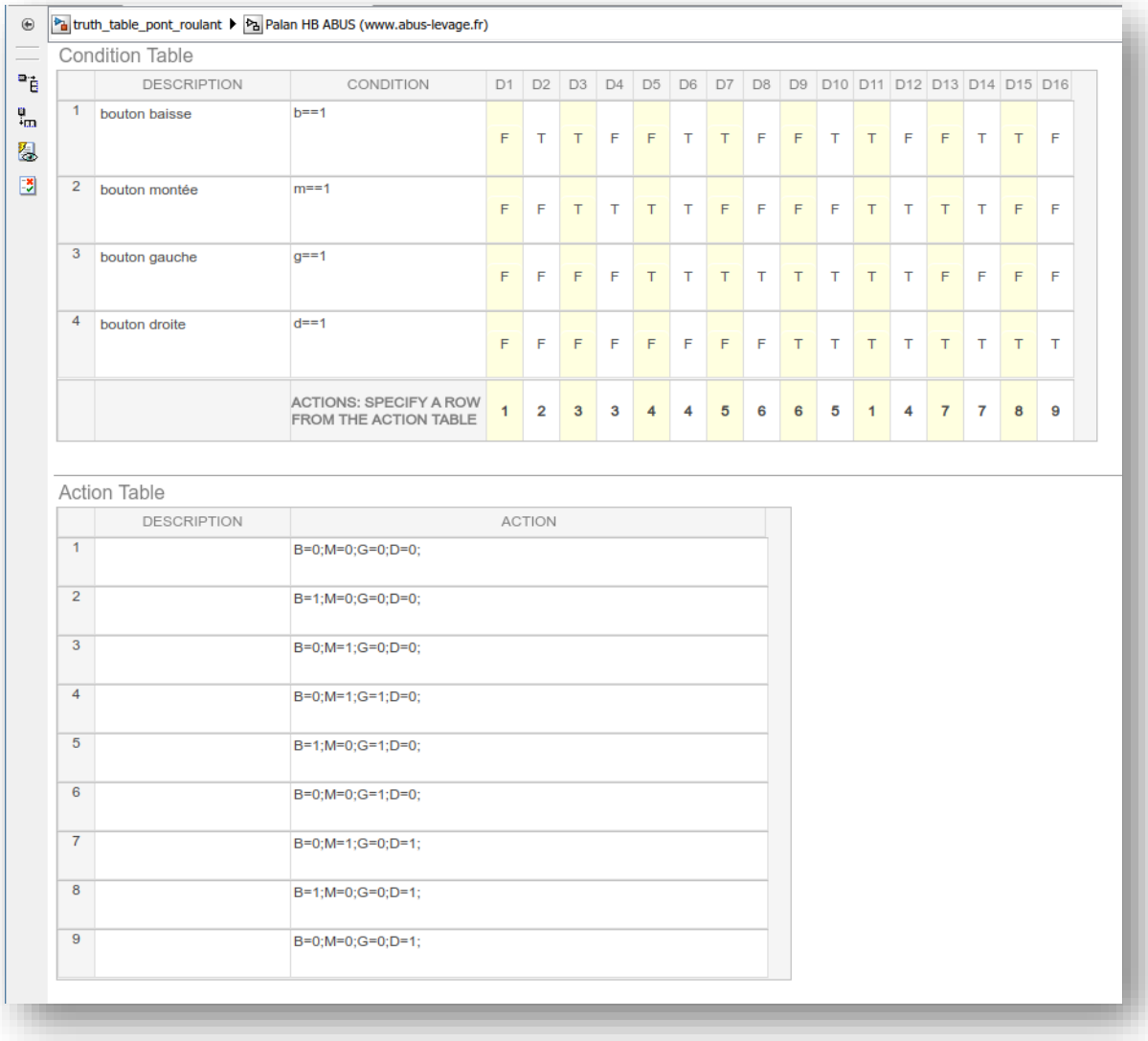

On rappelle que l'ajout de données (« **data** ») se fait en cliquant sur l'icône « **add data** » au milieu du bandeau supérieur de l'explorateur.

On vérifie bien que les entrées et les sorties ont été créées.

On ajoute ensuite les quatre « **Switches** » (bibliothèque « **Signal Routing** » de Simulink®) et les quatre « **Display Blocks** » (bibliothèque « **Sinks** » de Simulink®) qui afficheront les valeurs de leur entrée.



Une fois l'interface Simulink® prête nous nous intéressons à la table de vérité :


truth_table_pont_roulant ▸ Palan HB ABUS (www.abus-levage.fr)

Condition Table

| | DESCRIPTION | CONDITION | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 | D12 | D13 | D14 | D15 | D16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | bouton baisse | b==1 | F | T | T | F | F | T | T | F | F | T | T | F | F | T | T | F |
| 2 | bouton montée | m==1 | F | F | T | T | T | T | F | F | F | F | T | T | T | T | F | F |
| 3 | bouton gauche | g==1 | F | F | F | F | T | T | T | T | T | T | T | T | F | F | F | F |
| 4 | bouton droite | d==1 | F | F | F | F | F | F | F | F | T | T | T | T | T | T | T | T |
| | | ACTIONS: SPECIFY A ROW FROM THE ACTION TABLE | 1 | 2 | 3 | 3 | 4 | 4 | 5 | 6 | 6 | 5 | 1 | 4 | 7 | 7 | 8 | 9 |

Action Table

| | DESCRIPTION | ACTION |
|---|---|---|
| 1 | | B=0;M=0;G=0;D=0; |
| 2 | | B=1;M=0;G=0;D=0; |
| 3 | | B=0;M=1;G=0;D=0; |
| 4 | | B=0;M=1;G=1;D=0; |
| 5 | | B=1;M=0;G=1;D=0; |
| 6 | | B=0;M=0;G=1;D=0; |
| 7 | | B=0;M=1;G=0;D=1; |
| 8 | | B=1;M=0;G=0;D=1; |
| 9 | | B=0;M=0;G=0;D=1; |

Cette table contient :

- les quatre conditions b = 1, m = 1, g = 1, d = 1,
- seize décisions (repérées de D1 à D16) correspondant aux seize lignes de la table de vérité donnée,
- neuf actions (repérées de 1 à 9).

**Rappel sur la lecture de la table** : à une combinaison des entrées, par exemple la décision D5 (F, T, T, F), correspond une action, ici l'action 4, qui consiste à affecter les valeurs 0 à B, 1 à M, 1 à G et 0 à D.

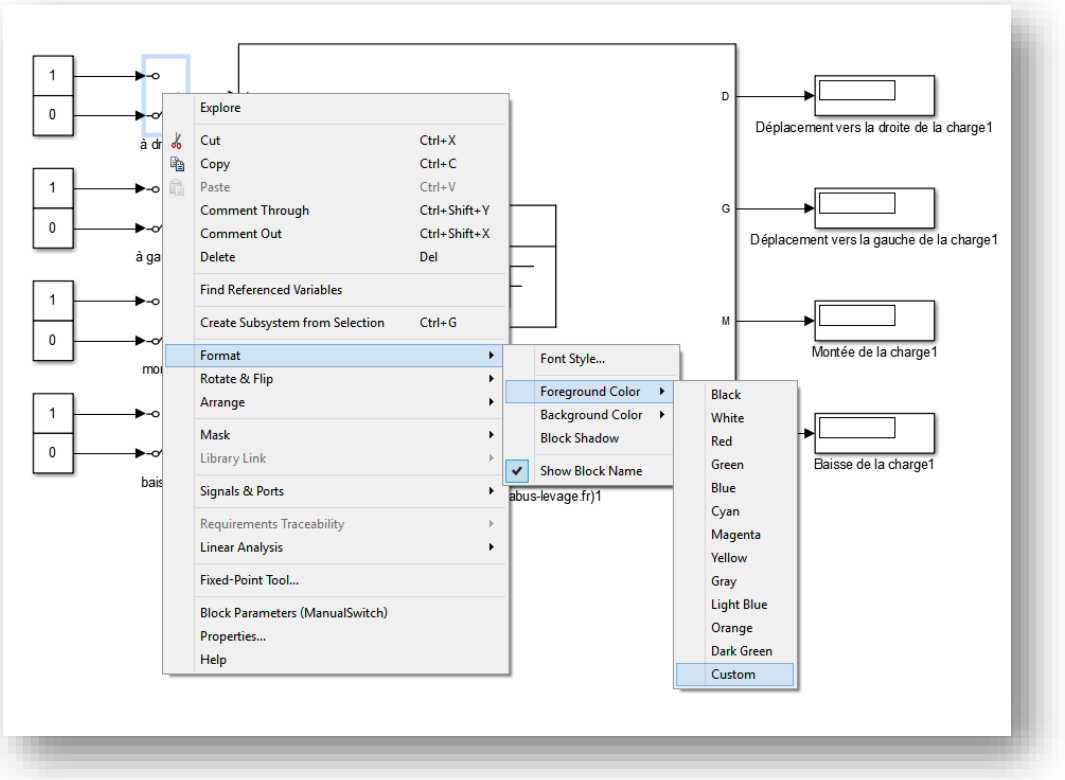
La table de vérité ainsi remplie permet de lancer une simulation et de vérifier la validité du modèle.

Choisir un temps de simulation infini et double-cliquer sur les « **Switches** » pendant la simulation pour valider le comportement du pont roulant.

Pour améliorer la présentation graphique du modèle il est possible d'ajouter de la couleur aux éléments ou de créer un masque sur la table de vérité et y ajouter une photo représentative du système modélisé.

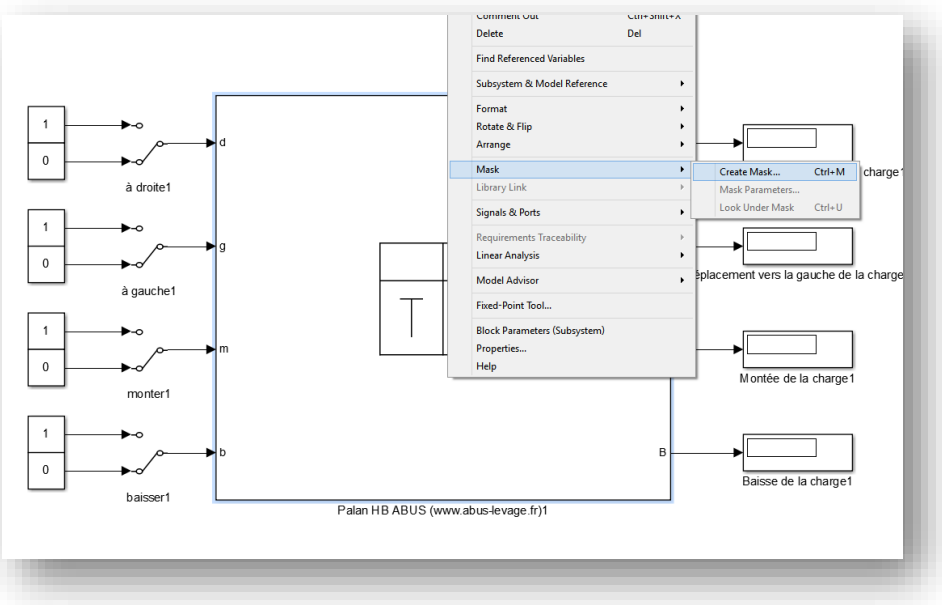
*Pour ajouter de la couleur :*

Sélectionner le bloc à colorier, afficher le menu contextuel par un clic droit, puis « **Format** », « **Background Color** » . Choisir une des couleurs proposées ou bien personnalisée la en cliquant sur « **Custom** ».

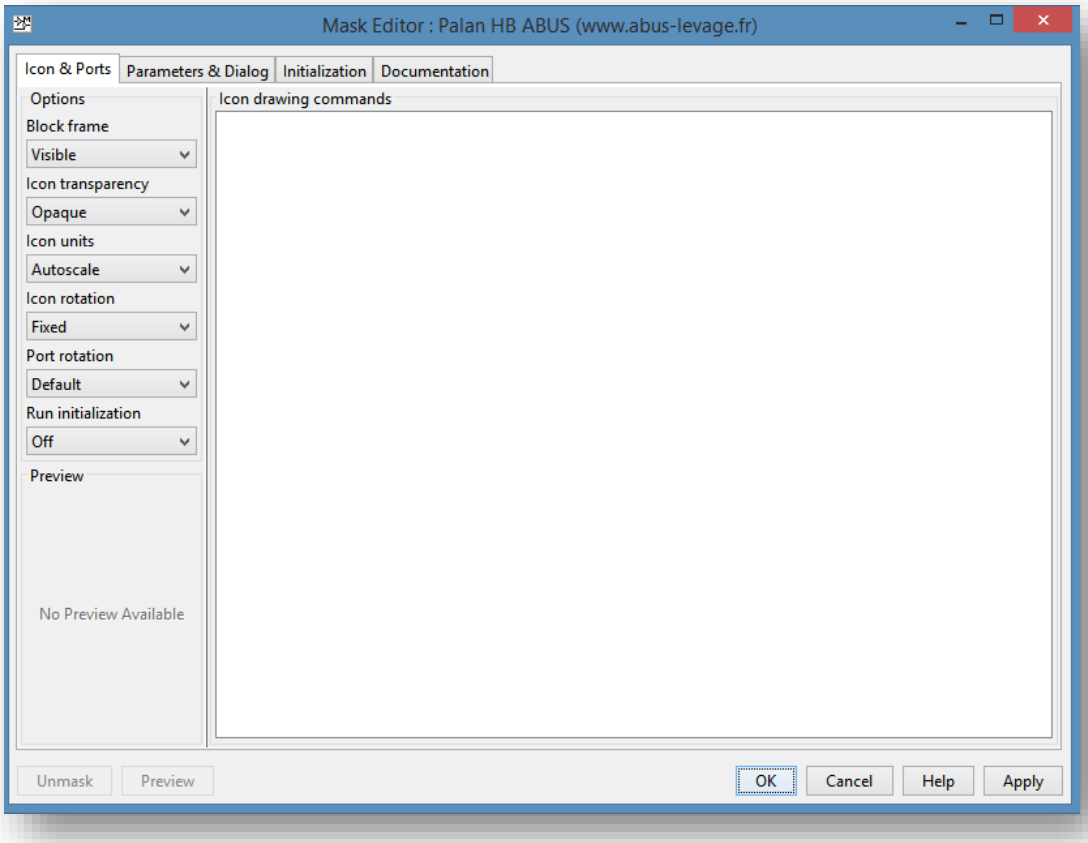



Pour créer un masque sur la table de vérité et y insérer une photo :

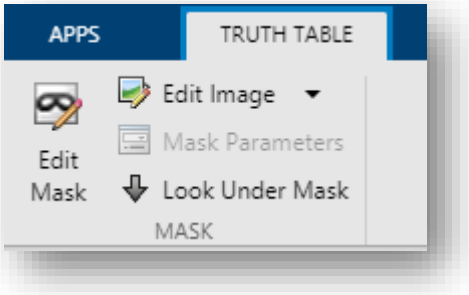
Sélectionner la table de vérité, puis clic droit, « **Mask** », « **Create Mask…** »




La fenêtre suivante s'ouvre. Il s'agit de l'éditeur de masque (« **Mask Editor** »)



Un autre moyen d'ouvrir l'éditeur de masque est de cliquer sur la table de vérité puis sur l'onglet « **TRUTH TABLE** » qui apparaît alors dans le bandeau supérieur. Dans le menu « **MASK** » de cet onglet, cliquer sur le bouton « **Edit Mask** »

