WP7 : METHODE SUPPORT POUR LE BON USAGE DE MATLAB/SIMULINK/STATEFLOW

RAPPORT N° CS/311-1/AJ000212/RAP/02/05/14 Version 1.0

Préparé par

CS SI
Division Opérationnelle Ingénierie Scientifique
Centre Opérationnel France Nord
16 avenue Galilée
92350 LE PLESSIS ROBINSON

Rédigé dans le cadre du projet

ACOTRIS

	NOM(S)	DATE(S)	VISA(S)
REDACTEUR(S)	N. TRUONG TRUNG	24/05/02	
VERIFICATEUR(S)	C. MODIGUY		
APPROBATEUR			

FICHE DE SUIVI DES MODIFICATIONS

Version/Révision		Références		Description des modifications	Auteur(s)
Indice	Date	Page	N° §		
1.0	24/05/02			Version initiale	N. TRUONG TRUNG

LISTE DE DIFFUSION

DIFFUSION EXTERNE

DIFFUSION INTERNE

Archivage DO **SCSA**

DOCUMENTS APPLICABLES ET DE REFERENCE

DOCUMENTS APPLICABLES

DOCUMENTS DE REFERENCE

SOMMAIRE

1. INTRODUCTION

Ce document a pour objectif premier de présenter une méthode support à l'utilisation des outils Matlab / Simulink / Stateflow.

Matlab est un puissant langage permettant en autre de simuler des modèles physiques et de visualiser leurs comportements.

Simulink est l'outil graphique associé à Matlab. Le système modélisé est alors construit sous forme de diagrammes et de schémas blocs. Comme Matlab, Simulink permet de simuler et de visualiser le comportement dynamique du système modélisé.

Stateflow intégré à Simulink, apporte enfin la prise en compte des aspects comportementaux et évènementiels, nécessaire à la modélisation des machines à états.

Bien sûr, la méthode est largement décrite dans ce document, à travers notamment la décomposition du modèle mais aussi à travers les règles syntaxiques qui régissent la construction du modèle.

Afin d'aborder la méthode de manière plus concrète, la méthode sera appliquée à un régulateur de vitesse.

Au-delà de l'aspect méthodologie, ce support vise une certaine cohésion entre l'approche automatique et informatique vis à vis des systèmes de contrôle et de régulation, fréquemment rencontrés dans le domaine industriel. La méthode doit ainsi faciliter le transfert de la modélisation du système considéré sous Matlab vers la conception sous UML.

2. ORGANISATION DU MODELE

2.1 CONTEXT DIAGRAM

L'objet de ce chapitre consiste à définir le contexte dans lequel le système modélisé est amené à être simulé. L'environnement comprend, suivant les besoins, les points énoncés ci-dessous :

- le fichier de calibration,
- les échanges de données via le Workspace de Matlab,
- Les entrées/sorties du système modélisé.

A ce niveau, nous obtenons le diagramme suivant :

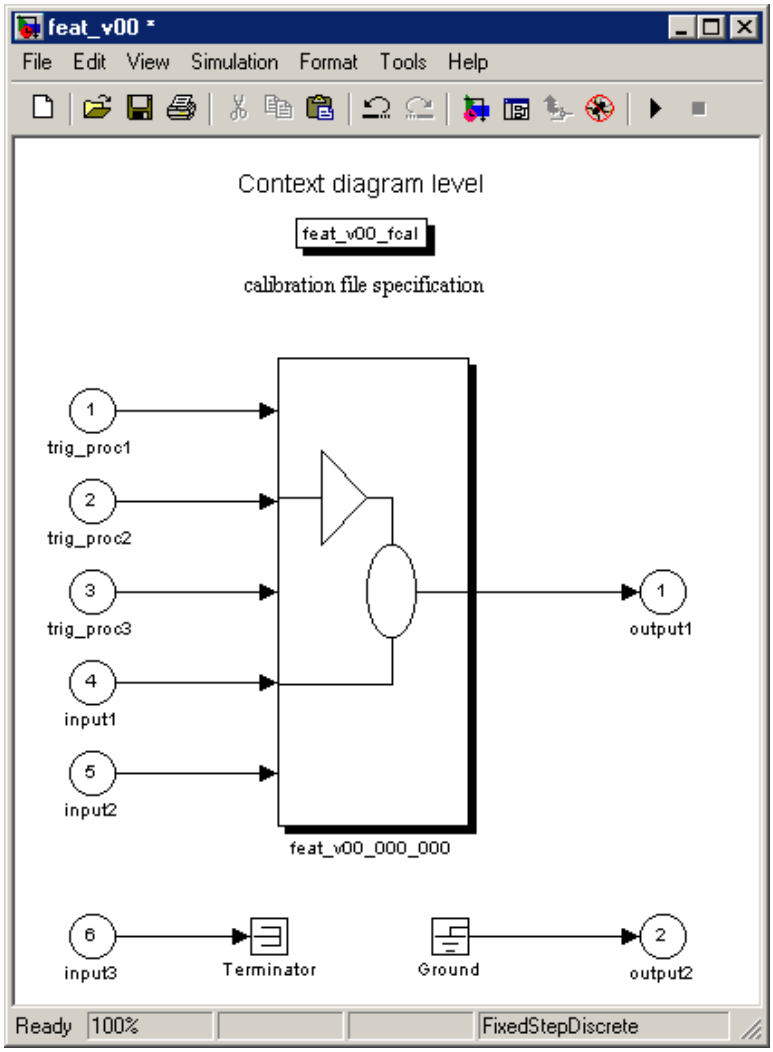


Figure 1 : Context Diagram

Sur la figure précédente, vous pouvez observer que le nom du fichier d'initialisation se trouve au-dessus du message **calibration file specification**. Ce fichier contient toutes les initialisations de constantes nécessaires au fonctionnement du système.
Les échanges de données via le Workspace sont uniquement autorisés dans le cadre de constantes partagées entre différents systèmes. Leur présence au niveau du Context Diagram assure ainsi une certaine traçabilité des données partagées. Néanmoins, il est nécessaire de s'assurer qu'un signal de ce type possède une terminaison avant d'être utilisé dans un quelconque calcul dans ce système.

Noter que le nom du système modélisé **feat_v00_000_000** a une signification **feat** constitue l'abréviation du projet, et **v00_000_000** constitue le numéro complet de la version.

2.2 EXECUTION CONTEXT DIAGRAM

La vue Execution context diagram vise à décomposer le système en ses grandes fonctionnalités. Ainsi, chaque fonctionnalité est représentée par un sous-système. Les phases de conception et de simulation s'en retrouvent alors simplifiées. La décomposition est réalisée suivant les différents évènements, triggers et fonctionnalités génériques du système.

Le diagramme ci-dessous présente un exemple de Execution Context Diagram. Le système se décompose en deux sous-systèmes, lesquels assurent des traitements spécifiques. Vous pouvez remarquer qu'il est possible d'échanger des données entre différents sous-systèmes.

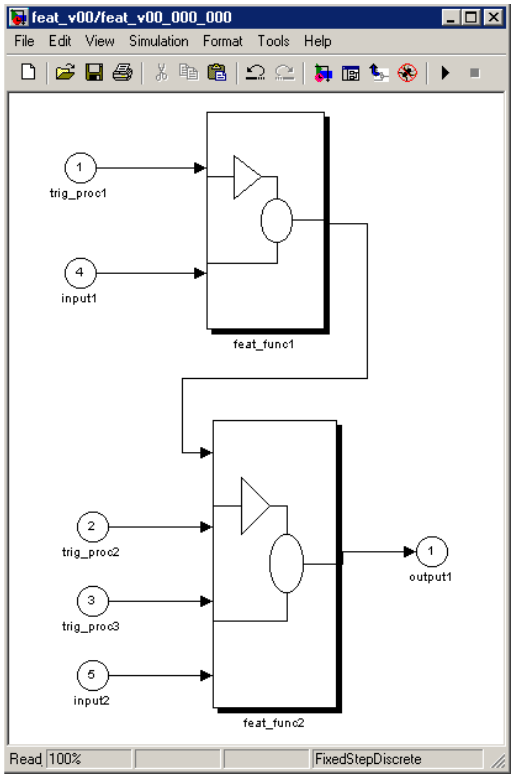


Figure 2 : Execution Context Diagram

2.3 DECOMPOSITION HIERARCHIQUE

Chaque sous-système du Context Execution Diagram donne lieu à un sous-diagramme, composé d'un C-Spec (pour Control Specification) et d'un P-Spec (pour Process Specification).

Un C-Spec assure généralement le traitement des évènements et des triggers associés à un sous-système. Un P-Spec assure uniquement quant à lui un traitement sur les données.

Chaque sous-système possède un unique C-Spec mais peut en revanche posséder plusieurs P-Spec. Un C-Spec peut être uniquement défini sous forme de flux de données sous Stateflow. A l'inverse, un P-Spec peut aussi bien être construit sous la forme de Schémas Blocs ou de flux de données sous Stateflow.

La figure 3 présente le sous-système **feat_func1** de la vue Execution Context Diagram.