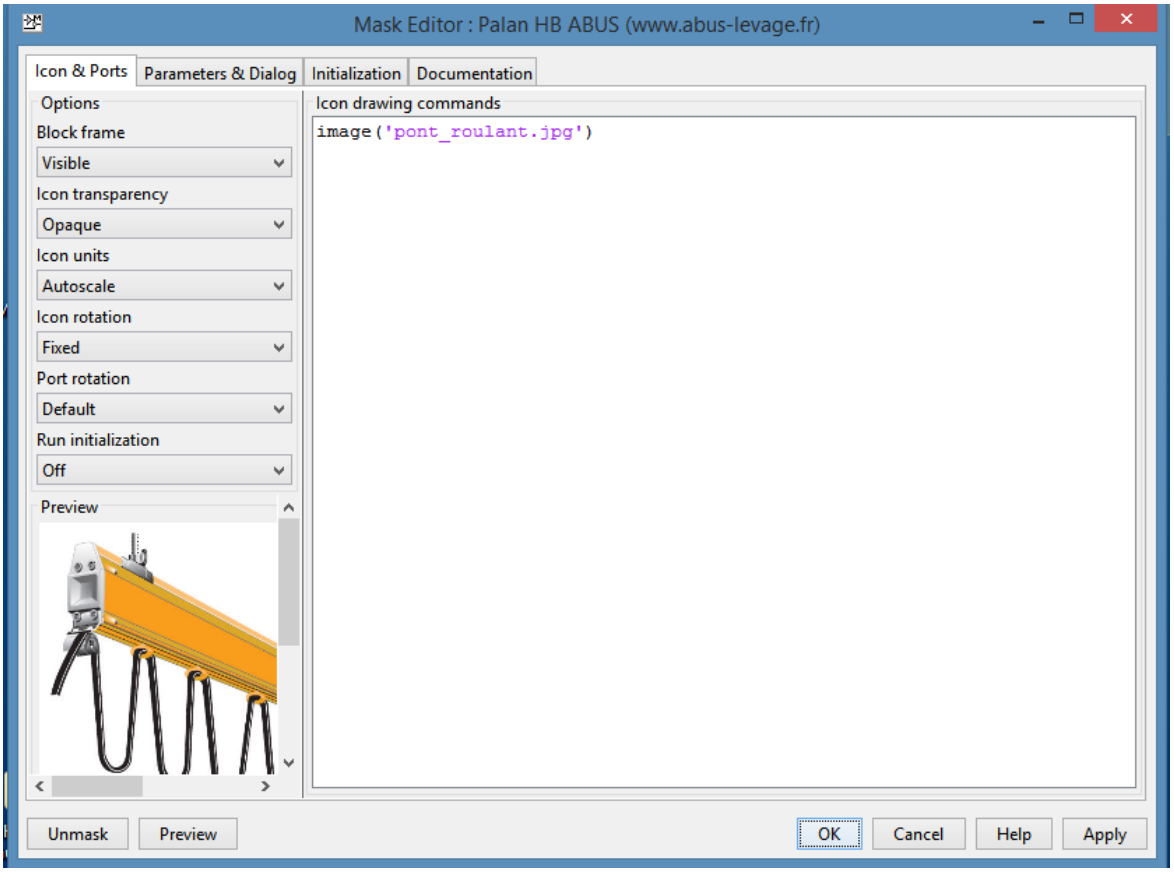



La syntaxe qui permet l'affichage d'une image est la suivante :

```
image('pont_roulant.jpg')
```

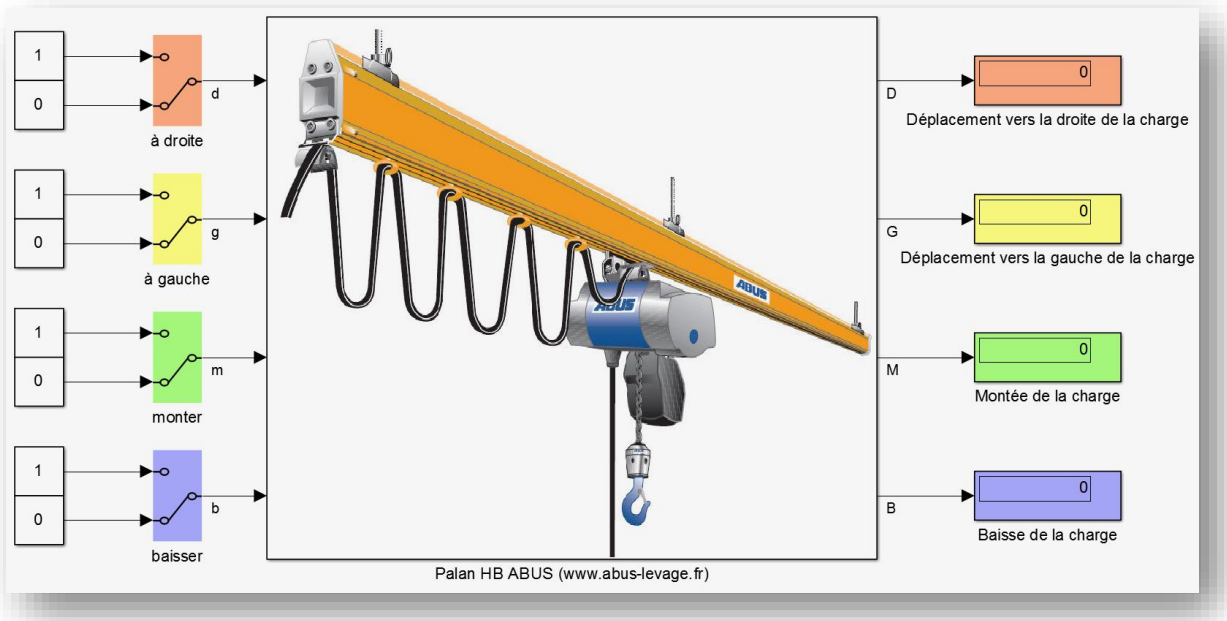
Le fichier de l'image est nommé « **pont_roulant** ».

**Attention** : pour que l'image s'affiche, ce fichier doit obligatoirement être présent dans le « **Path** » c'est-à-dire à l'endroit au le fichier Simulink® est sauvegardé.



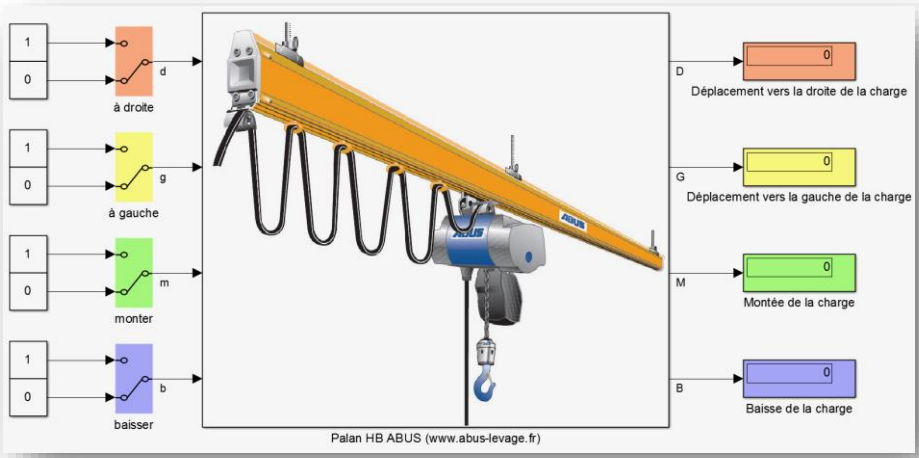

L'image apparait alors en bas à droite de l'éditeur puis doit apparaitre dans le bloc en cliquant sur le bouton « **Apply** ».

Pour supprimer le masque, il faut l'éditer et cliquer sur le bouton « **Unmask** » en bas à gauche de la fenêtre.

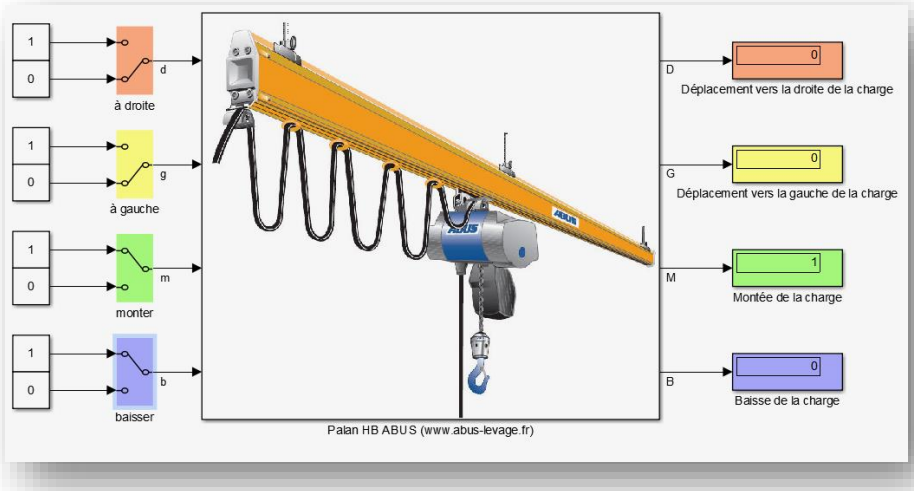



Validation du modèle :

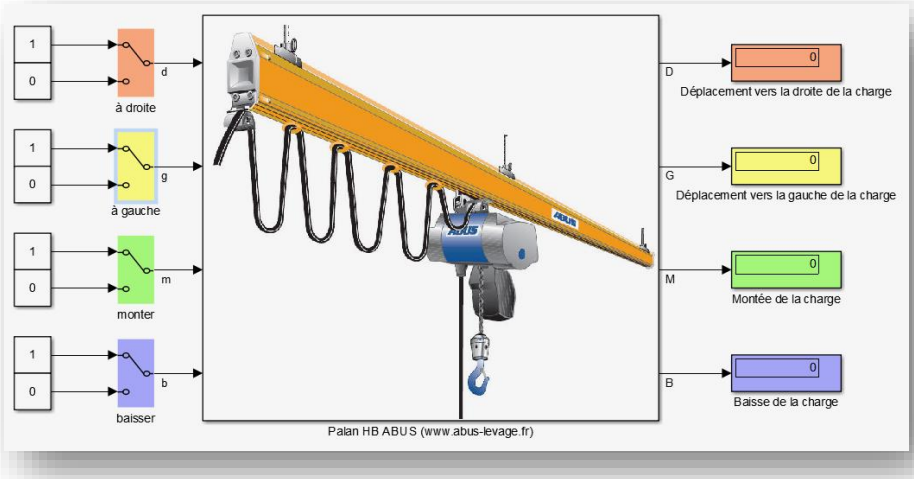
Ligne 1 de la table de vérité :



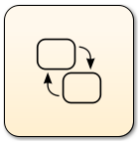
Ligne 3 de la table de vérité :




Ligne 11 de la table de vérité :

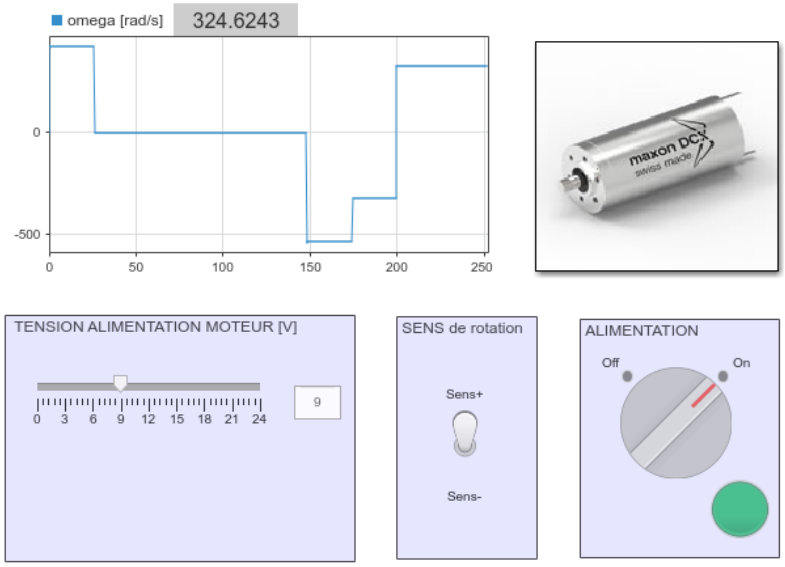



Etc…



# Chapitre 9

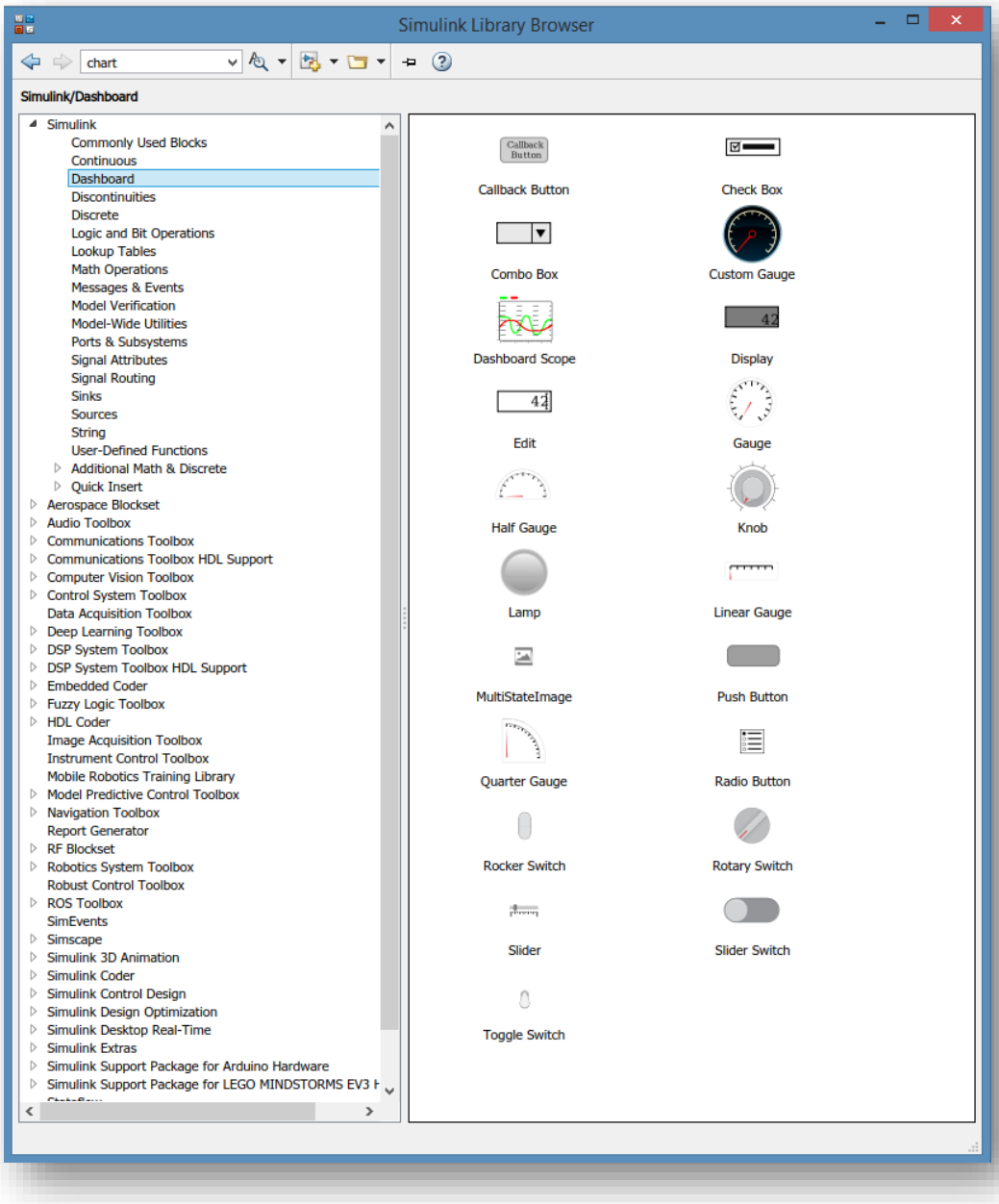
# Le « Dashboard » Simulink®



# Chapitre9 – Le « Dashboard » Simulink®…

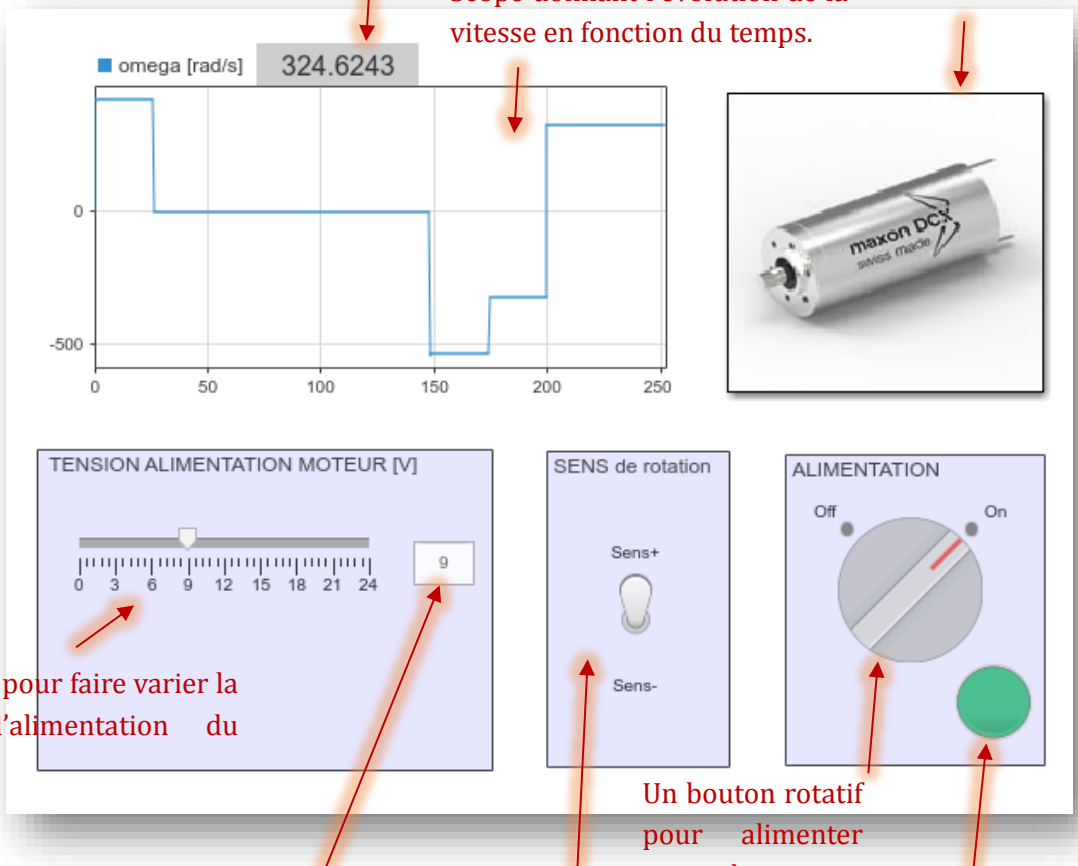
## 9.1 EXEMPLE DU MOTEUR A COURANT CONTINU

Le « Dashboard » est une bibliothèque qui comprend des boutons, des curseurs, des cadrans, des scopes, des voyants etc… qui peuvent améliorer grandement l'interface de pilotage d'un modèle.




Reprenons l'exemple du moteur à courant continu.

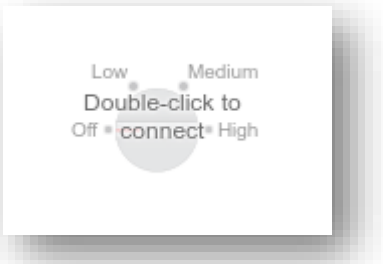
L'idée est de construire une interface qui doit présenter les éléments suivants :



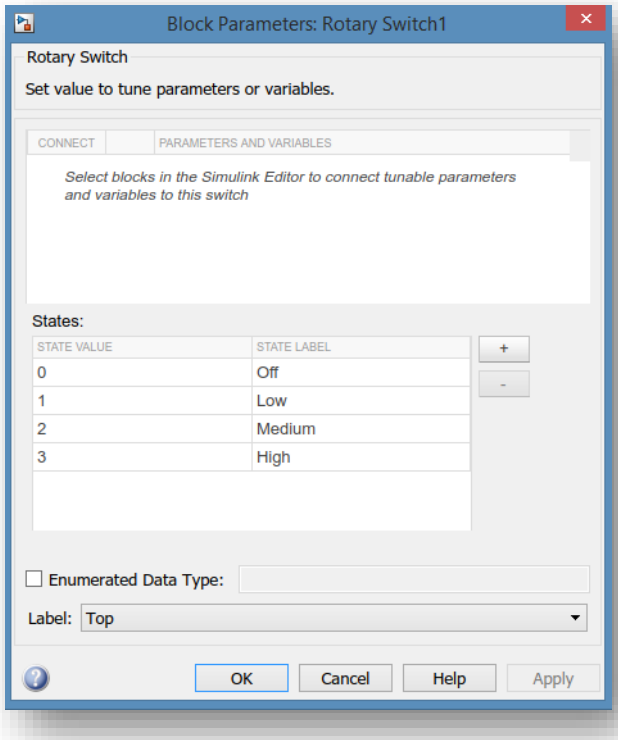

Pour ce faire il faut connecter les éléments du « Dashboard » aux éléments du modèle correspondant. Nous allons voir comment créer le bloc « **ALIMENTATION** » de l'interface, les autres blocs se construisant de manière identique.

Commençons par mettre en place le bouton rotatif : « **Rotary Switch** »

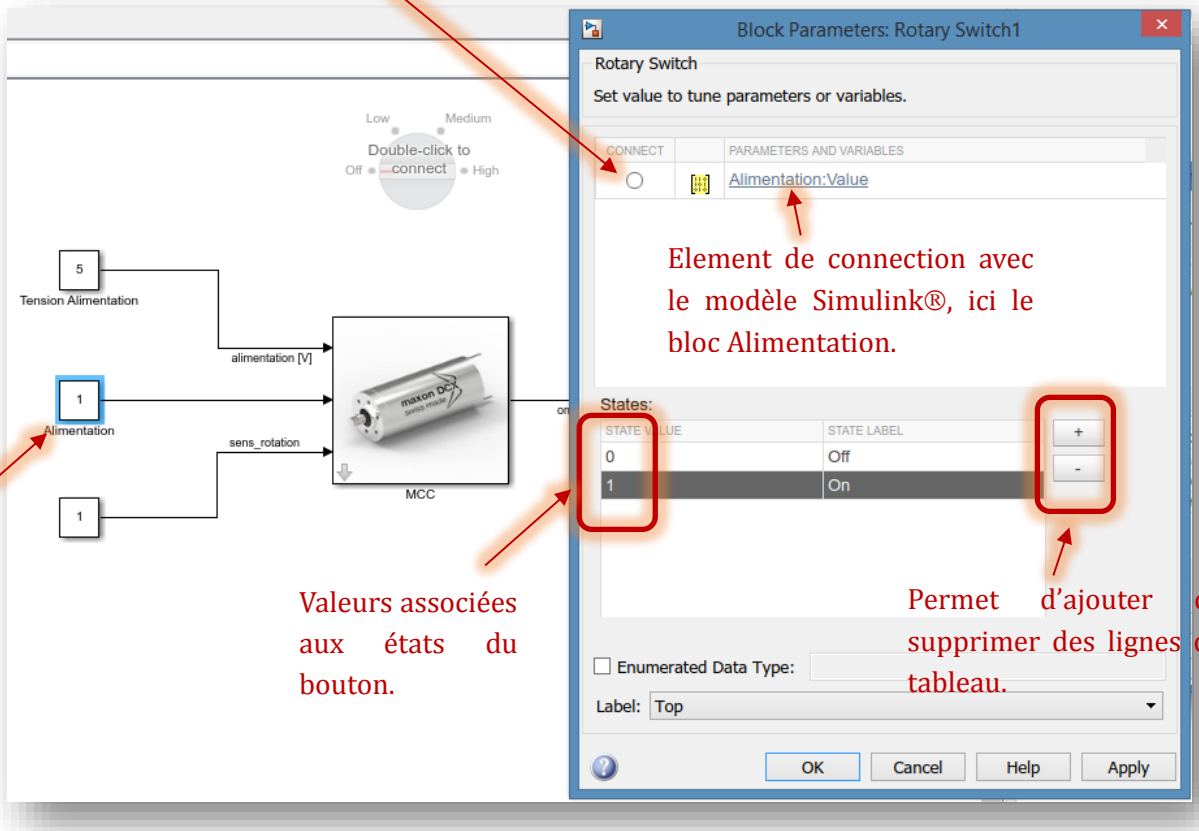
Un glissé-déposé affiche le bloc suivant indiquant qu'il faut double-cliquer pour le connecter :



Un double-clic permet alors d'afficher cette fenêtre qui permet de configurer le bouton que l'on souhaite et de le connecter au modèle Simulink® :



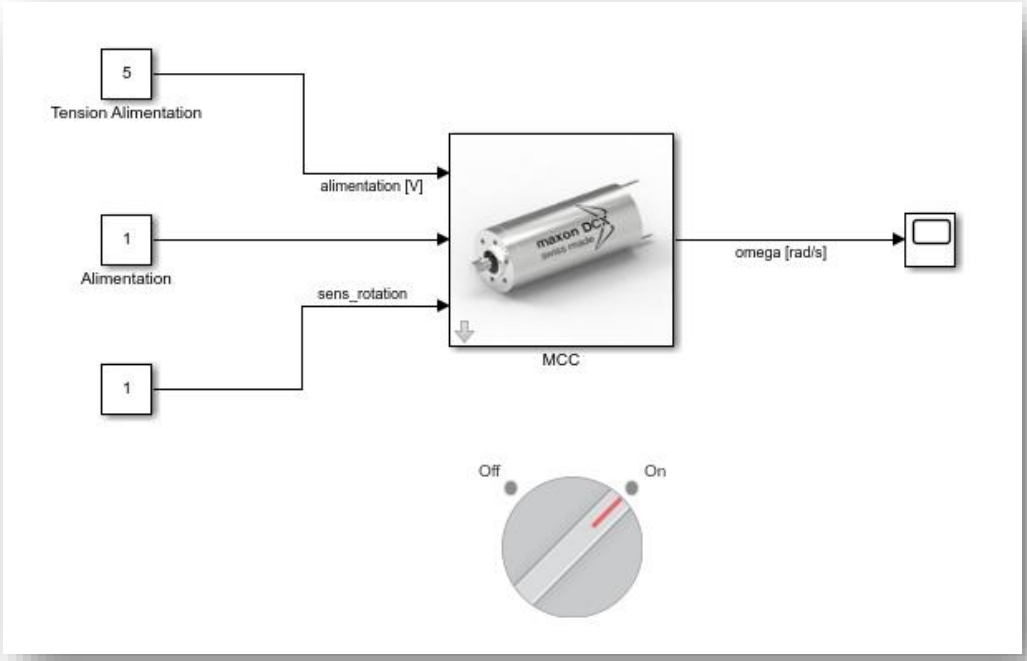

La connection est établie lorsque coché.




Element de connection avec le modèle Simulink®, ici le bloc Alimentation.

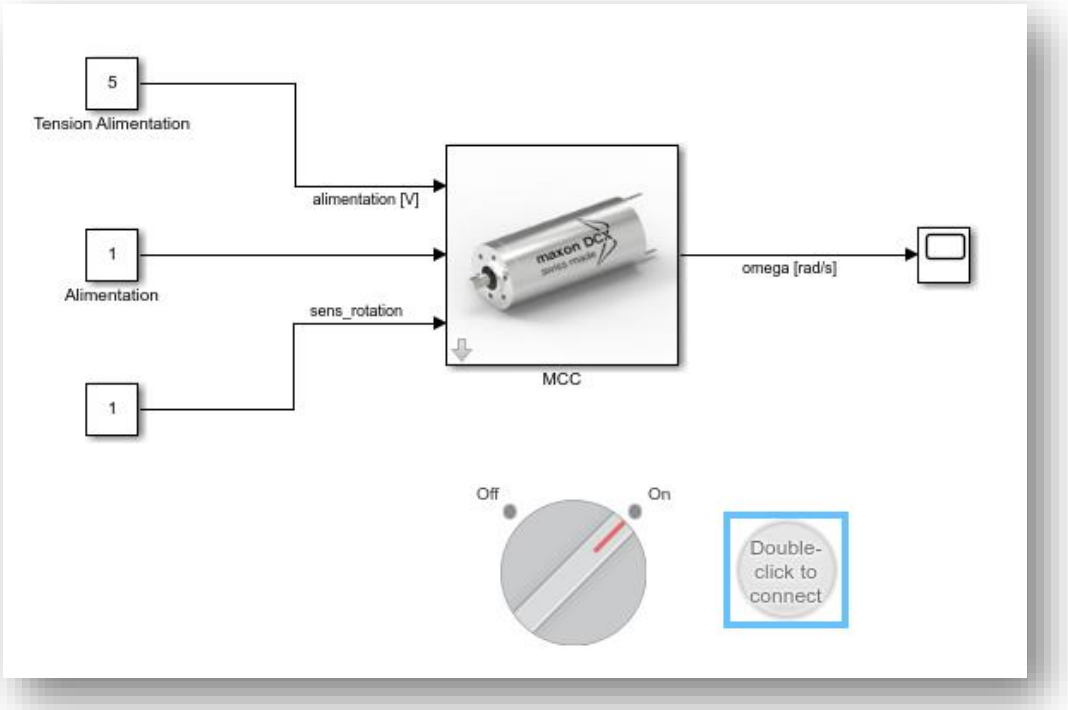
Valeurs associées aux états du bouton.

Permet d'ajouter ou supprimer des lignes du tableau.

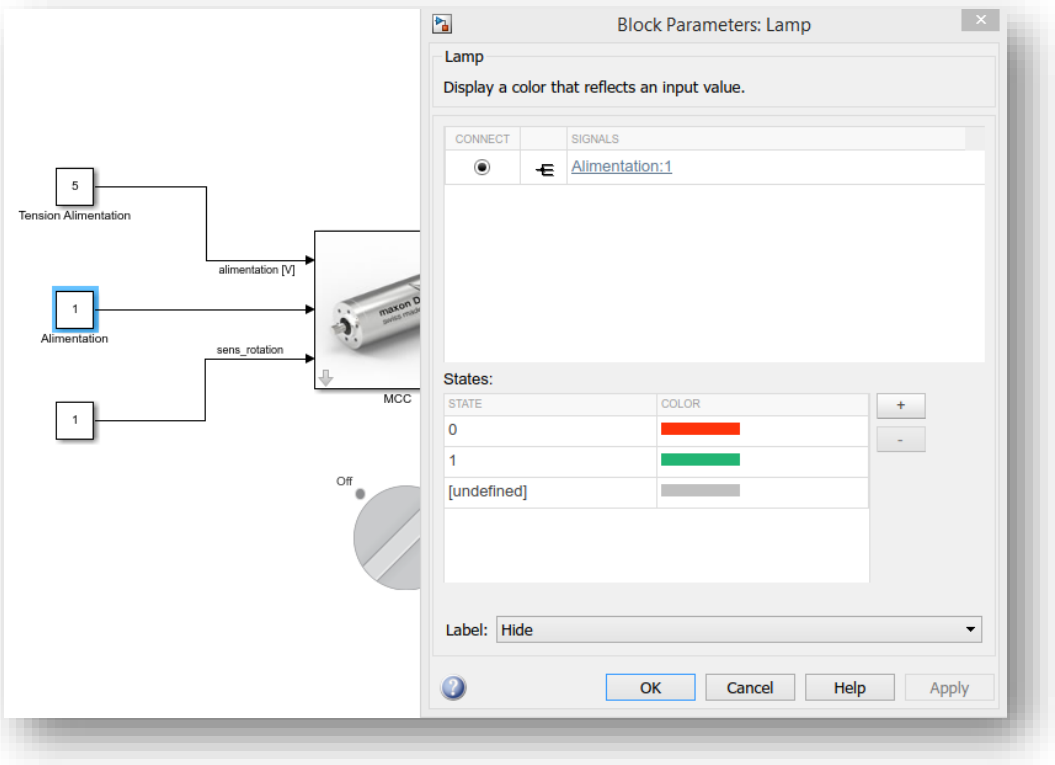



Le bouton apparait alors comme sur la figure précédente.

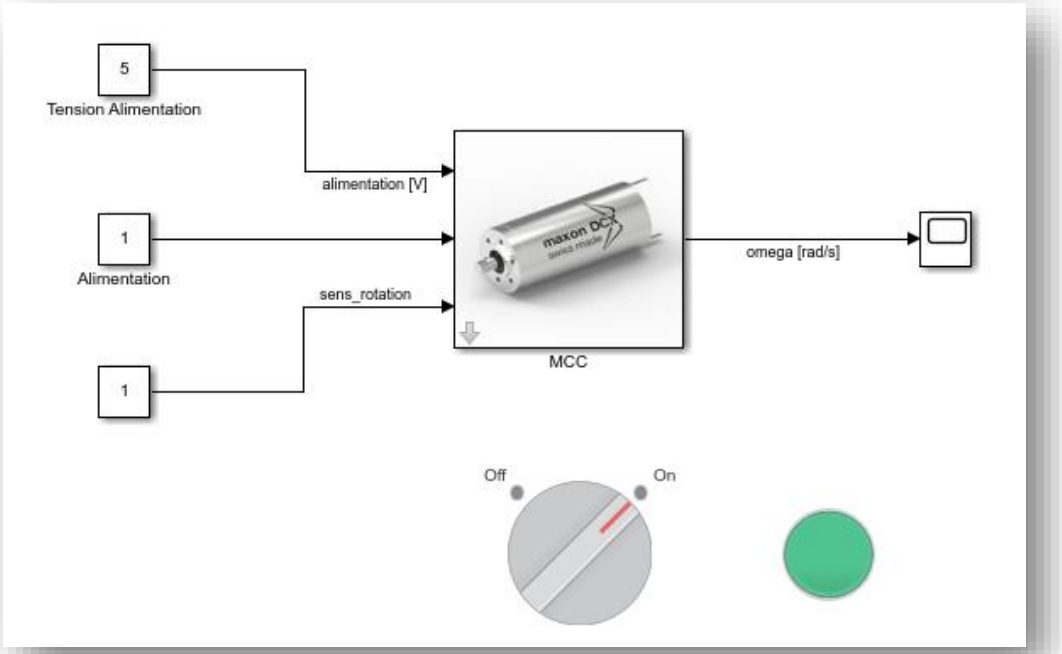
Nous allons maintenant positionner un voyant en ajoutant un élément « **Lamp** » de la bibliothèque :