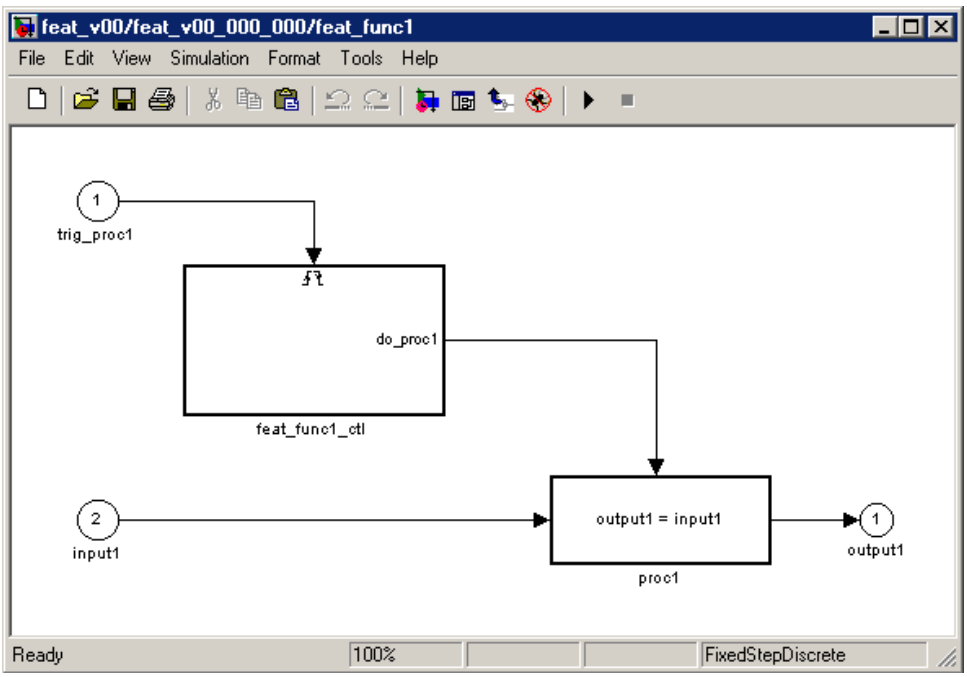


Figure 3 : Sous-système

Le choix a été fait dans ce cas très simple de modéliser le C-Spec **feat_func1_ctl** sous Simulink car nous ne disposions pas de Simulink. L'entrée 1 **trig_proc1** est un événement. L'entrée 2 **input1** constitue une donnée nécessaire au calcul assuré par le P-Spec **proc1**.

Lors de la réception d'un trigger **trig_proc1**, le C-Spec **feat_func1_ctl** génère un appel de fonction. Ce signal va alors déclencher le traitement du P-Spec **proc1**.

Les figures 4 et 5 ci-dessous montrent respectivement le détail des C-Spec et P-Spec associés au sous-système considéré.

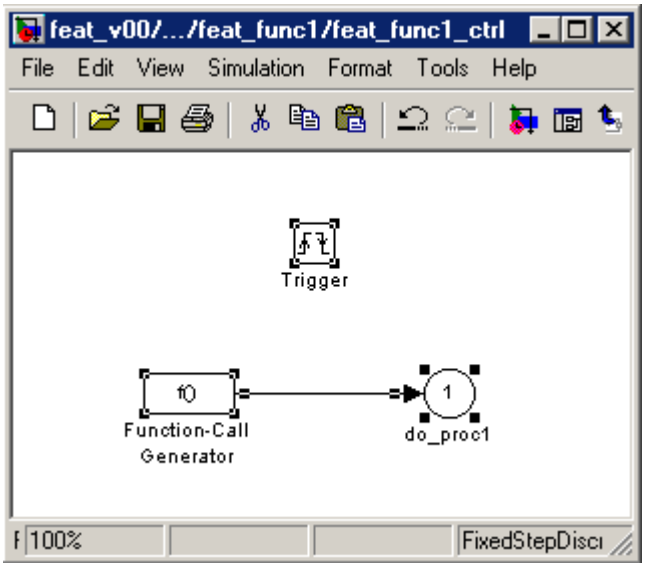


Figure 4 : C-Spec feat_func1_ctrl du sous-système feat_func1

La figure précédente permet de voir comment générer un appel de fonction sous Simulink en utilisant un bloc Function-Call Generator.

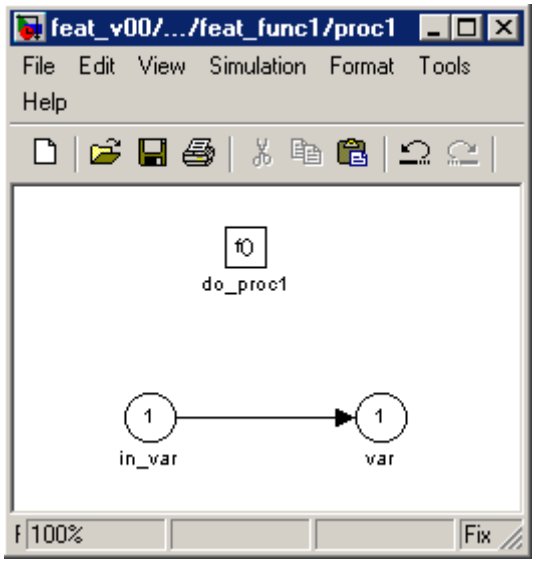


Figure 5 : P-Spec proc 1 du sous-système feat_func1

Le diagramme ci-dessus constitue le traitement sur les données qu'effectue le P-Spec **proc1**. Ce dernier envoie simplement sur la sortie **var** la valeur qu'il voit à son entrée **in_var**.

2.4 RESUME DES REGLES DE DECOMPOSITION

La liste suivante récapitule les principales règles.

- un sous-système effectuant une opération uniquement sur les données est un P-Spec;
- un P-Spec doit être masqué et présente sur ses sorties uniquement des données;
- un P-Spec peut se décomposer, pour des raisons de lisibilité et de modularité, en sept niveaux et sous-niveaux au maximum;
- tous les P-Spec doivent être déclenchés par des appels de fonctions;
- tous les triggers de P-Spec sont référencés par **do_<pspec subsystem block name>**;
- un appel de fonction provenant d'un C-Spec déclenche systématiquement un P-Spec, il est relié au port trigger du P-Spec graphiquement positionné au dessus du bloc;
- tous les C-Specs doivent être déclenchés par des évènements, des triggers ou des appels de fonction reliés au port supérieur du diagramme Stateflow;
- tous les C-Spec présentent sur leurs sorties des appels de fonction;
- chaque sous-système représentant une couche hiérarchique dans le modèle possède au moins une entrée **trig_<block name>**, entrant par le coté gauche et considérée comme la ou les première(s) entrée(s) du bloc;
- le C-Spec référencé **<block name>_ctl**, doit recevoir le signal **trig<block name>** sur le port supérieur input_event;
- seuls les caractères alphanumériques et l'underscore sont autorisés,
- chaque nom de données doit être unique, chaque variable uniquement nécessaire à la modélisation doit comporter le préfixe **ml_**;
- tout processus périodique doit utiliser une constante référencée **ML_<feat>_TASK<#>DT**, où <#> est le nombre entre 1 et n si le système comporte n processus périodiques. Cette période doit être défini dans le Workspace de la manière suivante : ML_<feat>_TASK1DT=0.01.

3. REGLES DE MODELISATION SOUS SIMULINK

3.1 MASQUAGE DES P-SPEC

Tous les P-Spec doivent posséder un masque et une forme de pseudo code les décrivant. Ceci facilite l'utilisation du diagramme sous Simulink dans le package de documentation.

Pour ce faire, cliquer droit sur le P-Spec considéré, puis sélectionner le menu **Edit Mask**. La fenêtre apparue est la suivante :

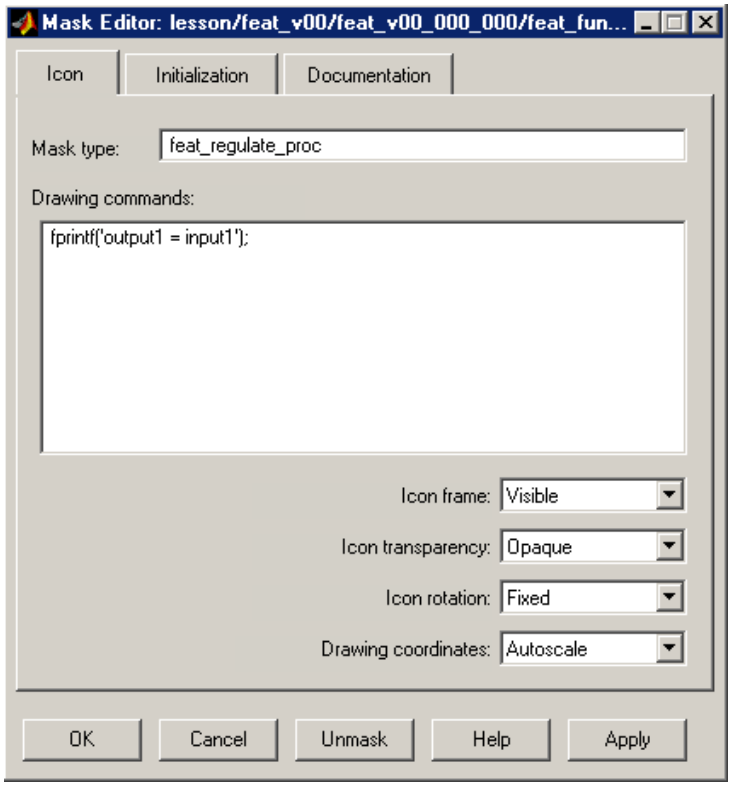


Figure 6 : Editeur de masque

Le texte entré dans la zone **Drawing Commands** permettra de connaître le traitement de ce processus sans avoir à visualiser le contenu du diagramme. La figure ci-dessous montre alors l'apparence du P-Spec obtenue.