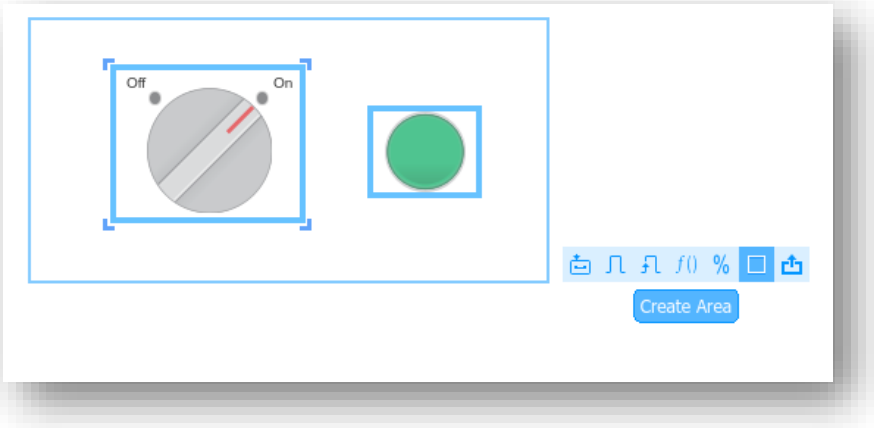
La fenêtre qui suit s'ouvre en double-cliquant sur le voyant, après paramétrage et en connectant le voyant au bloc « Alimentation » du modèle :

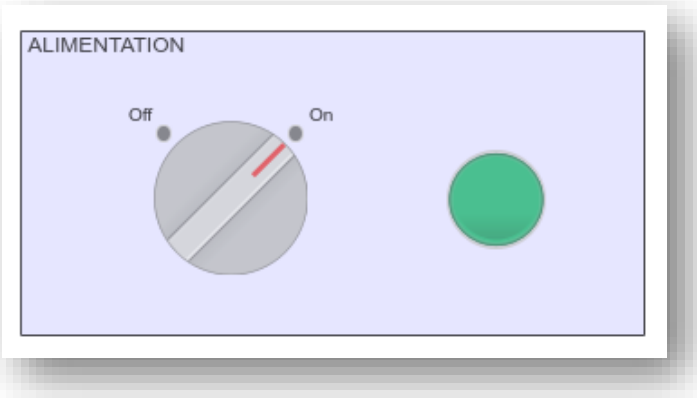



On obtient finalement l'interface suivante :



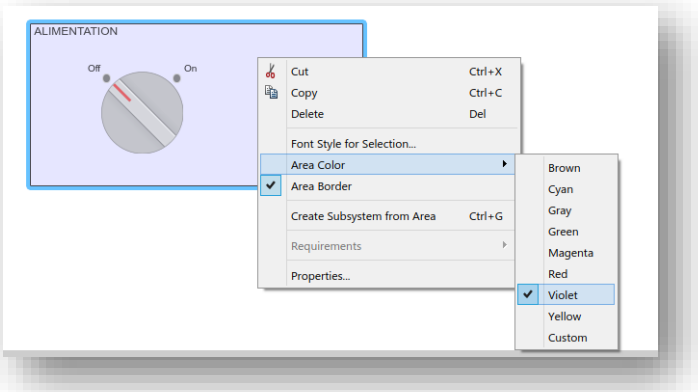
Il est possible de regrouper le bouton et le voyant dans un même bloc « **Area** ». Sélectionner les deux composants et choisir la commande « **Create Area** »




Les deux composants sont alors insérés dans un bloc qu'on pourra nommer « **ALIMENTATION** ».

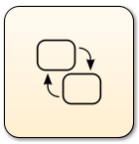
La couleur de fond du bloc peut être modifiée en ouvrant le menu contextuel par un clic droit puis en sélectionnant « Area Color » et la couleur souhaitée.




En lançant une simulation, il est possible de tester le bon fonctionnement de l'interface.

Une interface complète, personnalisable, est ainsi facilement mise en place.

Quel que soit le composant de la bibliothèque du « **Dashboard** », le paramétrage et la connexion se fait de la même manière.



# Chapitre 10

# Stateflow® et Simscape Multibody®

## Model In the Loop (MIL)

# **Chapitre 10** – Stateflow® et Simscape Multibody® (Model in the Loop)…

Simscape Multibody® (anciennement SimMechanics®) est un environnement de simulation multicorps pour les systèmes mécaniques 3D. Il est ainsi possible de modéliser des systèmes multicorps à l'aide de blocs représentant des solides, des liaisons, des contraintes, des actions mécaniques et des capteurs.

Il est aussi possible d'importer dans le modèle des assemblages de CAO complets, comprenant l'ensemble des masses, inerties, liaisons et contraintes, ainsi que la géométrie 3D.

Une animation 3D générée automatiquement sous Matlab® permet de visualiser le dynamique du système.

Simscape Multibody® permet de développer des systèmes de commande afin de les tester puis de les valider dans un environnement Model-In-the-Loop. Une fois valider conformément à un cahier des charges, il est alors possible de déployer la commande validée dans un environnement Hardware-In-the-Loop (HIL) comme nous le verrons dans le chapître suivant.

Dans ce chapitre le support retenu est la plateforme unicycle Pioneer 3DX développé par OMRON Adept Mobile Robots.

On désigne par unicycle un robot actionné par deux roues indépendantes et possédant éventuellement un certain nombre de roues folles pour assurer sa stabilité. C'est le cas de la plateforme Pioneer 3DX, objet de l'étude qui suit.

Ce type de robot est très répandu en raison de sa simplicité de construction et de mise en œuvre.

On note $R_0(\vec{x}_0, \vec{y}_0, \vec{z}_0)$ le repère fixe de référence lié au sol, dont l'axe $\vec{z}_0$ est vertical et $R(\vec{x}, \vec{y}, \vec{z})$ un repère mobile lié à la plateforme. On pose P le point situé au milieu de l'axe passant par les centres des roues motrices.

On appelle situation ou encore posture du robot le vecteur :

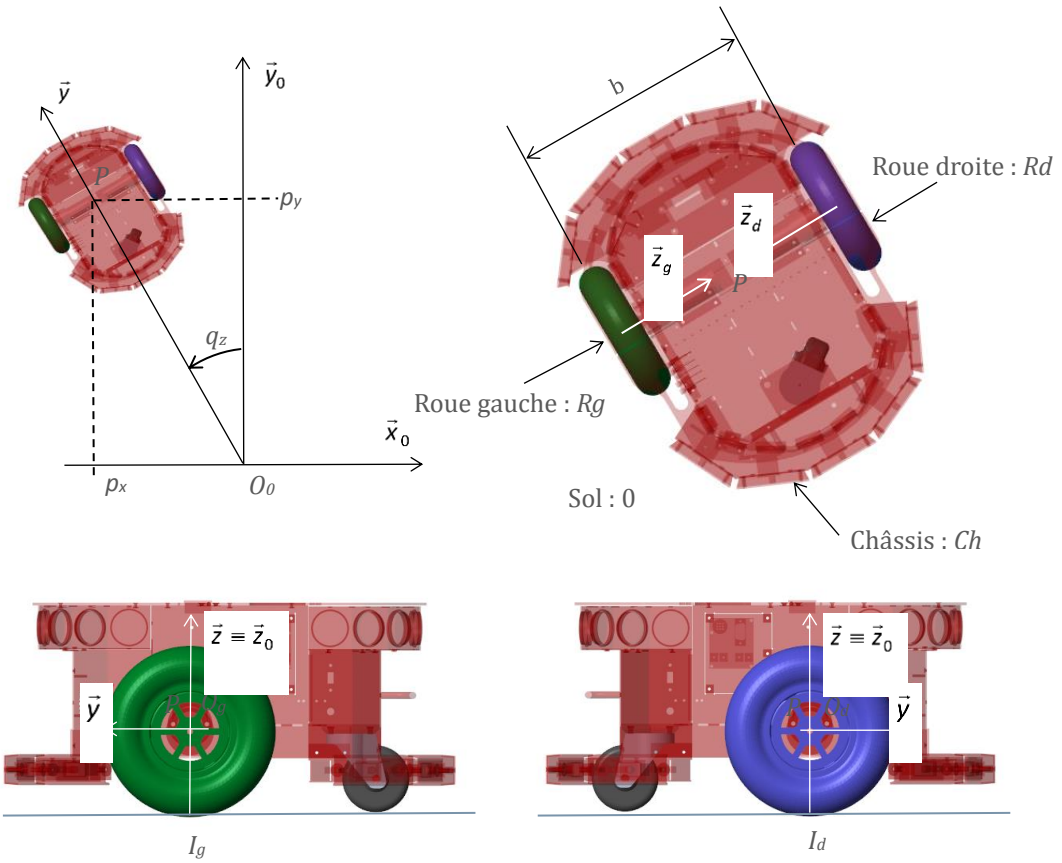
$$\xi = \begin{pmatrix} p_x \\ p_y \\ q_z \end{pmatrix}_{(\vec{x}_0, \vec{y}_0, \vec{z}_0)}$$

où $p_x$ et $p_y$ sont respectivement l'abscisse et l'ordonnée du point P dans $R_0(\vec{x}_0, \vec{y}_0, \vec{z}_0)$ et $q_z$ l'angle orienté $(\vec{y}, \vec{y}_0)$.

***Remarque*** : les notations proposées ici sont en accord avec celles que vous retrouverez en manipulant la suite Matlab®/Simulink®/SimMechanics®. D'autre part, le paramétrage donné ci-après est celui adopté pour la simulation.

On pose :

- $I_g$ et $I_d$ sont les points de contact des roues gauche et droite avec le sol.
- $\overrightarrow{I_gO_g} = \overrightarrow{I_dO_d} = r\vec{z}_0$, $r$ est le rayon des roues gauche et droite avec $r$ = 0.095 m
- La distance $I_gI_d$ (voie) vaut $b$, avec $b$ = 0.33 m.
- $\overrightarrow{O_0P} = p_x\vec{x}_0 + p_y\vec{y}_0 + r\vec{z}_0$
- $\vec{V}(P, Ch/0) = \dot{p}_x\vec{x}_0 + \dot{p}_y\vec{y}_0 = V\vec{y}$ et $\vec{\Omega}(Ch/0) = \dot{q}_z\vec{z} = \omega\vec{z}$
- $\vec{\Omega}(R_d/Ch) = \dot{\phi}_d\vec{z}_d$ et $\vec{\Omega}(R_g/Ch) = \dot{\phi}_g\vec{z}_g$



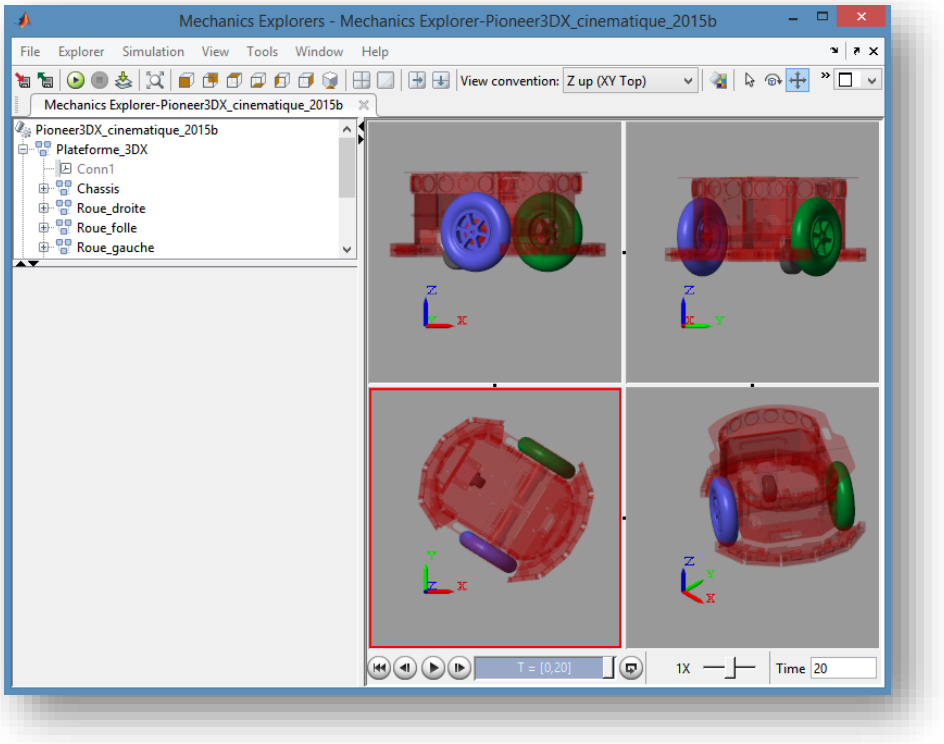

Pour commander un robot unicycle il est nécessaire d'agir sur les motorisations des roues droite et gauche. La problématique envisagée ici concerne la commande du robot.

Si nous nous contentons de l'approche cinématique, nous pouvons considérer que la commande est donnée par les vitesses de rotation des roues. Mais en général pour une appréciation du comportement du robot plus naturelle, on préfère exprimer cette commande par la vitesse longitudinale du robot V, et sa vitesse de rotation $\omega$.

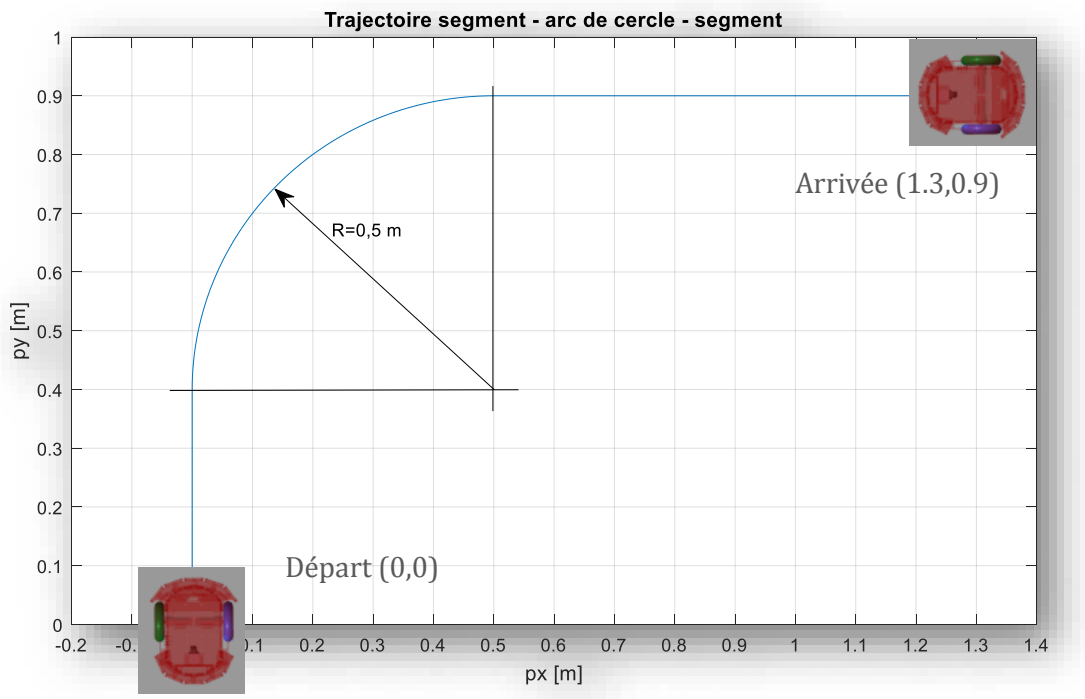
Notre objectif dans cette application est de mettre en place la commande qui permet à la plateforme un déplacement automatisé, c'est-à-dire la commande qui permet à la plateforme de suivre une trajectoire imposée.

La validation du comportement cinématique de la plateforme se fera au moyen d'une simulation avec la suite logicielle Matlab®/Simulink®/SimMechanics®.

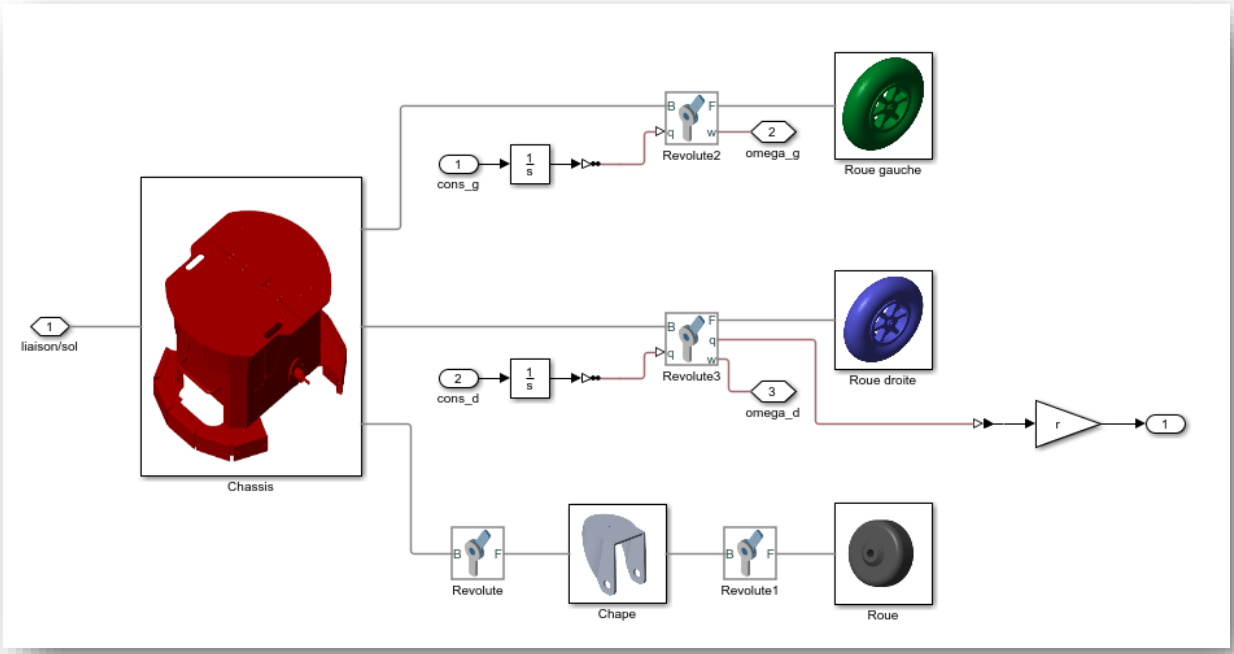
Une maquette numérique de la plateforme a été importée dans SimMechanics ce qui permet d'implanter la commande et de visualiser les trajectoires du point P lié à la plateforme assez simplement en utilisant les outils intégrés.