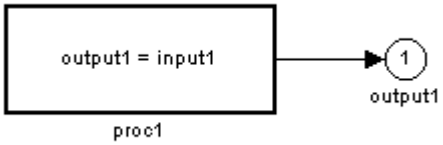


Figure 7 : Apparence d'un P-Spec possédant un masque

L'onglet **Documentation** de cette même fenêtre **Mask Editor** permet de visualiser la fenêtre suivante :

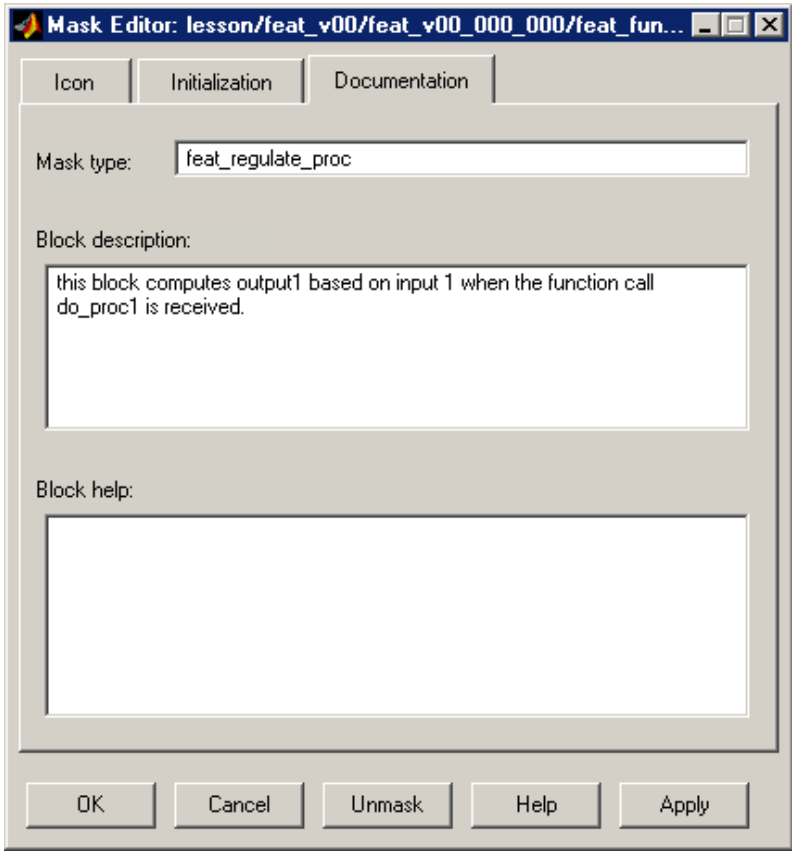


Figure 8 : Description du bloc

La description du bloc peut être saisie dans la zone **Block description**. L'effet d'un double-clic sur le P-Spec affiche la figure 9.

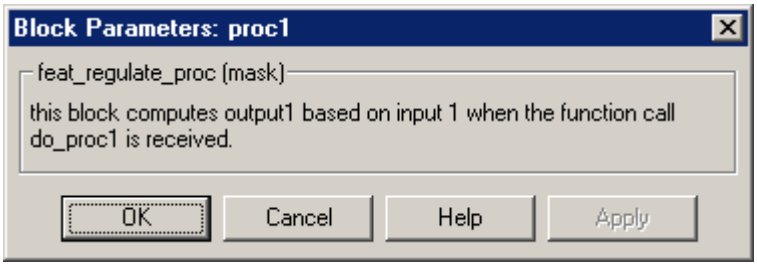


Figure 9 : Description du P-Spec

L'onglet **Initialization** de la fenêtre figure 6 permet de visualiser la fenêtre suivante :

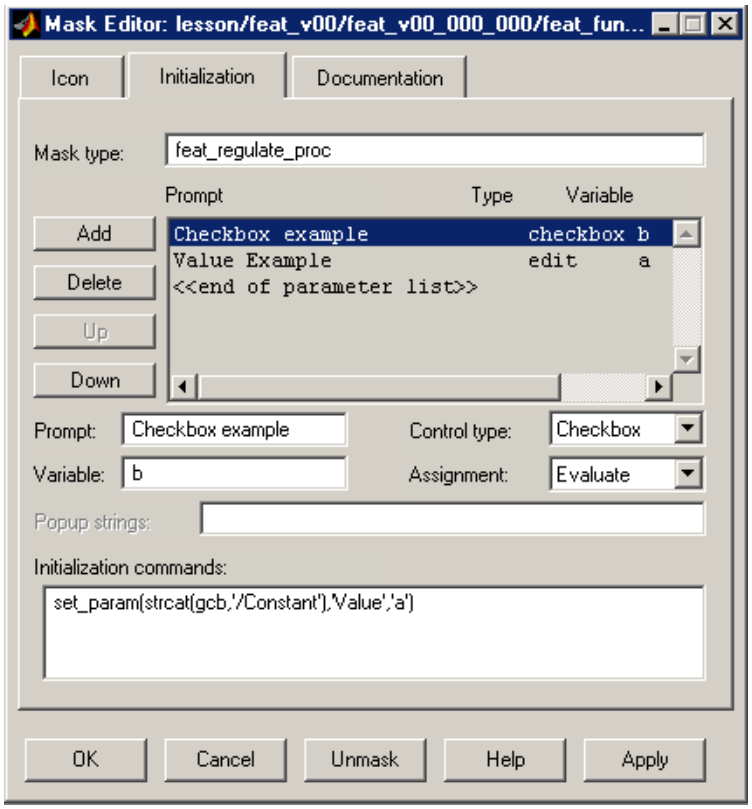


Figure 10 : Lien des paramètres avec le Workspace

La saisie des informations dans le champ **Initialization commands** permet de définir une variable **a** utilisé dans le traitement du P-Spec. Noter qu'une variable **b** a été créée, ceci afin de mémoriser la valeur du checkbox. De ce fait, le Workspace n'est plus accessible directement au bloc sous le masque. Ce sont les informations entrées dans la boite de dialogue ci-dessous qui assurent la valeur de la variable **a**. Cette fenêtre est obtenue en double cliquant sur le P-Spec proc1. La variable du Workspace XYZ constitue donc la valeur qui est passée à la variable **a**.

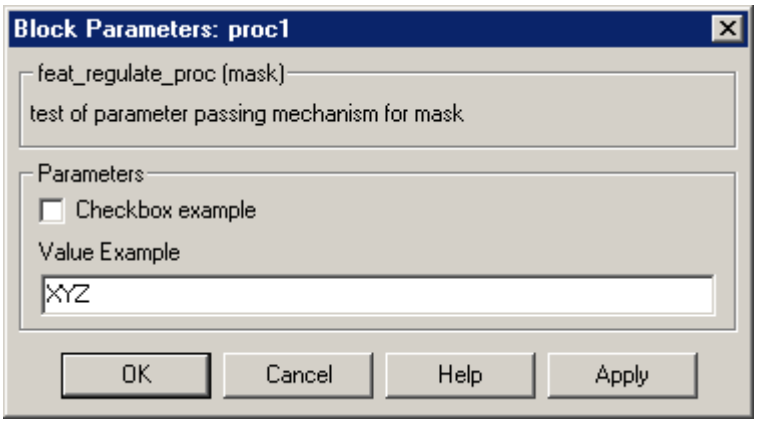


Figure 11 : Description et initialisation des variables du P-Spec

3.2 MEMORISATION DES DONNEES

3.2.1 Data Storage

Il est strictement interdit d'utiliser les Data Storage dans la modélisation des algorithmes. L'utilisation de ce type de fonction peut mener à un écrasement non désiré des données. Veuillez utiliser de préférence les blocs Delay ou les blocs Merge.

3.2.2 Delay

Afin d'utiliser la dernière valeur passée de n'importe quel paramètre au sein d'un même diagramme P-Spec, utiliser un 1/z Discrete Time Delay comme le montre la figure suivante.

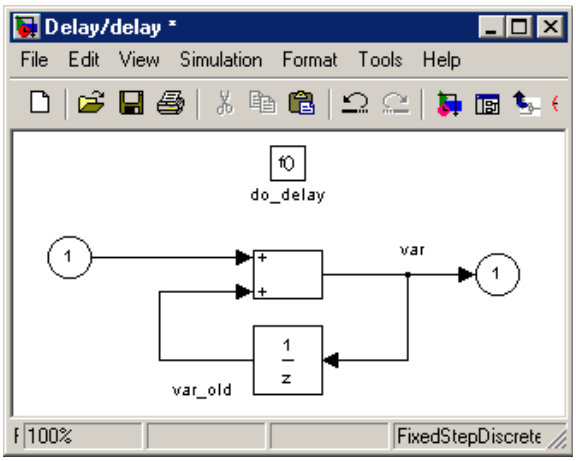


Figure 12 : Utilisation du bloc Delay pour la mémorisation des données

Lorsqu'un événement **do_delay** est reçu, **var= input1+var_old** est l'opération effectuée, où **var_old** est la valeur de **var** au dernier événement **do_delay** reçu. Il est possible de fixer, via la fenêtre ci-dessous, la valeur initiale de la sortie et la période d'échantillonnage du bloc Delay. Dans le cas présent, la période d'échantillonnage doit être égale à **-1**, afin qu'elle soit héritée du bloc parent et donc de l'événement **do_delay**.