Le modèle à saisir doit permettre l'animation de la maquette numérique de la plateforme unicycle de manière la plus réaliste possible. Nous nous intéressons ici à son comportement cinématique et à la gestion de la détection d'un obstacle sur la trajectoire imposée. Celle-ci est composée de trois tronçons segment – arc de cercle - segment. Ci-dessous est représentée la trajectoire du point P lié à la plateforme :




Examinons la structure de la maquette :

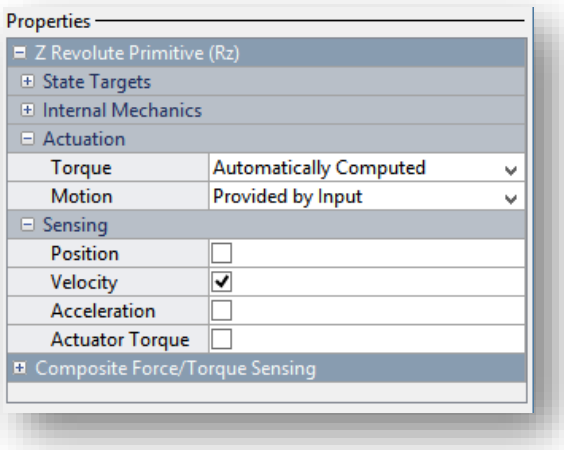


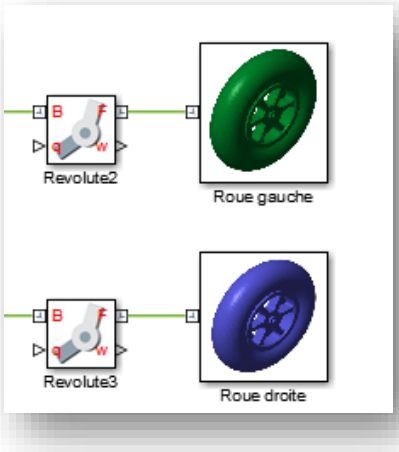

Les roues gauche et droite sont en liaison pivot avec le châssis. La roue folle est en liaison pivot avec la chape qui elle est en liaison pivot avec le châssis.

Pour animer la maquette il suffit d'appliquer les consignes de vitesse aux roues droite et gauche.

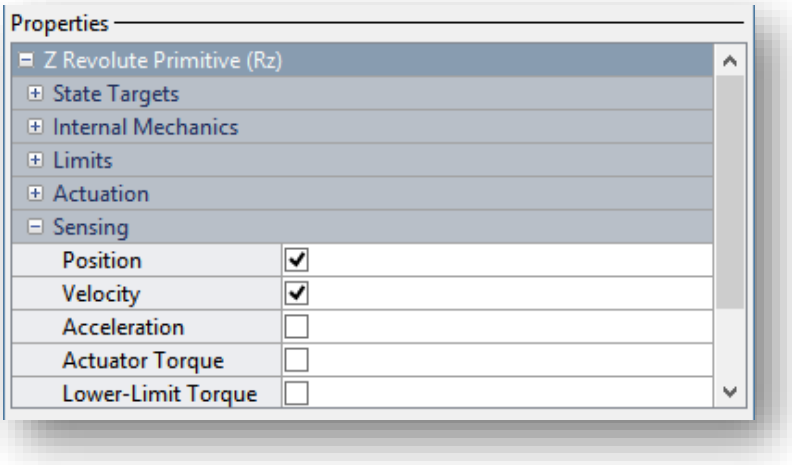
Nous allons donc « piloter » les liaisons pivot entre les roues et le châssis de la plateforme. Le paramètre d'entrée piloté est obligatoirement une position angulaire notée *q*, d'où la présence des deux blocs intégrateurs qui intègrent les vitesses de consigne.

Pour chacune des liaisons pivot (Revolute), double-cliquer sur la liaison et paramétrer la liaison comme suit :



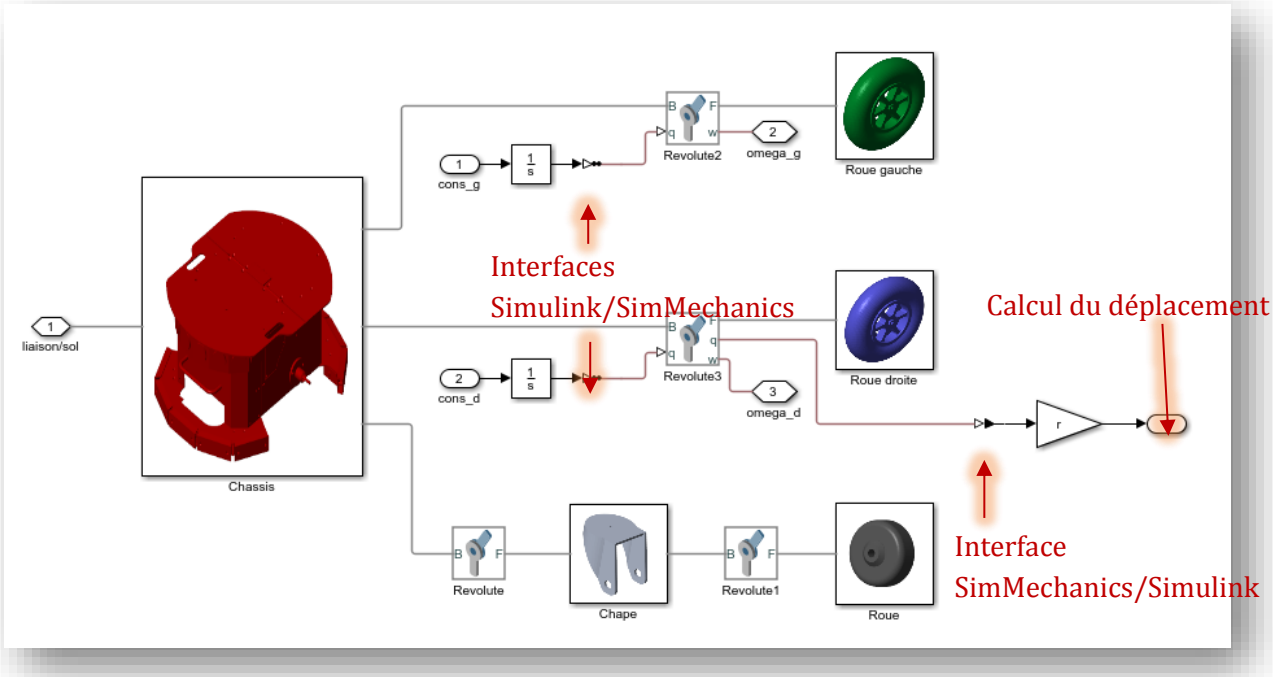




D'autre part , nous allons relever la position angulaire de la roue droite par rapport au châssis à chaque instant afin d'évaluer le déplacement de la plateforme, comme on pourrait le faire avec un codeur sur une plateforme réelle.



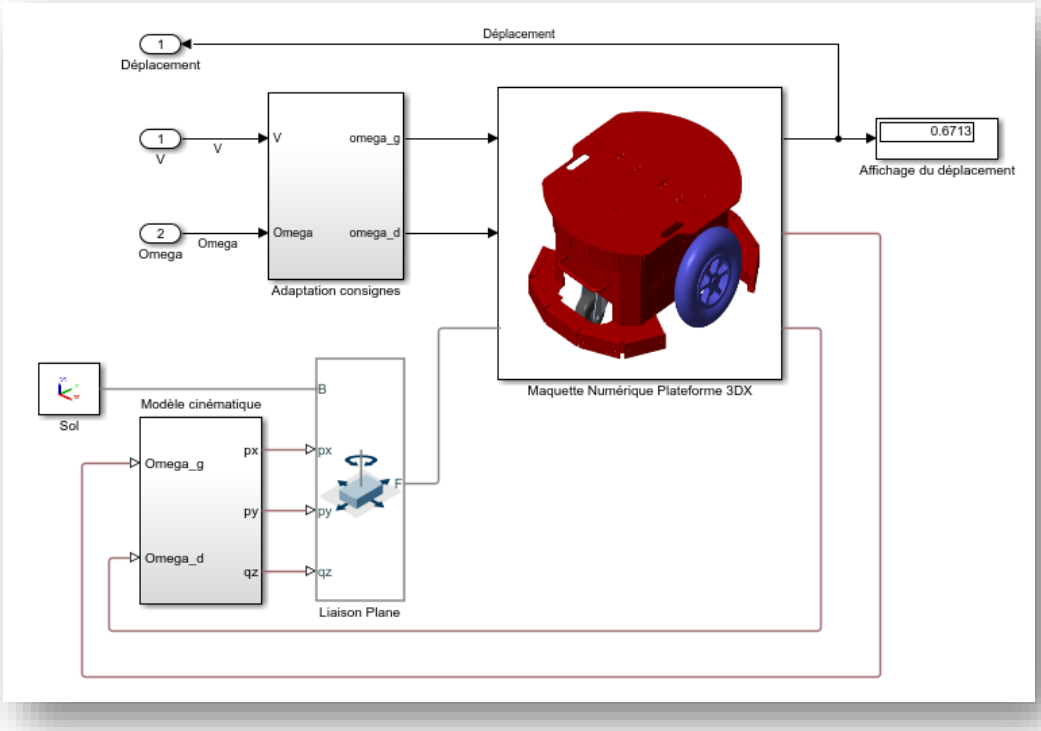

Cocher en plus de « **Velocity** », « **Position** » dans le menu « **Sensing** » de « **Z Revolute Primitive (Rz)** ».

Le déplacement de la plateforme *s* sera donné par le produit du déplacement angulaire de la roue droite, *q*, et du rayon *r* de la roue.

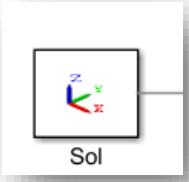



Enfin pour passer du « domaine » Simulink au « domaine « SimMechanics », et inversement, il est nécessaire du positionner des interfaces : la flèche blanche est associée à SimMechanics®, la noire à Simulink®.

Affichons le niveau supérieur du modèle :



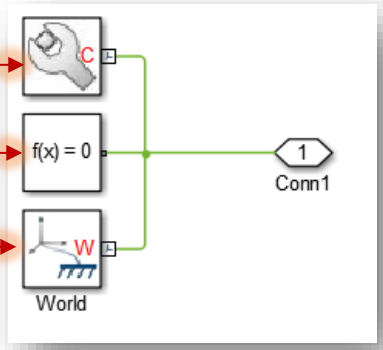
Dans bloc « Sol » sont regroupés les éléments de configuration du modèle ainsi que le repère de référence lié au sol :




Configuration du modèle où par exemple l'accélération de la pesanteur est définie

Configuration du solveur qui permettra les calculs

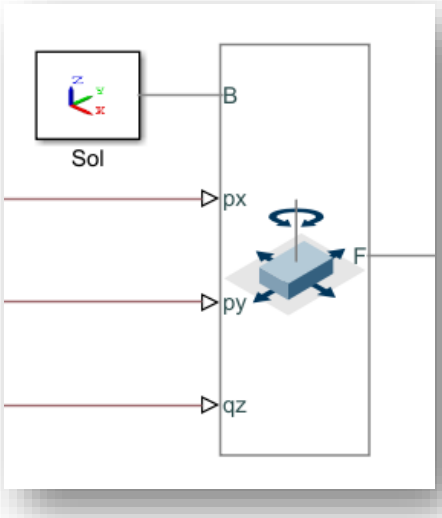
Repère de référence




La liaison entre la plateforme et le sol est une liaison plane (« Plane Joint ») qui autorise trois degrés de liberté.

Dans le cas qui nous intéresse nous allons « piloter » cette liaison pour mouvoir la maquette de la plateforme.

Nous devrons donc agir sur les trois paramètres de la liaison, c'est-à-dire les trois composantes du vecteur qui définit la posture de la plateforme :



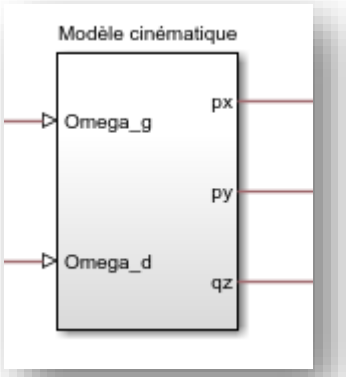

$$\xi = \begin{pmatrix} p_x \\ p_y \\ q_z \end{pmatrix}_{(\vec{x}_0,\vec{y}_0,\vec{z}_0)}$$

Le bloc dispose des trois entrées situées située sa gauche, $p_x$ $p_y$ et $q_z$ .

le bloc « Sol » est relié au port B (Base) de la liaison et le bloc Plateforme 3DX au port F (Follower) de la liaison.

La plateforme est donc contrainte à se déplacer sur un plan. Cette contrainte est liée aux conditions de roulement sans glissement entre le sol et les roues.

Le bloc « Modèle cinématique » met en relation les vitesses angulaires de chaque roue et la posture de la plateforme.

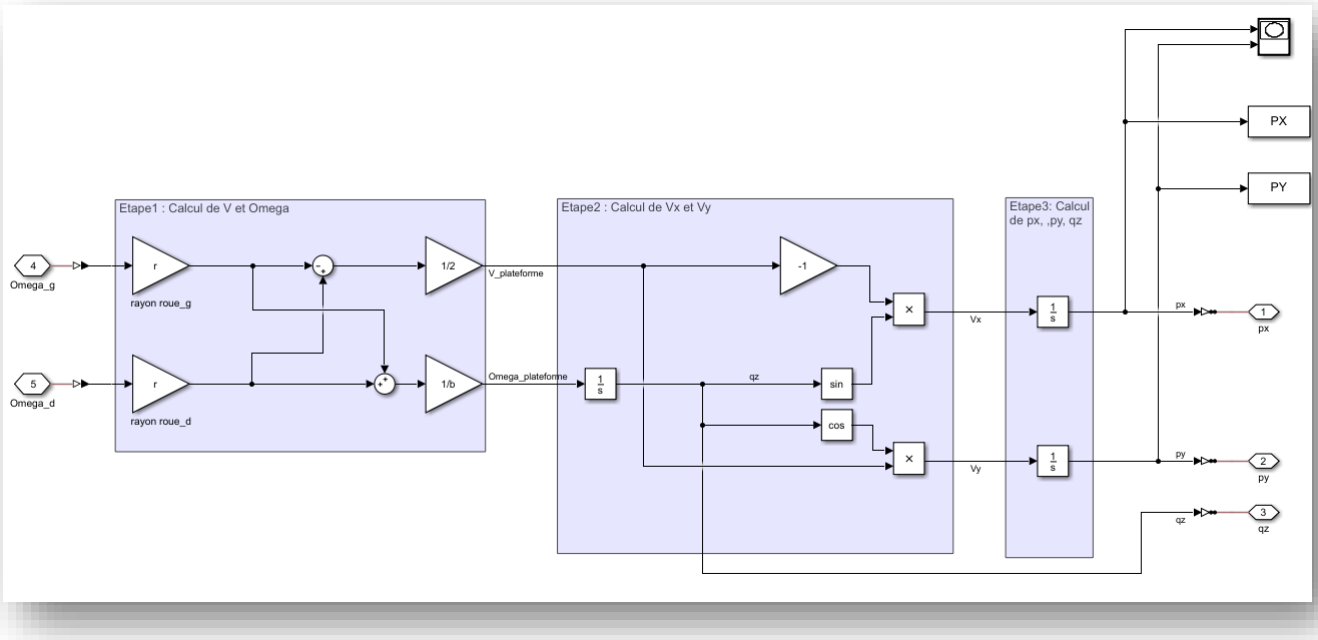



Cette relation est établie en trois étapes :

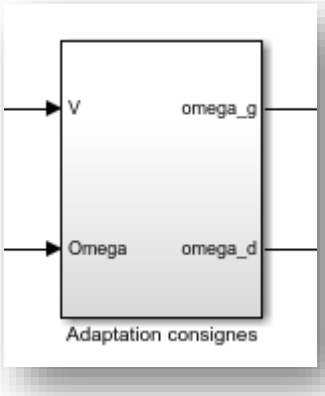
- Etape 1 : calcul de V et Omega en fonction des vitesses anulaires des roues mesurées sur la maquette et telles que : $\vec{V}(P, Ch/0) = V\vec{y}$ et $\vec{\Omega}(Ch/0) = \dot{q}_z\vec{z} = \omega\vec{z}$

- Etape 2 : calcul de Vx et Vy telles que : $V\vec{y} = \dot{p}_x\vec{x}_0 + \dot{p}_y\vec{y}_0 = V_x\vec{x}_0 + V_y\vec{y}_0$

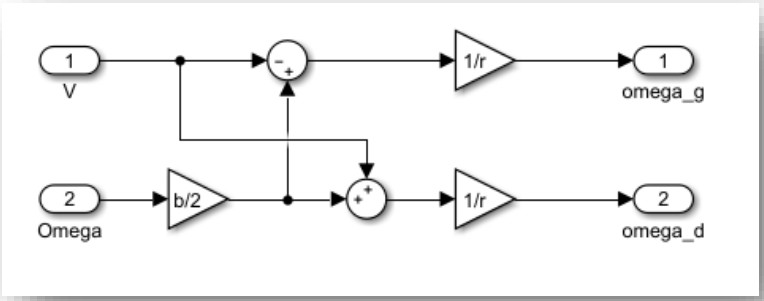
- Etape 3 : calcul de px, py et qz correspondant à la posture de la plateforme



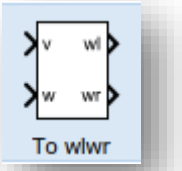

Le « scopeXY » permet l'affichage de la trajectoire du point P de la plateforme.

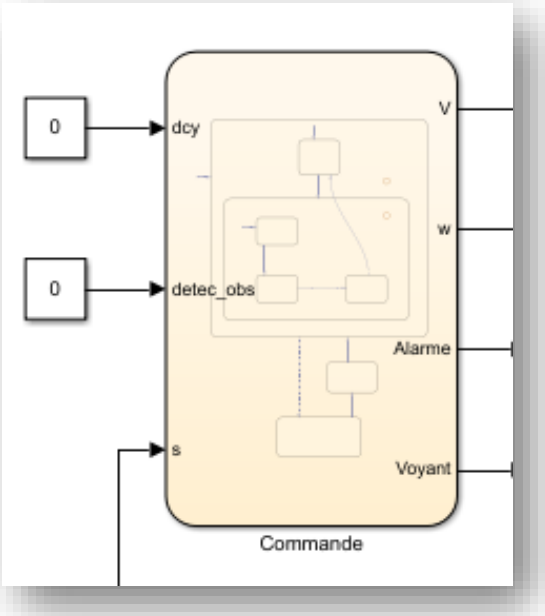



Le bloc « Adaptation consignes » permet d'élaborer les consignes de vitesse à appliquer aux roues droite et gauche de la plateforme en fonction de la commande (V, Omega).




Pour information, ce bloc est disponible dans la bibliothèque « **Mobile Robotics Traning Library** » de Simulink®.




Dans le niveau encore supérieur, le bloc « Commande » génère la commande qui permettra de suivre la trajectoire imposée. Cette commande est générée par un « Chart » dont les entrées sont dcy, detec_obs et s.

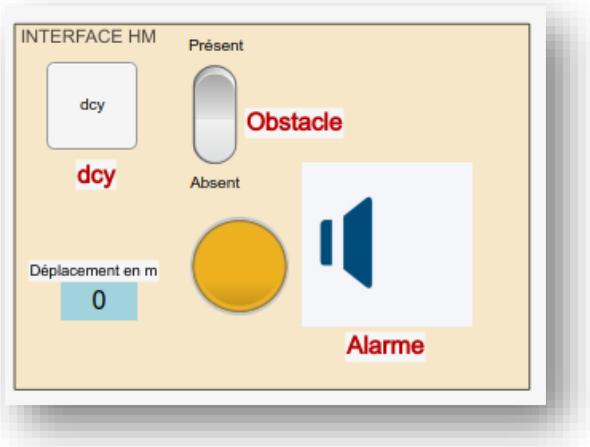
Avec

- dcy : variable associée au bouton départ cycle
- detec_obs : variable associée au bouton d'arrêt d'urgence
- s : variable associée au déplacement de la plateforme.

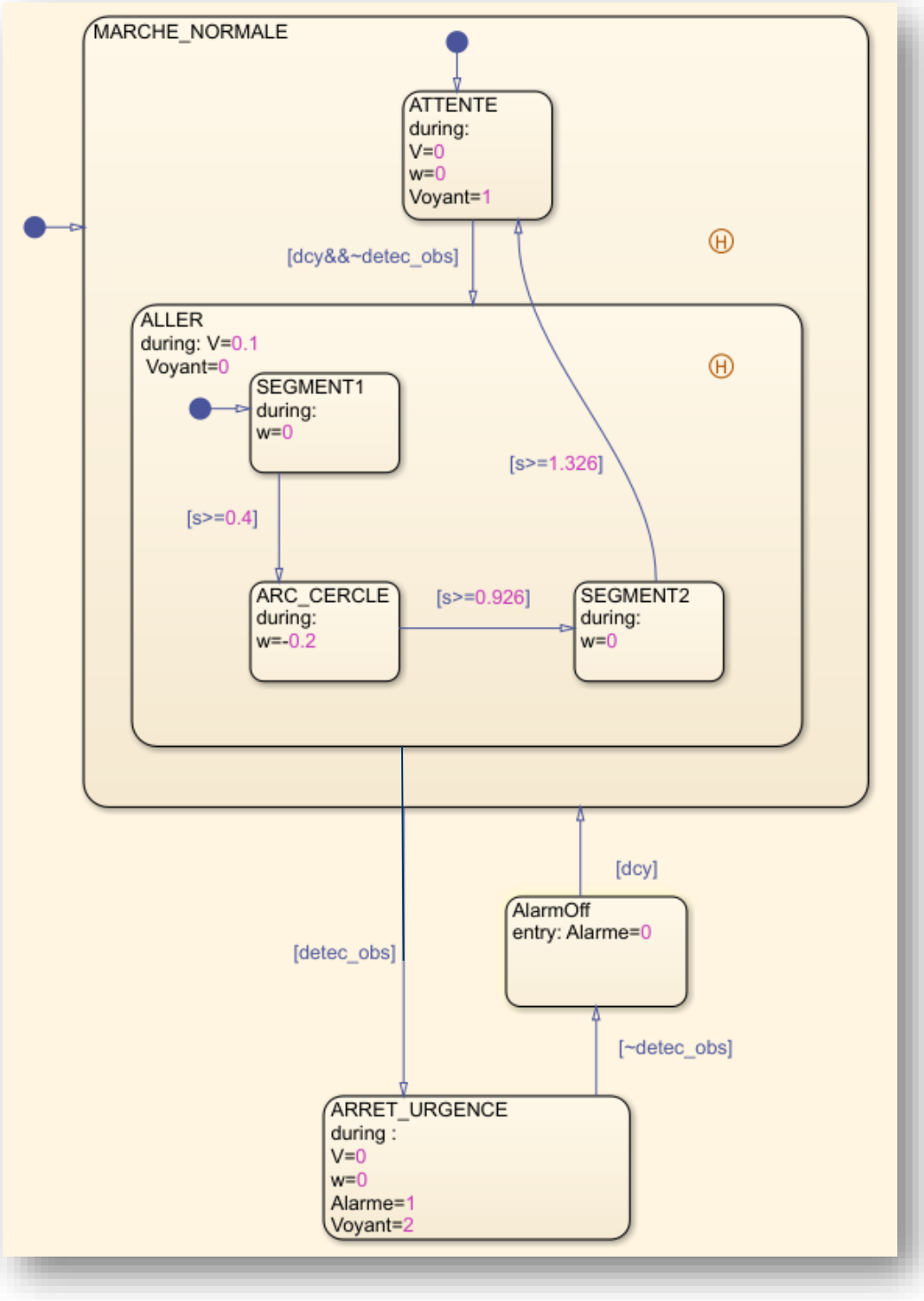
Les éléments du cahier des charges qu'il convient de respecter sont les suivants :

- Le point P de la plateforme doit suivre la trajectoire imposée constituée de deux segments orthogonaux reliés par un quart de cercle de rayon 0.5m.
- La plateforme se déplace à vitesse constante (0.1m/s)sur tout le trajet imposé.
- Pour lancer le déplacement de la plateforme, un opérateur doit appuyer sur le bouton dcy. Un voyant préalablement orange, passe au vert prévenant ainsi l'opérateur du bon fonctionnement de l'ensemble.
- Si la plateforme détecte un obstacle sur son trajet, elle doit s'arrêter. Une alarme retentit et un voyant rouge s'allume dans le but de prévenir l'opérateur qui doit enlever l'obstacle.
- Une fois l'obstacle retiré l'alarme s'éteint et l'opérateur doit appuyer à nouveau sur le bouton dcy pour que la plateforme reparte et continue son chemin

Pour la simulation, le bouton départ cycle sera monostable. La présence ou non d'un obstacle sur le trajet de la plateforme sera modélisée par l'action sur un bouton bistable.

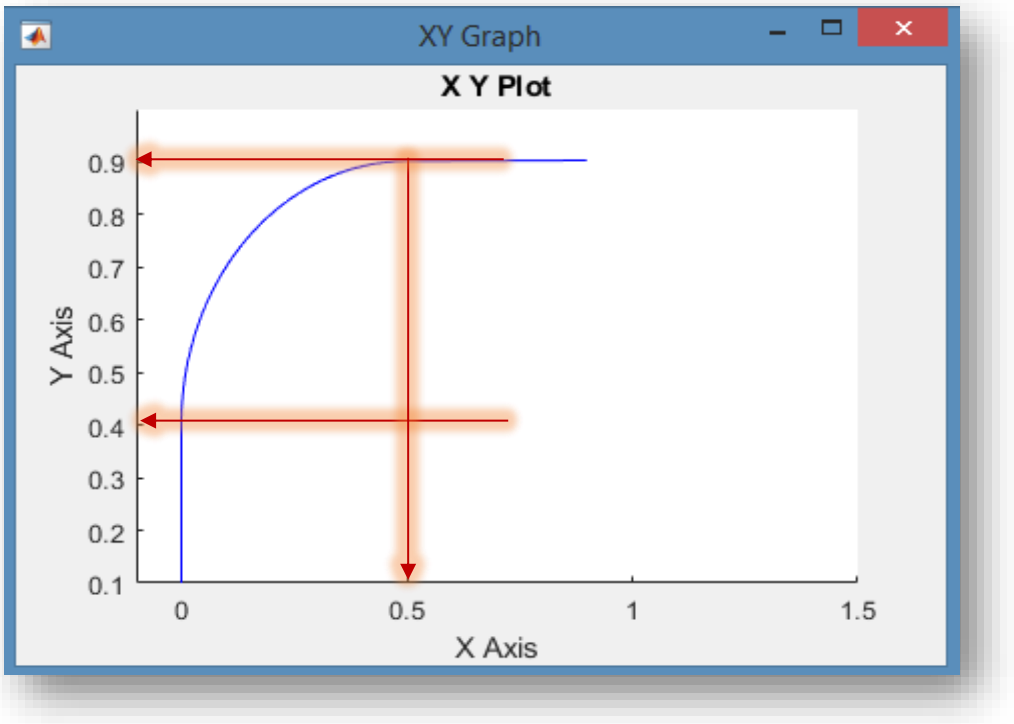


Une solution envisageable pour générer la commande pourrait être la suivante :

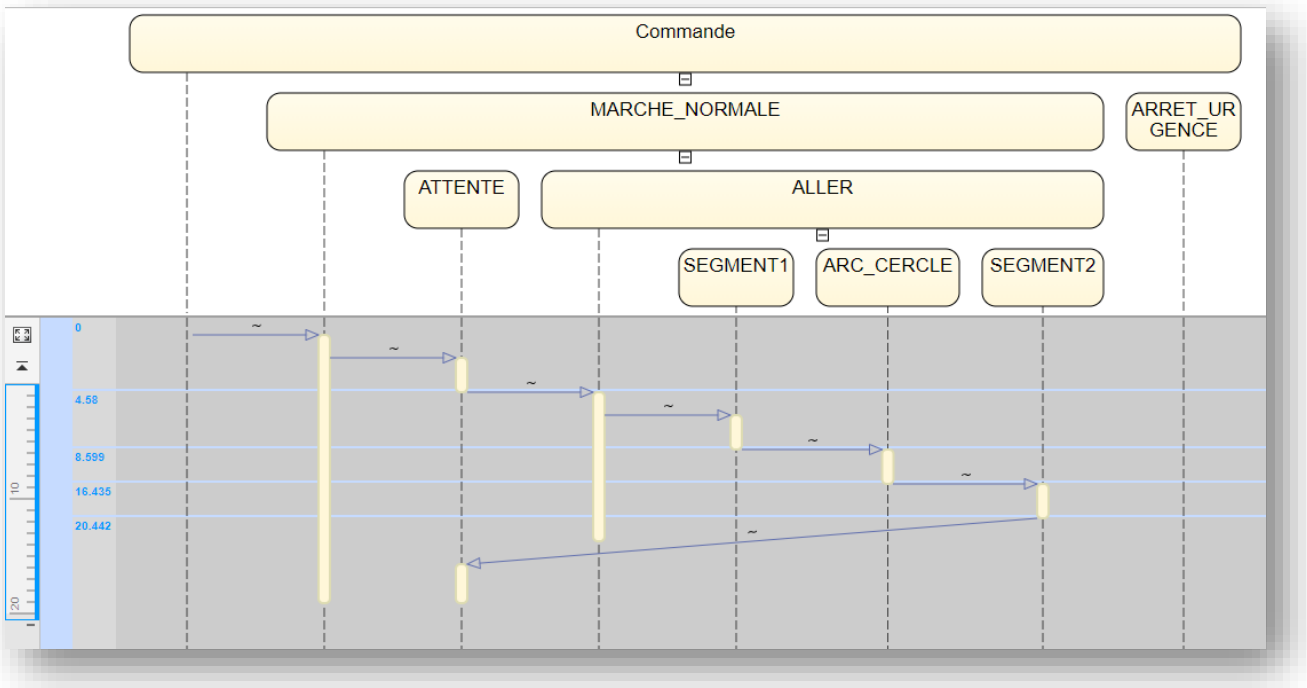



Attention ici, le calcul de s est celui du déplacement du point de contact avec le sol de la roue droite et non pas celui du déplacement du point P.

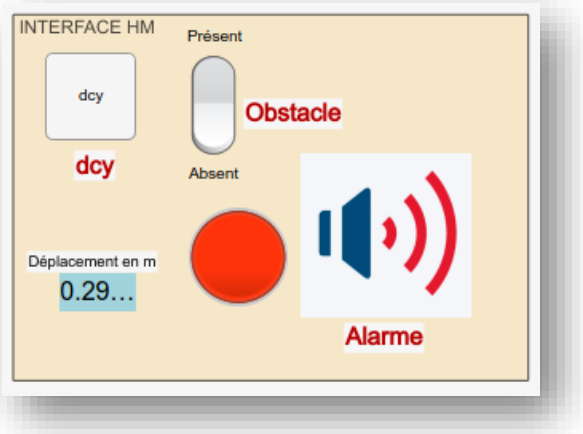
L'affichage du « scope XY » donne par contre la trajectoire du point P lié à la plateforme, celle-ci est bien conforme au cahier des charges :



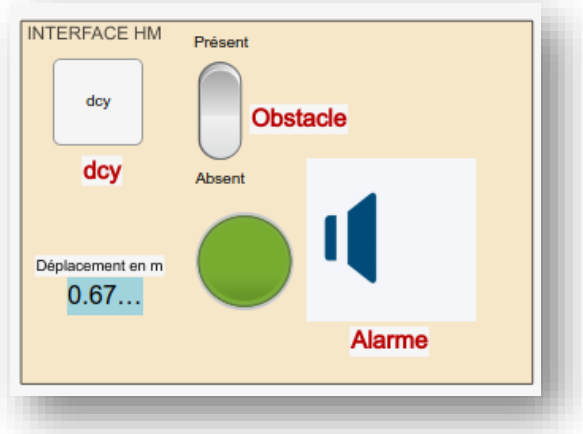

Le « **Sequence Viewer** » permet d'afficher le diagramme de séquence pour un fonctionnement sans détection d'obstacle :



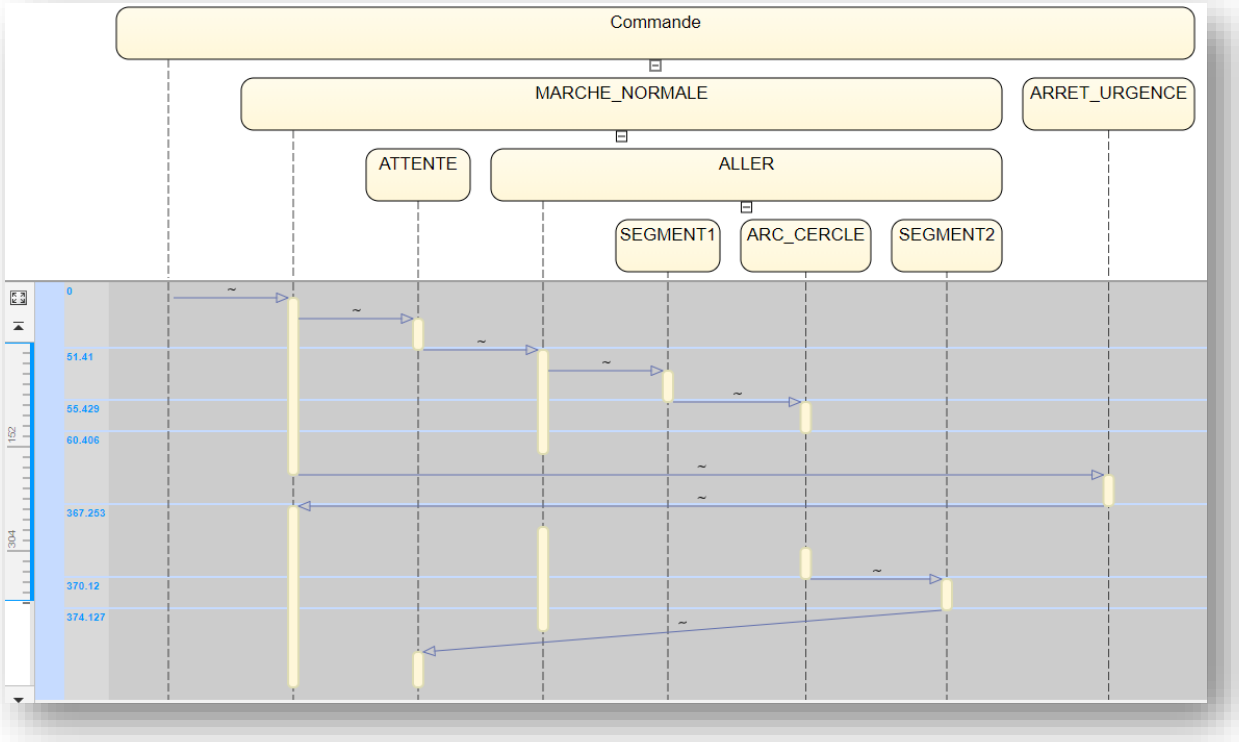

Observons le comportement de la plateforme lorqu'un obstacle est détecté dans le virage :



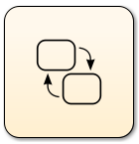

Dans ce cas la plateforme s'arrête, un voyant rouge s'allume et l'alarme retentit.