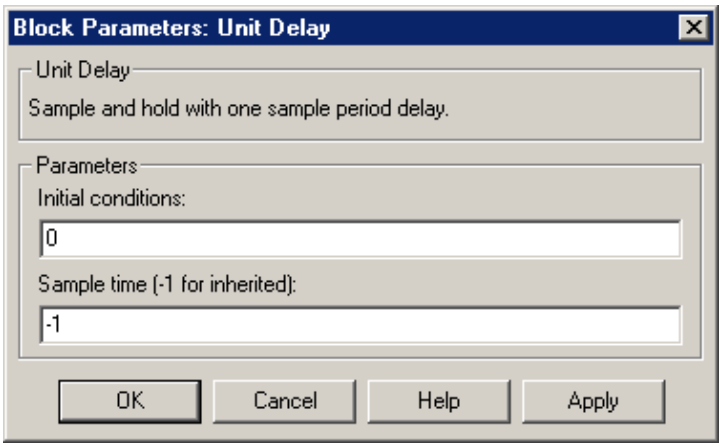


Figure 13 : Configuration du bloc Delay

3.2.3 Merge

L'opération Merge doit être utilisée lorsqu'un paramètre est écrit dans plus d'un sous-système. Cette opération permet de fusionner en quelque sorte une donnée qui, soit serait écrite dans plusieurs Execution Context, soit nécessite différents modes de calcul à l'intérieur d'un sous-système (figure 14).

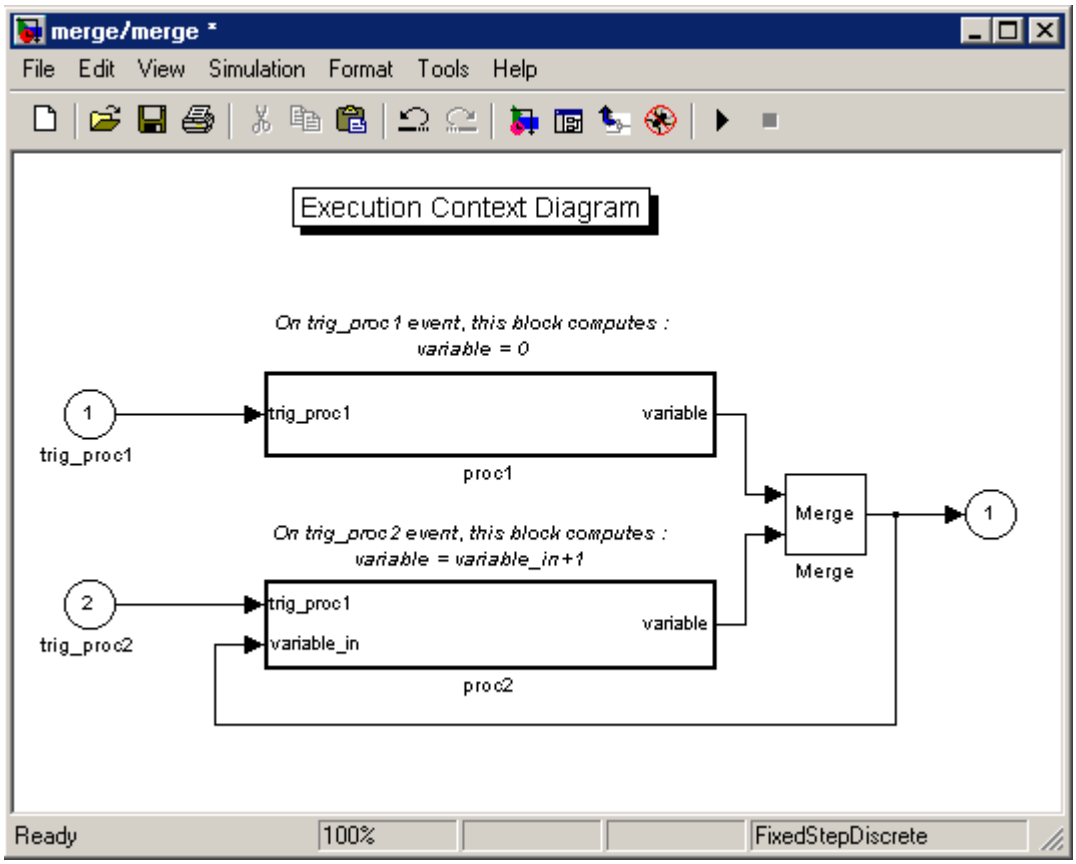


Figure 14 : Utilisation du bloc Merge

Il est à noter qu'il strictement interdit d'utiliser deux blocs Merge en cascade. Il est conseillé d'utiliser un bloc Mux, surtout si chaque traitement a besoin du paramètre pour effectuer son traitement. Ce Mux doit se trouver à un niveau plus bas dans la décomposition hiérarchique du modèle, comme le montre la figure page suivante.

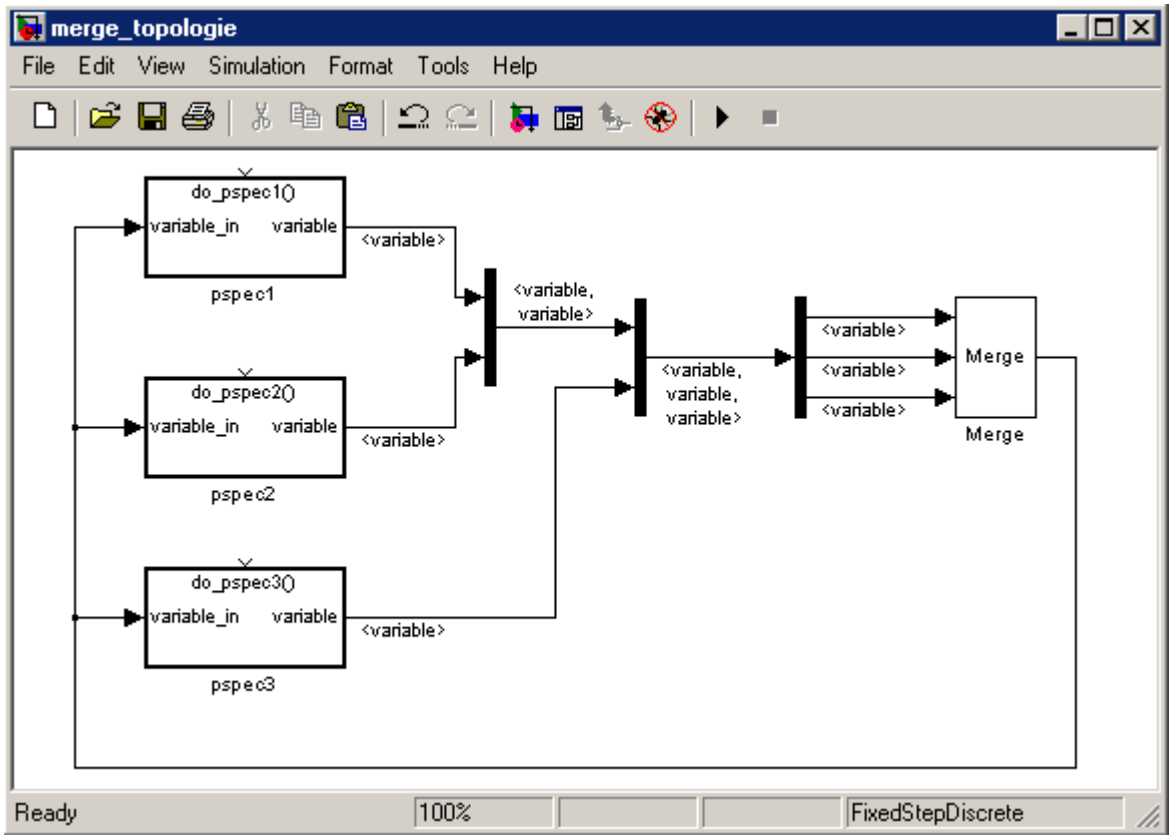


Figure 15 : Combinaison de blocs Mux et Merge

Dans le cas ci-dessus, l'utilisation du bloc Mux n'est pas nécessaire. Elle l'est que lorsque le signal traversent les limites des sous-systèmes, à travers ses ports de sortie.

Il est à noter que les entrées et la sortie d'un bloc Merge sont par définition un même et unique signal, aussi épier l'entrée un bloc Merge n'a aucun sens et n'est pas autorisé par Simulink. Cependant, il est toujours possible, quoique non conseillé, de venir lire la valeur du Merge (figures 16 et 17 page suivante).