L'obstacle est retiré, l'opérateur appuie à nouveau sur le bouton de départ cycle, la plateforme redémarre comme le montre le diagramme de séquence.



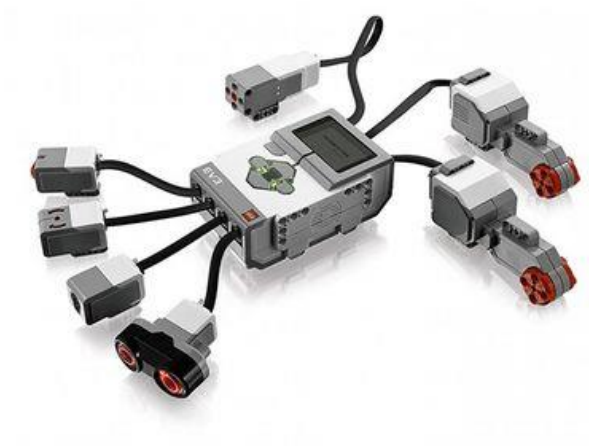

Comme nous venons de l'observer, la simulation permet la validation du cahier des charges.



# Chapitre 11

# Le prototypage

## Hardware In the Loop (HIL)



# **Chapitre 11** – Le prototypage (Hardware In the Loop)…

Dans ce chapitre consacré au prototypage dans un environnement Hardware-In-the-Loop, nous proposons une application qui prend comme support cible la brique LEGO® Mindstorms EV3.

Cette cible polyvalente a été choisie ici pour sa facilité de mise en œuvre et son coût réduit. Aussi, elle permet d'appréhender des problématiques complexes de l'automatique très rapidement avec un budget très raisonnable.

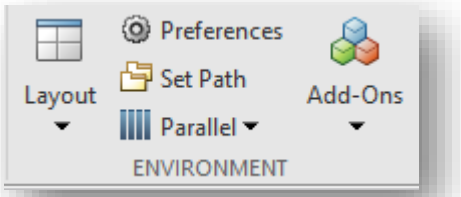
## 11.1 – LE KIT LEGO® MINDSTORMS EV3 EDUCATION


Le kit propose une brique EV3, plusieurs capteurs, deux types de servomoteurs et un ensemble de pièces permettant de construire, par exemple, une plateforme unicycle. Nous nous servirons de cette plateforme dans ce chapitre. Elle est composée de la brique EV3, de deux servomoteurs pilotés indépendamment l'un de l'autre, d'un capteur de luminosité pour le suivi d'une ligne noire, et d'un capteur à ultra-sons pour la détection d'obstacle.

Les caractéristiques des différents éléments du kit sont disponibles sur le site internet de LEGO® : https://www.lego.com/fr-fr/themes/mindstorms

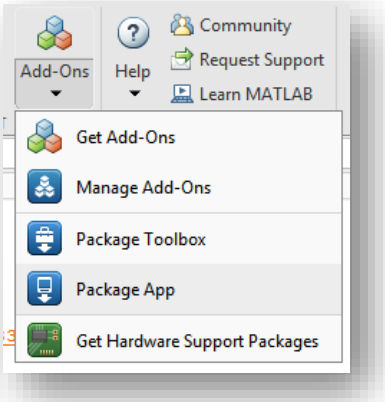
## 11.2 – LEGO MINDSTORMS EV3 ET SIMULINK®

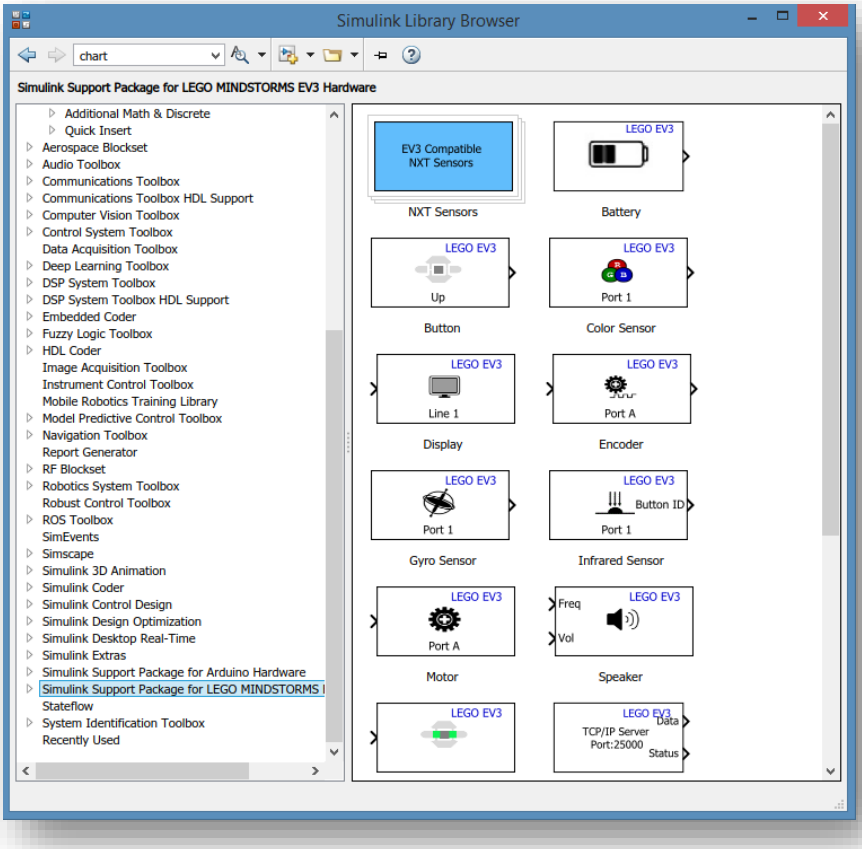
Pour pouvoir communiquer avec la brique LEGO® Mindstorms EV3 il faut télécharger et installer le Add-On « **Simulink Support Package for LEGO MINDSTORMS EV3 hardware** » :


Pour ce faire cliquer sur l'icône « **Add-Ons** » du menu « **ENVIRONNEMENT** » de MATLAB.

Cliquer ensuite sur « **Get Hardware Support Package** » et installer le module.



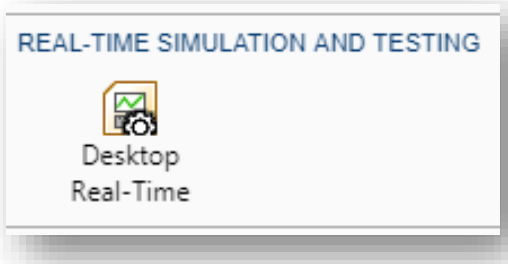

Une fois le module installé on accède aux différents composants dans la bibliothèque Simulink.

Ces blocs permettront la génération d'un programme qui sera ensuite charger dans la cible, la brique EV3.

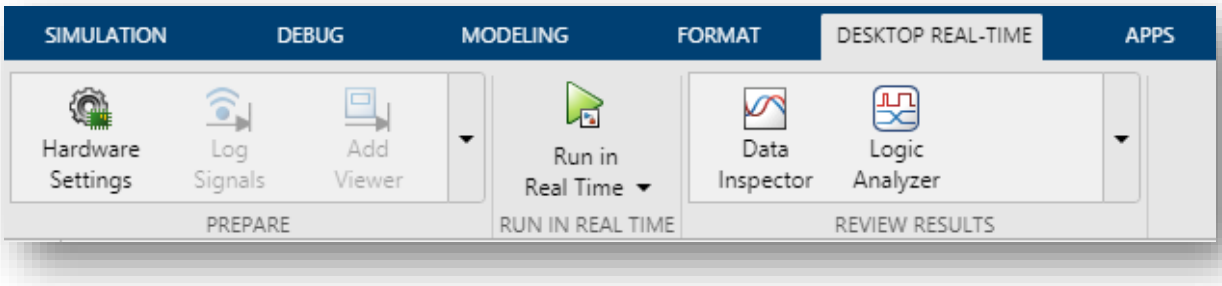
L'environnement Hardware-In-the-Loop consiste à piloter la cible matérielle au moyen de la commande élaborée et validée dans le chapitre précédent.

Le comportement dynamique de la cible sera simulé et testé en temps réel.

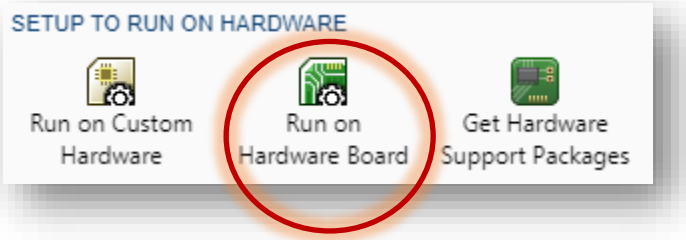
Pour ce faire, il faut cliquer sur l'onglet « **APPS** » du bandeau supérieur et sélectionner l'application « **DESKTOP REAL-TIME** » qui se trouve dans la rubrique « **REAL-TIME SIMULATION AND TESTING** »




L' onglet « **DESKTOP REAL-TIME** » apparait dans le bandeau supérieur.

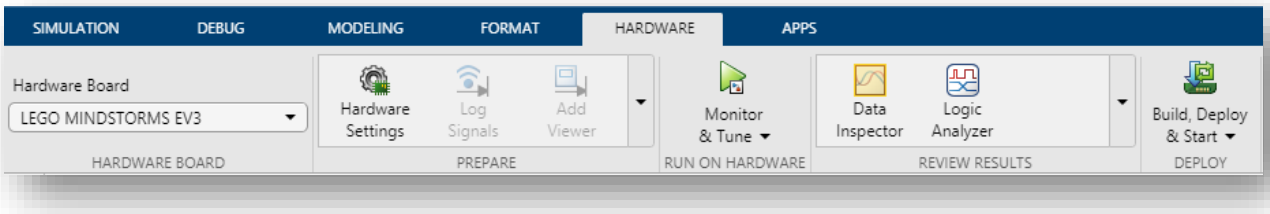



Pour que Simulink reconnaisse la brique EV3, dans « APPS » cliquer sur le bouton « Run on Hardware Board » du menu « SETUP TO RUN ON HARDWARE »

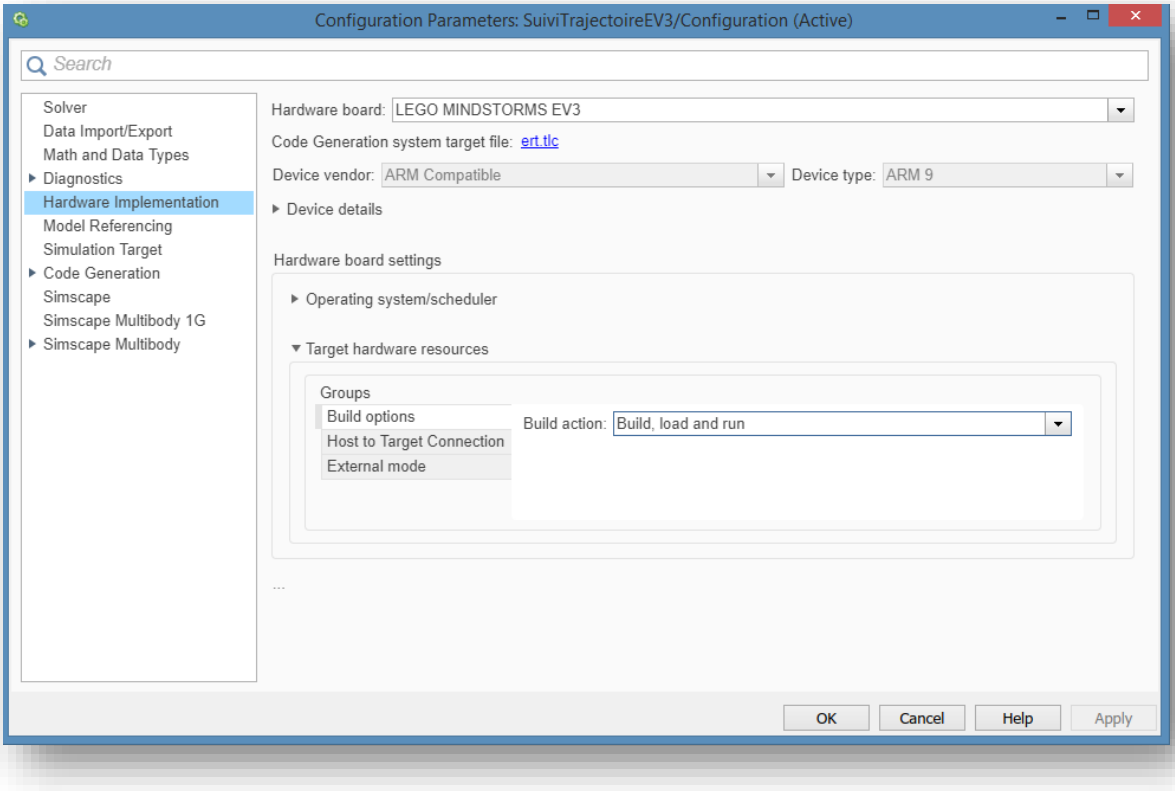



Sélectionner la cible LEGO MINDSTORM EV3.

Le bandeau supérieur se présente alors de la manière suivante :

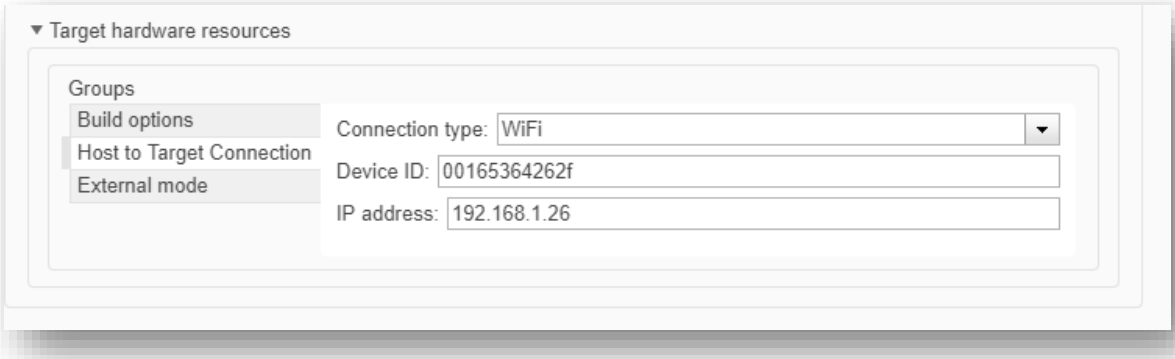



Il convient ensuite de configurer l'ensemble en cliquant sur le bouton « **Hardware Settings** » du menu « **PREPARE** » de l'onglet « **HARDWARE** » :



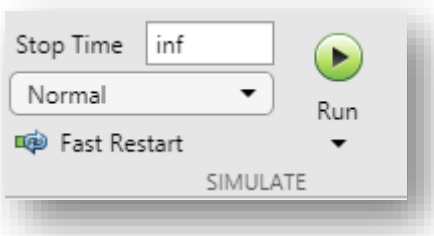

Il faut configurer la connexion. Ici elle se fait en WiFi. Néanmoins elle peut se faire via un câble USB, en Bluetooth, ou encore via un réseau ethernet.