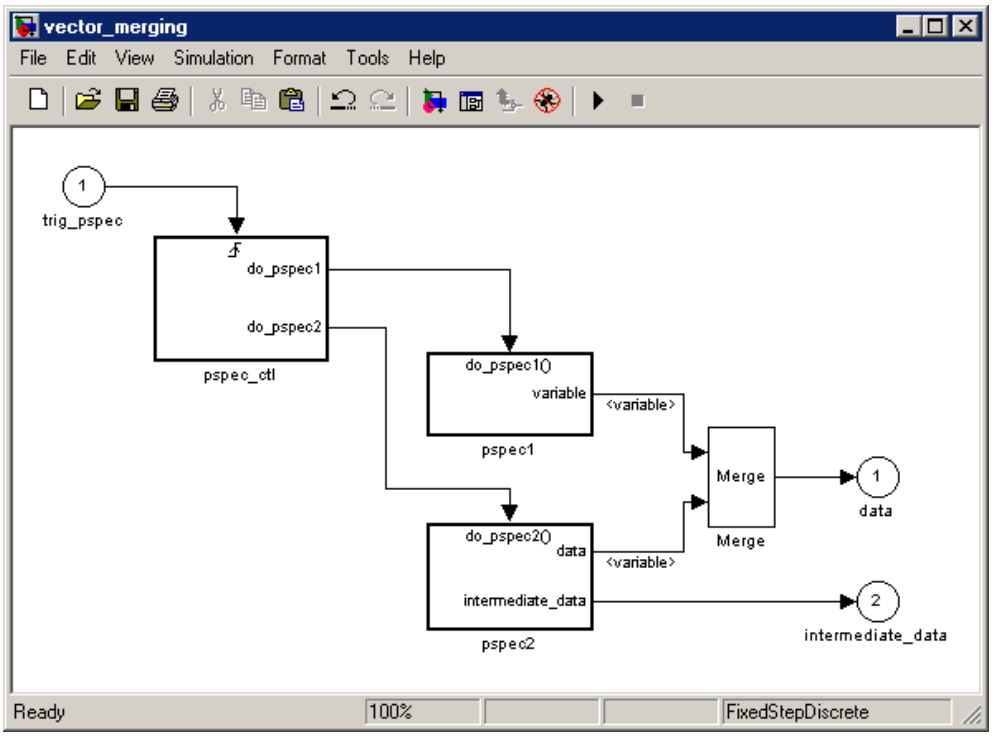


Figure 16 : Lecture de la valeur en entrée d'un Merge 1

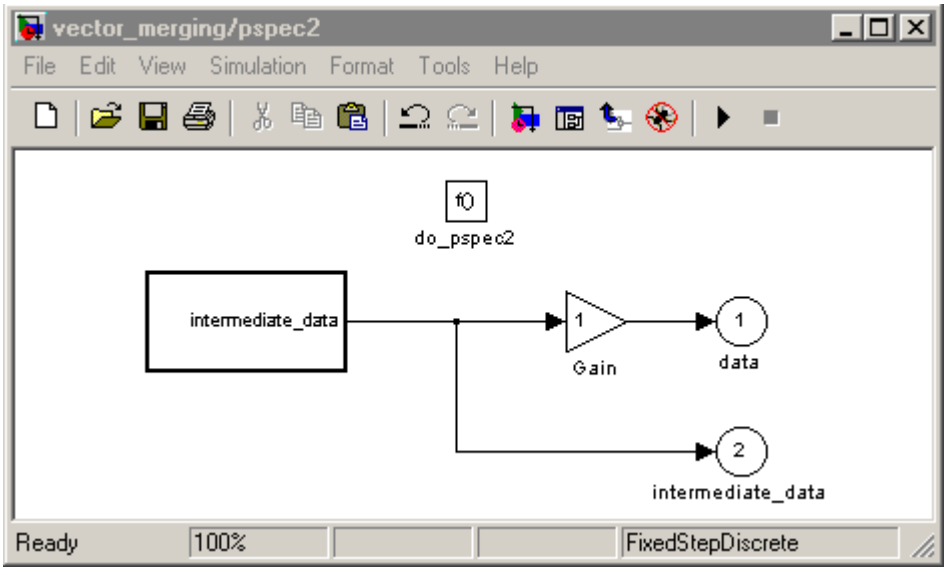


Figure 17 : Lecture de la valeur en entrée d'un Merge 2

Noter que l'insertion d'un gain est nécessaire pour distinguer les signaux **data** et **intermediate_data**.

Par ailleurs, Simulink interdit de connecter plusieurs ports non-virtuels sur un unique port appartenant à un bloc Merge. Ainsi, le diagramme ci-dessous est incorrect.

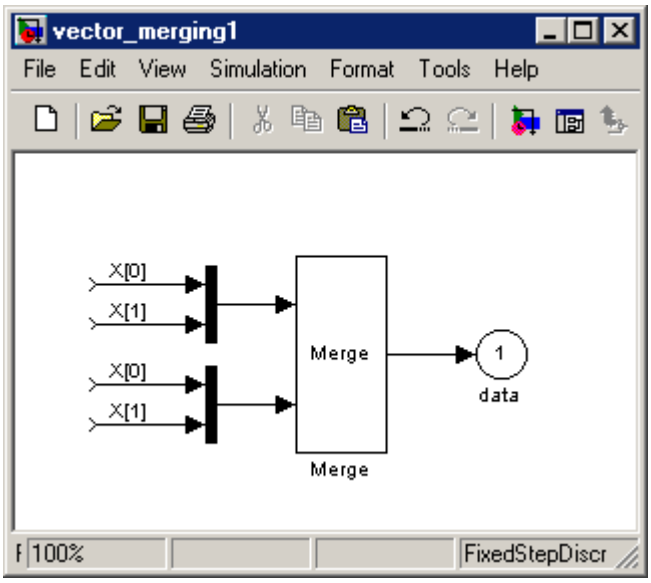


Figure 18 : Utilisation incorrecte du Merge

La première solution consiste à envoyer sur chaque entrée du bloc Merge une sortie de type vector d'un bloc non virtuel Stateflow ou Multiport DE. Cette solution est applicable uniquement si le signal en entrée du Merge est traité par le même bloc non virtuel.

Dans les autres cas, il faut donc construire manuellement un bloc Merge traitant les vecteurs (figures 19 et 20 page suivante).

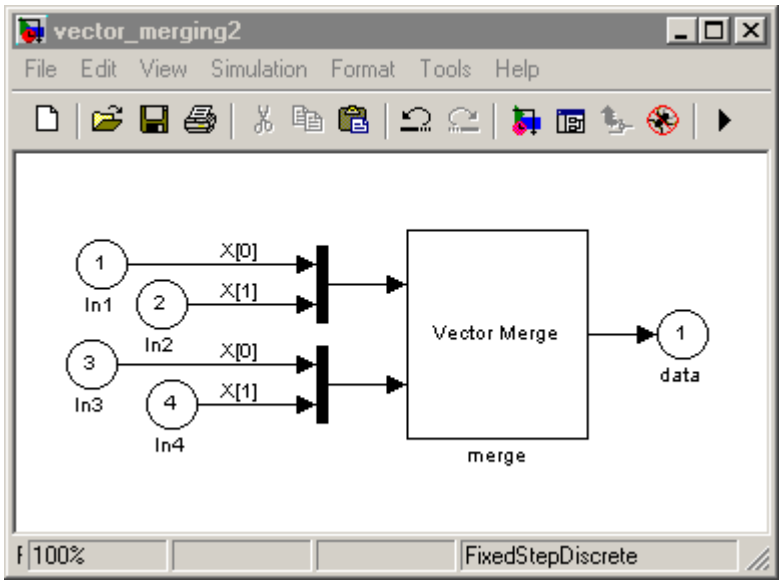


Figure 19 : Vecteurs en entrée du sous-système composé de blocs Merge

Noter qu'un masque est appliqué au sous-système Vector Merge. Le détail de ce sous-système est décrit ci-dessous.

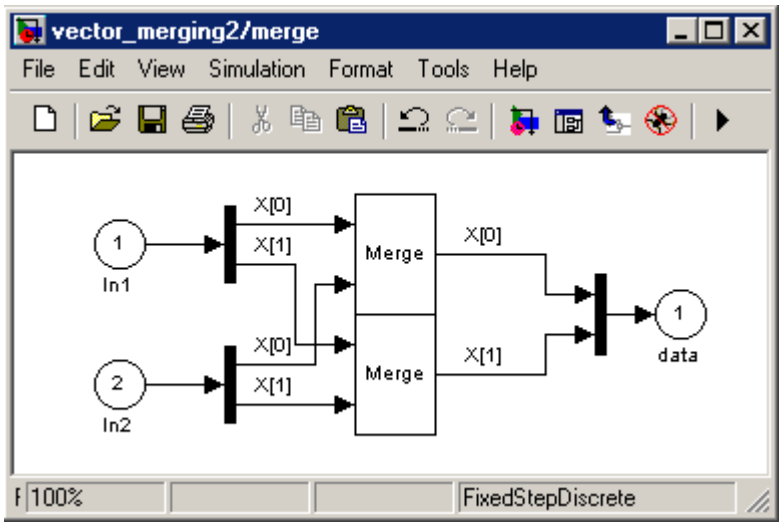


Figure 20 : Détail du sous-système Vector Merge

3.3 LES SIGNAUX

3.3.1 Noms de signaux

Les flux entrant et sortant du Feature Context Level doivent posséder des Labels.

Les signaux qui représentent des variables spécifiques à la modélisation qui ne doivent pas obligatoirement apparaître lors du codage possèdent le préfixe **ml_**.

Les blocs Contants ne sont pas considérés comme des flux de données, aussi les signaux provenant de tels blocs ne doivent pas posséder de Labels, que ce soient des constantes initialisables ou non. Tout bloc Constant possède un nom unique, aussi le rajout de Label sur le signal sortant n'apporte rien de plus.

Les signaux, entrant ou sortant d'un sous-système masqué, doivent posséder un Label.

Si un sous-système est utilisé une seule fois, chacun de ses ports doit porter un nom identique au Label du signal connecté. Il n'est pas en revanche nécessaire d'afficher le nom du port. L'information serait en effet redondante. Si un sous-système est utilisé plusieurs fois, alors le nom du port doit être visible et différent du Label du signal connecté.

Enfin, dans le cas ou un paramètre est en même temps un signal d'entrée et de sortie, alors le port entrant doit comporter le suffixe **_in**.

Les signaux de type événement ou encore les tics d'horloge déclenchant des processus périodiques doivent comporter les préfixe **trig_**.

Seuls les caractères alphanumériques et l'undescore '_' sont autorisés pour nommer les signaux.

3.3.2 Règle de visualisation des données

Tous les Labels de signaux sous Simulink et sous Stateflow (données de flux et évènements) peuvent être visualisés au niveau local et global.

3.3.3 Goto/From Tags

L'utilisation de ces connecteurs permet de ne pas croiser différents signaux afin de rendre plus lisible les diagrammes du point de vue graphique. Tout Tag doit être connecté à un signal.

Les Tags doivent seulement posséder des noms descriptifs, faisant appel au bon sens afin que ces derniers ne soient pas trop long. Pour rendre les diagrammes encore plus lisibles, il est possible de jouer sur les couleurs des Tags.
Les Tags peuvent être utilisés pour interrompre au niveau local tout type de signaux, événements ou données. Si le Goto/Form Tag est connecté à un événement, alors le signal ne possède pas obligatoirement un label.

3.3.4 Vectors

Il est possible de regrouper plusieurs signaux en utilisant le bloc Mux. Ceci rend ainsi un diagramme plus lisible. Un bloc Mux constitue par ailleurs un moyen de construire sous Simulink un tableau de données.

L'utilisation du bloc Mux répond aux exigences suivantes :

- tout signal connecté à un bloc Mux ou Demux possède un Label ;
- si un Mux/Demux représente un tableau, les signaux connectés doivent posséder un Label de la forme **array[n]** ou n représente l'indice du tableau et array le nom de base du tableau ;
- si le signal vectorisé est un tableau alors, son Label porte le nom du tableau, sinon il porte un nom comportant le préfixe **ml_** ;
- des vecteurs peuvent être multiplexés, cela donne lieu à des vecteurs de vecteurs,
- toute donnée vectorisée doit être démultiplexée avant d'entrer dans un Stateflow ;
- tout vecteur composé de données indépendantes, ne devant pas être considéré dans le logiciel final comme un tableau, doit apparaître dans le Context Diagram comme des scalaires individuels ;
- un port d'entrée-sortie représentant un vecteur doit porter le nom du tableau qu'il représente si ce vecteur est un tableau, et porter un nom descriptif si vecteur est une collection de données vectorisées ;
- le multiplexage de données et de Function calls est interdit,
- si un tableau est une sortie directe d'un bloc Stateflow, le bloc Bus Selector ne peut être utilisé, utiliser le bloc Demux ;
- les connexions Bus à Bus sont interdites ;
- il est interdit de multiplexer des Function calls, excepté ceux qui constituent une entrée directe pour un bloc Stateflow ou un port trigger.

Lors de l'utilisation de vecteurs de vecteurs, la définition de la taille respective de chacun des vecteurs a la plus grande importance, sans quoi des erreurs apparaîtraient. Considérons l'exemple de la page suivante, et plus particulièrement les multiplexeurs de gauche à droite.

Le tableau ci-dessous résume les vecteurs définis dans l'exemple.

numéro du vecteur	Composition	Définition
1	3 scalaires	[1,1,1]
2	1 vecteur de 3 scalaires 2 scalaires	[3,1,1]
3	1 vecteur 1 scalaire	[1,1]

Tableau 1 : Définition des vecteurs

La figure 22 présente quant à elle le paramétrage du bloc Bus Selector, associé au diagramme de la figure 21.

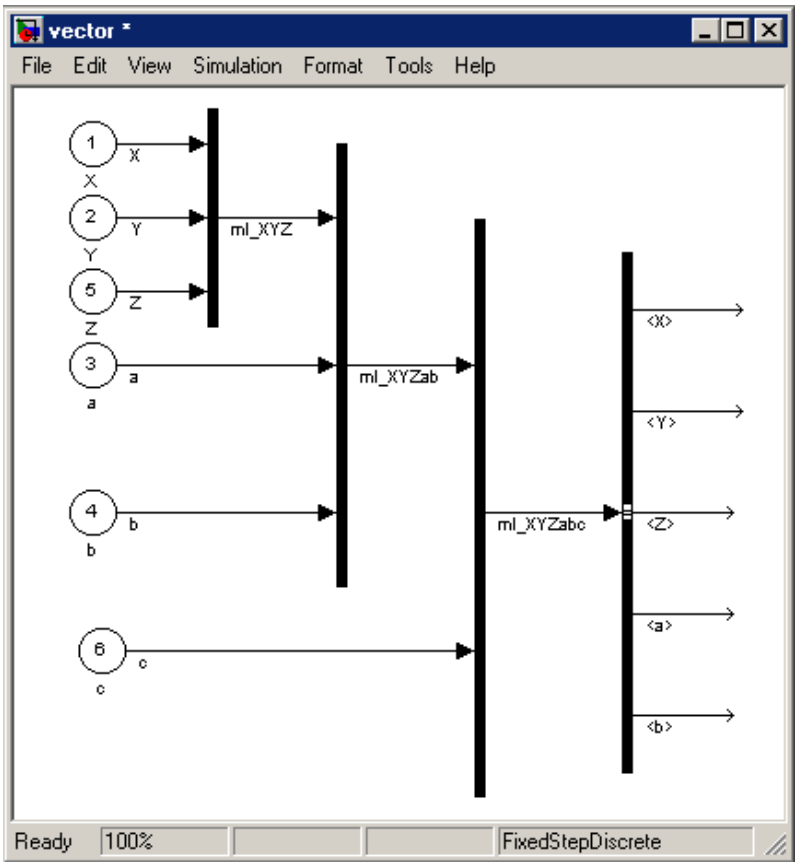


Figure 21 : Exemple de Blocs Mux en cascade

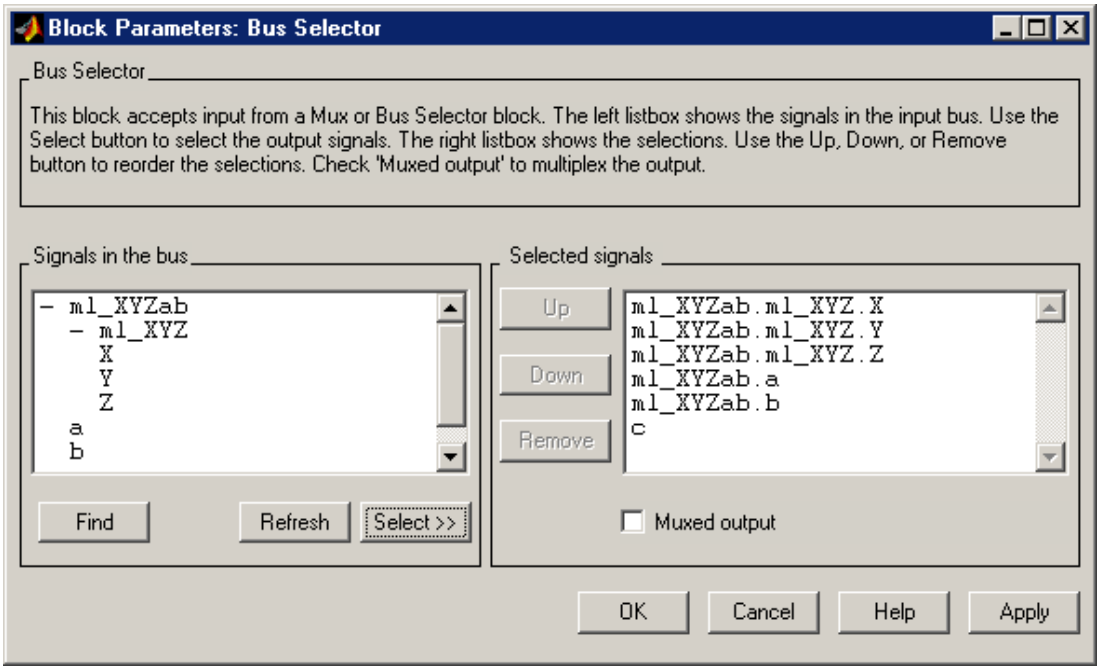


Figure 22 : Paramètre du bloc Bus Selector

3.4 S-FUNCTION

Les S-Functions, ou le code développé par l'utilisateur peuvent être inclus dans un modèle uniquement si :

- L'implémentation S-Function est un bloc appartenant à la librairie Utils. Cette implémentation peut avoir été réalisée en langage M ou C.
- L'implémentation S-Function est un bloc n'appartenant à aucune librairie, implémentée en langage C uniquement. L'implémentation en langage M est strictement interdit car elle risque d'occasionner des interférences avec les flux de bas niveaux entre processus. Ce type d'implémentation doit être utilisé en dernier recours lorsque Simulink ou Stateflow n'apportent plus de solution.

4. REGLES DE MODELISATION SOUS STATEFLOW

Le présent chapitre spécifie comment l'outil Stateflow peut être utilisé pour modéliser aussi bien les C-Spec que les P-Spec.

Bon nombre de stratégies existantes repose principalement sur l'utilisation de la logique combinatoire indépendamment de l'utilisation des états.

Stateflow combine quant à lui l'utilisation de la logique combinatoire et celle des machines à états finis.

4.1.1 Flow Chart

Il est recommandé d'utiliser le flowchart lorsque aucune information claire n'existe concernant le système modélisé.

4.1.2 Les concepts de base

La figure suivante présente les concepts élémentaires du flowchart:

- lorsqu'un flowchart est sollicité, il débute par défaut au point noir en haut de la figure;
- par la suite, c'est une combinaison d'états (cercles) et de transitions (flèches);
- chaque transition possède, placé(s) graphiquement à sa droite, une ou plusieurs condition(s) et un ou plusieurs traitement(s) associé(s);
- une transition ne possède pas obligatoirement un traitement associé;
- si un état donne lieu à plusieurs transitions possibles, la lecture se fait dans le sens anti-trigonométrique et la dernière transition ne doit pas posséder de condition, sa présence est d'ailleurs obligatoire pour le bon fonctionnement de Stateflow même si elle n'est pas utile du point de vue de la modélisation ;
- un flowchart doit posséder un unique point d'entrée et un unique point de sortie.