Dans notre cas, il faut saisir l'ID de la brique EV3 ainsi que son adresse IP :




Ces informations se lisent sur l'écran de la brique, dans le menu « Brick Info » lorsque celle-ci est connectée au serveur WiFi.

Dans l'onglet « SIMULATION » ou « MODELING » régler « Stop Time » sur infini :




L'environnement logiciel est alors prêt pour le prototypage.

# 11.3 – LA PLATEFORME FIFIBOT

## 11.3.1 Rappel du cahier des charges et choix technologiques

Les éléments du cahier des charges qu'il convient de respecter sont les suivants :

- Le point P de la plateforme doit suivre la trajectoire imposée constituée de deux segments orthogonaux reliés par un quart de cercle de rayon 0.5m.
- La plateforme se déplace à vitesse constante (0.1m/s) sur tout le trajet imposé.
- Pour lancer le déplacement de la plateforme, un opérateur doit appuyer sur le bouton dcy. Un voyant préalablement orange, passe au vert prévenant ainsi l'opérateur du bon fonctionnement de l'ensemble.
- Si la plateforme détecte un obstacle sur son trajet, elle doit s'arrêter. Une alarme retentit et un voyant rouge s'allume dans le but de prévenir l'opérateur qui doit enlever l'obstacle.
- Une fois l'obstacle retiré l'alarme s'éteint et l'opérateur doit appuyer à nouveau sur le bouton dcy pour que la plateforme reparte et continue son chemin

Les choix technologiques sont liés aux éléments à la fois disponibles dans le kit et dans la bibliothèque Simulink®.

- Pour détecter la présence d'obstacle sur le trajet on utilisera un capteur à ultrasons.
- Le bouton dcy correspondra au bouton central de la brique.
- Les codeurs des deux servomoteurs utilisés délivrent la position angulaire.
- L'alarme correspond au buzzer de la brique
- Le voyant sera celui associé au bouton central de la brique

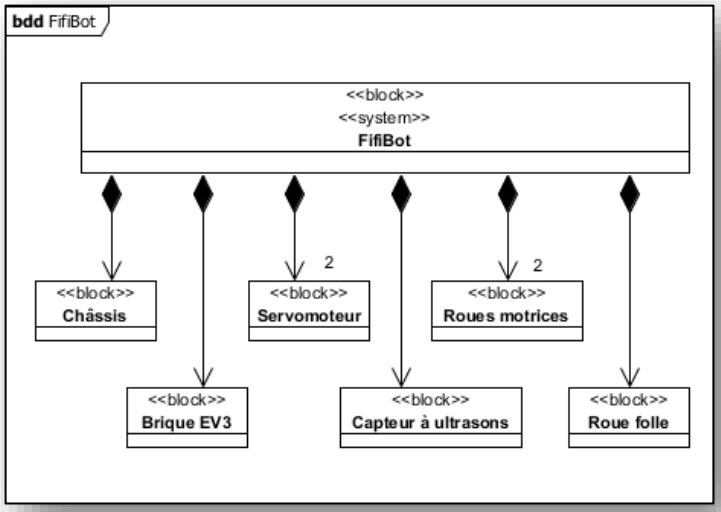
## 11.3.2 Présentation de la structure de la plateforme FifiBot

FifiBot est une plateforme unicycle polyvalente construite avec le kit LEGO® Mindstorms EV3.

Cette plateforme est composée d'un châssis sur lequel sont montés deux servomoteurs qui entrainent indépendamment l'un de l'autre les roues droite et gauche.

Un capteur à ultrasons fixé sur l'avant permet de détecter des obstacles sur le trajet.

Enfin la brique EV3, alimentée par une batterie, permet la gestion des signaux reçus par les capteurs et générés pour commander les actionneurs.



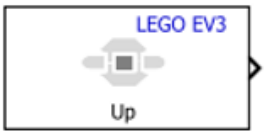
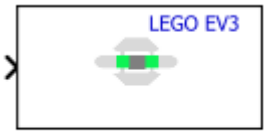

La brique et sa batterie

Les deux motoréducteurs

Le capteur à ultrasons

### 11.3.3 Présentation des éléments de la bibliothèque Simulink®

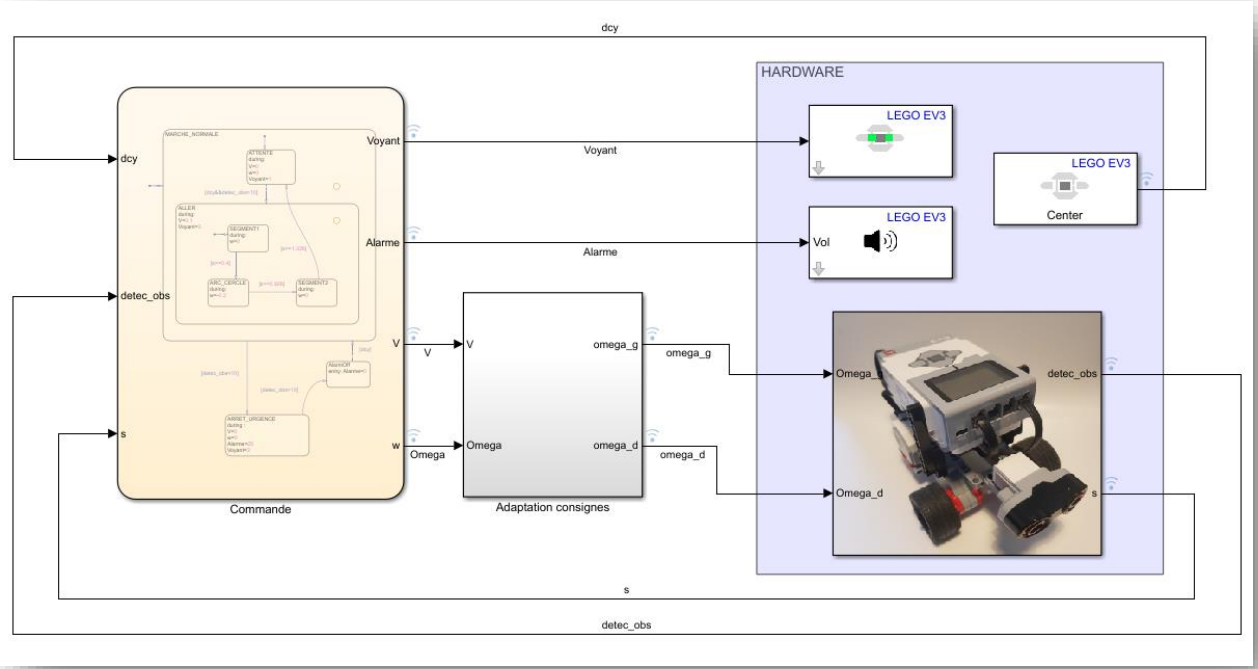
Nous aurons besoin des éléments de la bibliothèque suivants :

| | |
|---|---|
|  <br>Button | Le bouton dcy sera matérialisé par le bouton central de la brique. Le bloc renvoie la valeur 1 si on appuie sur le bouton sinon il renvoie la valeur 0. |
|  <br>Speaker | Nous considèrerons ici uniquement le volume du buzzer. Lorsque l'alarme entrera en service le bloc renverra une valeur de 50 sur une échelle de 0 à 100. La fréquence sera de 100 Hz et la durée du son émis sera de 4 secondes. |
|  <br>Motor | Ce bloc reçoît une valeur comprise entre 0 et 100 correspondant au pourcentage de la tension délivrée par la batterie utilisé pour alimenter les moteurs. |
|  <br>Encoder | Ce bloc renvoie la mesure de l'angle en degrés parcouru par l'arbre de sortie du servomoteur. |
|  <br>Ultrasonic Sensor | Ce capteur à ultrasons permet la mesure de la distance d'un objet de 0 à 255 cm. |
|  <br>Status Light | La couleur du voyant dépend de la valeur renvoyée par le bloc :<br>• 1 : vert fixe<br>• 2 : rouge fixe<br>• 3 : orange fixe |

### 11.3.4 Préparation de l'environnement HIL

L'environnement Hardware-In-the-Loop permet le pilotage de la plateforme réelle à partir de Simulink®. Aucun programme n'est chargé dans la brique.

Le principal intérêt de cette stratégie est d'adapter en temps réel la commande de la plateforme.



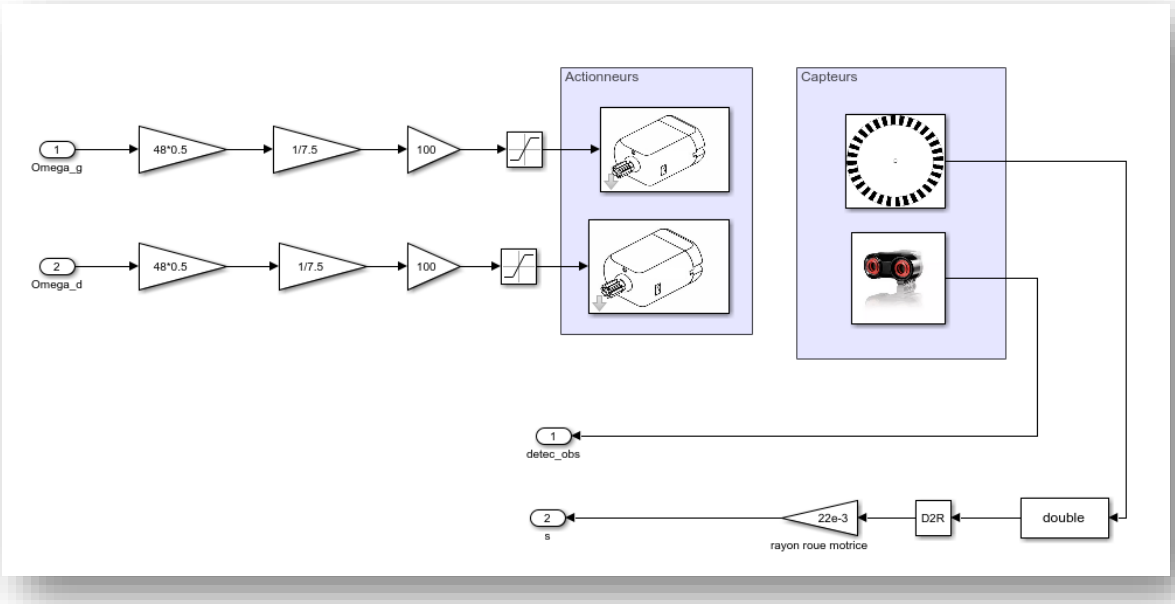

Après avoir présenté les éléments matériels de la plateforme, nous allons maintenant adapter la commande pré-établie à cette nouvelle plateforme en changeant les paramètres géométriques comme la voie et le rayon des roues motrices, d'une part, et en adaptant l'alimentation des servomoteurs d'autre part.

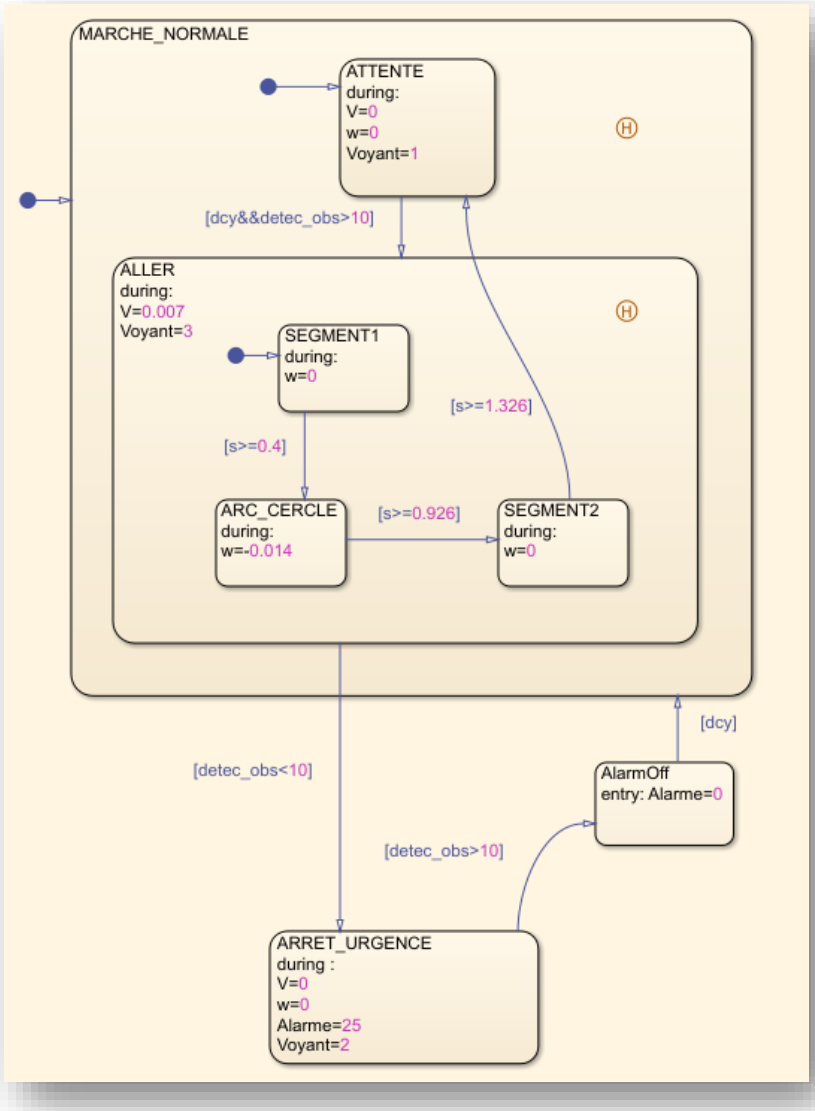
La commande des servomoteurs s'exprime en pourcentage de la tension délivrée par la batterie. Par exemple, à la commande 50 correspond une tension de 50% de la tension délivrée par la batterie. Par conséquent pour une même commande en % la tension de commande dépend de l'état de charge de la batterie.

La vitesse de déplacement maximale est obtenue pour une tension de charge de la batterie maximale soit pour 9V.

Quelques données utiles :

- Le rapport de transmission du réducteur des servomoteurs est de 1:48.
- La constante de fem du moteur des servomoteurs vaut : $k_e$ = 0.5 V/s.
- Le rayon des roues motrices est de 22 $10^{-3}$ m.
- La voie est de 0.1 m
- La vitesse maximale est de 8.25 10-3 m/s




Dans notre application, la tension de la batterie est de 7.5 V ce qui permet à la plateforme d'atteindre 7 $10^{-3}$ m/s.

Les résultats obtenus avec cette commande sont plutôt satisfaisants. Néanmoins, les moteurs droit et gauche n'ont pas rigoureusement le même comportement et donc ne tournent pas rigoureusement à la même vitesse. Cela impacte légèrement les trajectoires.

Pour corriger ce problème il conviendrait de mettre en place un asservissement de vitesse des deux moteurs. Cette étude sort du cadre de cet ouvrage.

Nous venons donc de voir qu'une commande préalablement établie dans un environnement MIL peut être réutilisée en l'adaptant dans un environnement HIL.

Une fois les exigences vérifiées, il est alors possible de télécharger le programme dans la cible pour un fonctionnement en mode embarqué.

# Pour conclure…

Cet ouvrage reprend l'essentiel des fonctionnalités disponibles dans Stateflow® qui associé à Simulink® reste simple d'utilisation et permet le traitement de problématiques complexes dans un même environnement logiciel. Il permet la conception et la simulation de la dynamique des systèmes à évènements discrets mais aussi des systèmes hybrides.

L'environnement Simulink® Stateflow® permet de combiner des représentations graphiques et tabulaires, dont des diagrammes d'états (Charts), des diagrammes de flux (Flow Charts), des tables d'état transition (State Transition Table) et des tables de vérité (Truth Table), pour modéliser la dynamique d'un système qui réagit à des événements et/ou des conditions.

Stateflow® inclut une animation des diagrammes d'états pour une meilleure analyse du comportement du système étudié. Il permet aussi de suivre l'évolution d'un diagramme de séquence.

D'un point de vue méthodologique, et comme nous avons pu l'entrevoir dans les derniers chapitres, Stateflow® trouve naturellement sa place dans la démarche Model-Based-Design.

Enfin, comme je le suggère dans le sous-titre de cet ouvrage de découverte et d'initiation, j'invite tous les lecteurs qui ont apprécié son contenu, à le partager, à le mettre entre toutes les mains, entre celles des curieux et celles de tous ceux qui s'intéresse à la modélisation et à la simulation du comportement dynamique des systèmes à évènements discrets et des systèmes hybrides.