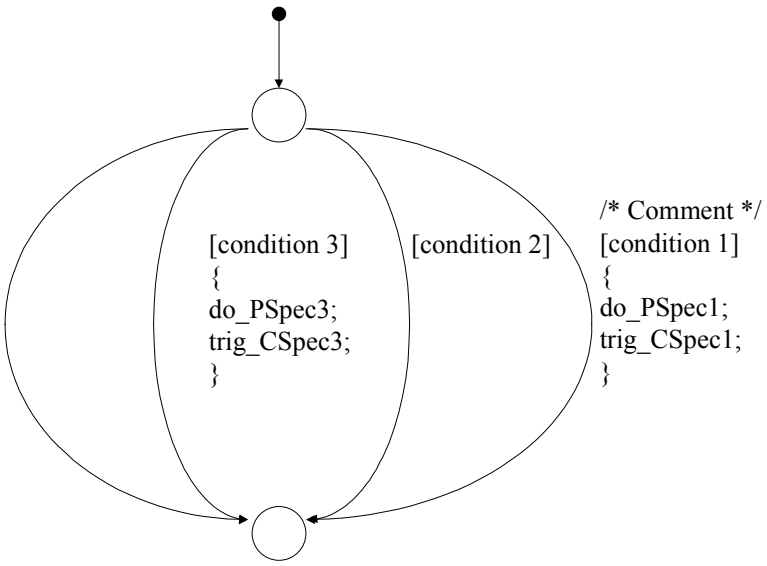
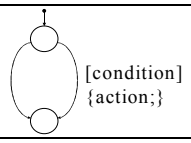
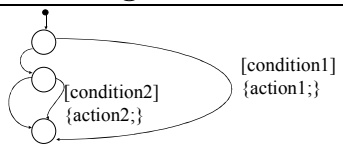
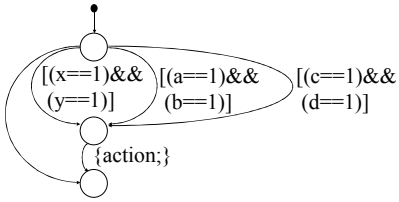
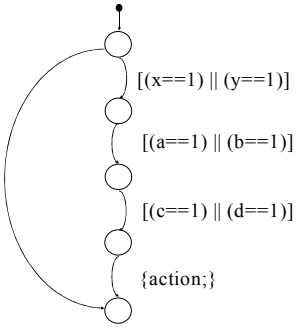
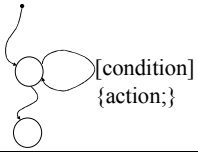
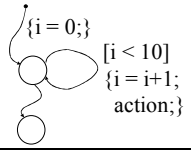


Figure 22 : Concepts de base des flowcharts

4.1.3 Flowchart autorisés

Le tableau ci-dessous présente les différents diagrammes élémentaires état/transition autorisés.

Description	Représentation FlowChart	Equivalent en langage C
If-Then		If (condition) {action;} else{}
If-Then-Elseif		If (condition1) {action1;} else if (condition2) {action2;} else{}
OR gate (3 branches parallèles sans action). Noter la branche obligatoire sans condition		If ((x==1 && y==1)|| (a==1 && b==1)|| (c==1 && d==1)) {action;} else {}
AND gate (3 branches en cascade sans action). Noter la branche obligatoire sans condition		If ((x==1 || y==1)&& (a==1 || b==1)&& (c==1 || d==1)) {action;} else{}
While loop		While (condition) {action;}
For loop		For (i = 0; i < 10; i++) {action;}

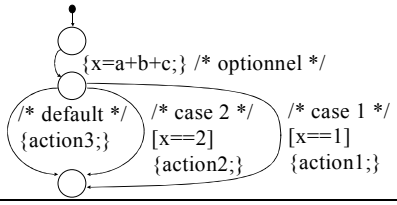
Switch case
Noter que les actions ne sont pas obligatoires et qu'elles ne doivent pas être des function calls.
		Switch (x)
		{
		case 1: action1; break;
		case 2: action2; break;
		default : action3; break;
		}

Tableau 1 : Diagramme élémentaire d'état/transition

Noter que toutes les combinaisons de boucles sont permises pourvu que cela ne se traduise pas dans le code par un goto.

4.2 MACHINE A ETAT

La figure suivante montre un exemple très simple de machine à deux états.

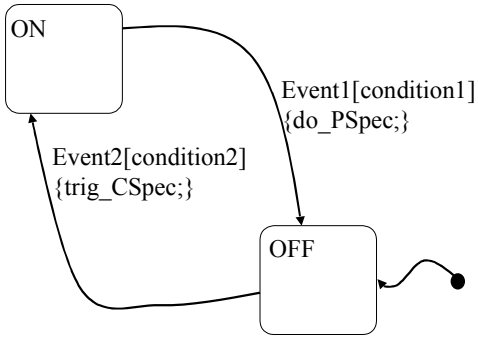


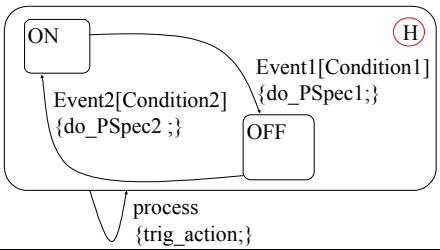
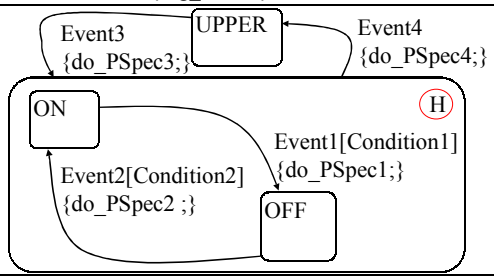
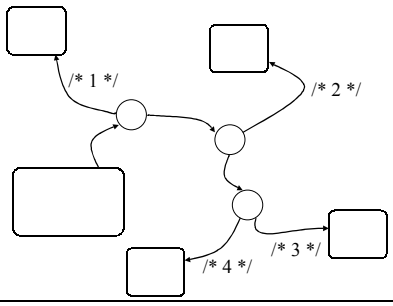
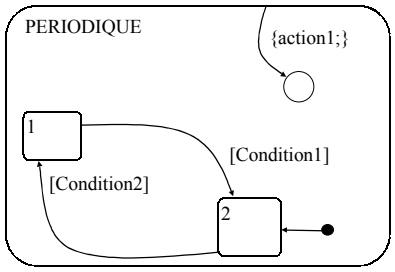
Figure 23 : Exemple de machine à deux états

Les deux états de cette machine sont On et Off.
L'état par défaut est l'état Off, comme le montre le point noir à l'extrême droite du Stateflow.
Le passage d'un état à l'autre est obligatoirement déclenché soit par l'évènement **Event1** soit par l'événement **Event2**.
Tous les évènements doivent déclencher des function calls sauf si ces évènements sont uniquement utilisés pour simuler le comportement de l'appareil ou les entrées du système modélisé.
Les conditions **condition1** et **condition2** sont en revanche optionnelles.

4.3 DIAGRAMME D'ACTIONS DE TRANSITION ET LES ETATS/FLUX

4.3.1 Structure d'un flowchart

L'objet de ce chapitre est d'utiliser les flowcharts et les machines à états en vue de décrire un système. Le tableau ci-dessous présente les diagrammes élémentaires.

Description	Représentation flowcharts
History Junction (Cercle rouge) Quel que soit l'état du système, cet argument permet de lancer un traitement, puis de retourner dans l'état pré-traitement.	
Hiérarchie Cette figure illustre le concept de hiérarchie d'état. Lorsque l'évènement **Event4** est reçu, le système se retrouve dans l'état UPPER, et ce quel que soit l'état dans lequel il se trouve.	
Transition Etat/Etats Si un état possède plusieurs transitions, l'ordre de leur scrutation doit être clairement défini. Le numéro en commentaire est optionnel.	
Opération périodique Il est possible de modéliser un process périodique action1 comme le montre la figure correspondante, qui assure aussi le passage de l'état 1 à l'état 2 ou vice versa.	

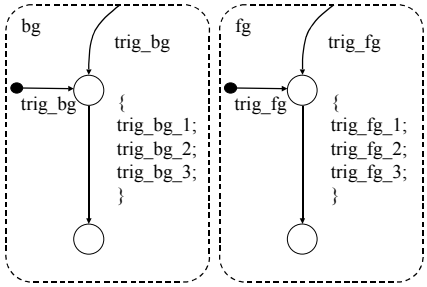
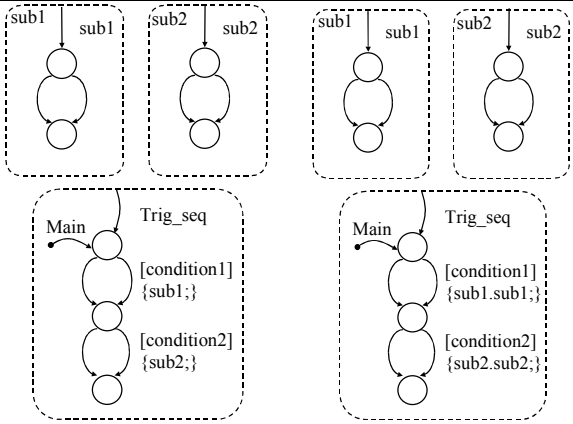
Concurrence La concurrence ou le parallélisme peut être représenté de la manière suivante. Le parallélisme des tâches permet aussi d'obtenir un modèle plus compact. Il est extrêmement important que chaque chemin d'accès à un processus parallèle possède un garde. Stateflow réveille en effet la machine à états à chaque événement reçu. Ceci explique le garde **trig_bg** dans la tâche **bg**.	
Sous-programme La figure de gauche constitue une autre manière d'utiliser le parallélisme des tâches. Le diagramme de gauche correspond donc à un appel de sous-routine. Le diagramme Main doit être le dernier à s'exécuter sinon une erreur d'initialisation se produirait. Si les évènements **sub1** et **sub2** sont vus au niveau du diagramme, alors la tâche Main doit posséder un garde de sorte à ne pas se lancer sur un autre événement. Si les évènements **sub1** et **sub2** sont vus directement au niveau des sous-tâches alors l'événement broadcast peut être utilisé et son nom est **state_name.event_name**, comme le montre la figure de droite.	

Tableau 2 : Structure élémentaire d'un flowchart

4.3.2 Règles de syntaxe

Le tableau suivant résume les préfixes obligatoires.

Nom du préfixe	Description
in(state_name)	Une fonction qui retourne un booléen utilisable dans un conditionnel.
change(data_name)	Une fonction qui retourne un booléen utilisable dans un conditionnel.
t	Temps depuis le début de la simulation
exit:	Mot clé attaché à un label d'état utilisé pour spécifier les actions à effectuer avant de quitter un état.

Tableau 3 : Préfixes obligatoires

Le tableau suivant résume les mots clé interdits.

Mots Clé interdit
during :
entry(state_name)
entry :
exit(state_name)
on event_name
send(event_name, state_name)
matlab()
matlab.MATLAB_Workspace_data

Tableau 4 : Mots Clé interdits

Le tableau suivant présente les fonctions liées à des compilateurs que le concepteur peut utiliser.

Mots clé	Description de la fonction
abs()	calcule la valeur absolue d'un entier
fabs()	calcule la valeur absolue d'un réel
min()	recherche la valeur minimale dans un tableau
max()	recherche la valeur maximale dans un tableau
log()	calcule le logarithme de base 10

Tableau 5 : Fonctions utilisables

Les transitions sous Stateflow peuvent être annotées (/* annotation */).
Pour rendre le diagramme encore plus clair, il est possible d'utiliser un BOX afin de regrouper certaines entités. Il ne correspond à rien du point de vue fonctionnel de la machine à états. En revanche, un BOX affecte l'ordre d'activation des états parallèles des diagrammes. Ainsi, les BOXES sont activés avant l'activation des états parallèles, de gauche à droite et de haut en bas. L'activation des états parallèles suit aussi le même ordre de gauche à droite et de haut en bas.

4.3.3 Autres considérations

Le tableau ci-dessous présente les diagrammes interdits et la solution de remplacement éventuelle.

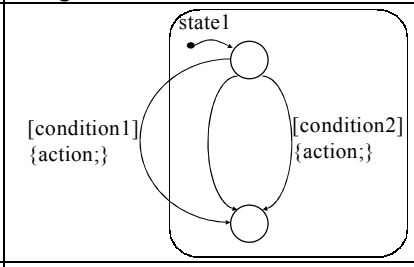
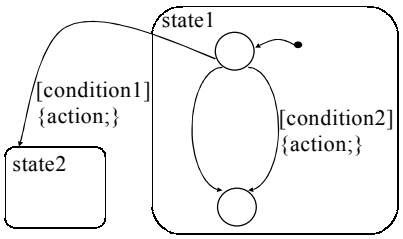
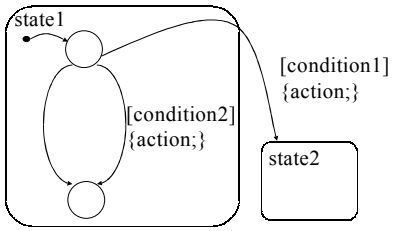
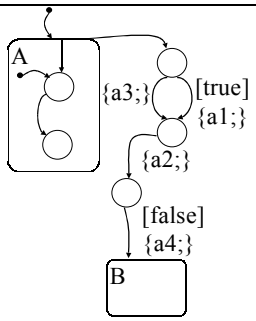
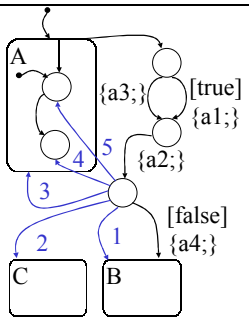
Description	Diagramme interdit	Solution préférée
Transition quittant et ré-entrant un état.		Pas de solution.
Transition quittant un état La solution assure la cohérence entre l'ordre d'exécution du modèle et la règle des transitions multi-branches (lecture dans le sens des aiguilles d'une montre).		
Répétition d'une action due au fonctionnement de Stateflow Lorsque la condition [true] est rencontrée, l'action **a2** est effectuée. Si [false] est rencontrée, alors Stateflow va retracer la logique et exécuter l'action **a3**, puis **a2** de nouveau. Pour palier à ce problème, il faut ajouter une transition sans garde parmis les 5 transitions bleu présentées dans la solution.		

Tableau 6 : Autres considérations concernant les diagrammes d'état/transition

4.4 SIGNAUX

4.4.1 Evènement

Les évènements peuvent être visualisés. Pour ce faire, cliquer sur le menu **Add** de l'éditeur Stateflow. L'événement est ainsi défini dans le Stateflow Data Dictionnary. Par la suite, les évènements et les données seront alors visibles via le menu **Tools ➔ Explore**.

Tous les évènements externes à un bloc Stateflow doivent être de type function call. Un événement de type function call est un lien unique point à point.

Les évènements représentant l'environnement (le monde externe) doivent être obligatoirement un front montant, descendant ou les deux, ceci afin de simuler le comportement par interruption du système d'ordonnancement.

Un signal de type Event provenant d'un port Output doit posséder un Label portant le nom du P-spec ou du C-spec recevant l'évènement. Dans le cas où ce même signal serait relié à un connecteur Goto/From, seul le connecteur et non pas le signal porte un Label.

Enfin, il est à noter que les broadcasts d'événement sont interdits.

4.4.2 Data Dictionnary

Le dictionnaire contient les données et les évènements utilisés sous Stateflow. C'est le seul moyen d'interfacer les entrées/sorties entre les blocs Stateflow et Simulink.
Les éléments du Data Dictionnary sont directement référencés dans le diagramme Stateflow. A moins que l'élément donnée soit nouveau pour le Ford Data Dictionnary, il est interdit de définir un attribut pour un élément donnée.

Pour consulter le contenu du Stateflow Data Dictionnary, un explorateur de modèle existe dans le menu **Tool ➔ Explore**.

4.4.3 Nom des ports

Les Data Dictionnary Entries sont constitués par des noms de port. En retour, les noms de port sous Stateflow sont référencés dans la spécification graphique de la fonction sous Stateflow.

Si un diagramme Stateflow possède une instance unique, alors les noms des ports doivent être identiques aux noms des signaux Simulink pour des raisons d'une part de modélisation et d'autre part de cohérence avec le Data Dictionnary. Si un diagramme possède plusieurs instances, alors les noms de port doivent être suffisamment descriptifs et bien sûr différents des noms de signaux connectés.

Si un paramètre est une entrée mais aussi une sortie pour un diagramme Stateflow, alors la convention suivante doit être appliquée.

Si le paramètre est modifié dans un seul bloc Stateflow et pas dans un autre bloc Simulink ou Stateflow alors il doit être déclaré dans l'explorateur Stateflow comme un output. Si le paramètre est modifié dans plusieurs blocs, il doit alors être aussi défini comme un input avec le même nom suivi du suffixe **_in**.

4.4.4 Types de données

Il existe deux classes distinctes de données. La première catégorie regroupe les variables de flux de contrôle. Ce sont des signaux discrets et énumérés (flags, types énumérés, compteurs, événement etc…).

Les variables de flux de données sont des variables continues associées à des données continues (vitesse, température etc…).

Les données initialisables peuvent être aussi bien des variables discrètes que continues.

Un soin particulier doit être pris concernant le type des données afin d'assurer une cohérence lors de l'utilisation de bloc conditionnel. Le type de données par défaut (réel-double) dans l'explorateur de Stateflow ne doit être modifié uniquement si des opérations sur les bits ou d'autres opérations spécifiques doivent être effectuées.

Enfin le tableau ci-dessous présente les différentes déclarations de variables sous Matlab et leur correspondance en langage C.

Matlab	Langage C
Local	static
Temporary	variables automatiques de fonction, initialisées à chaque appel de fonction.
Workspace (Variable initialisable)	
Constant	#define

Tableau 7 : Correspondance du type de variables Matlab/Langage C

La description de la méthode est présentement terminée. Nous allons maintenant appliquer la méthode support Matlab/Simulink/Stateflow à un exemple.

5. EXEMPLE DU REGULATEUR DE VITESSE

5.1 CAHIER DES CHARGES

L'exemple choisi en vue de l'application de la méthode est volontairement simple. Il permet à terme de maîtriser la démarche et de découvrir les grands axes de la méthode.

Le régulateur de vitesse a pour fonction de réguler la vitesse d'un véhicule en fonction de la consigne de vitesse fixée lors de la mise en marche de la régulation.

Le régulateur possède les entrées suivantes :

- un bouton pressoir On/Off,
- une pédale d'accélérateur qui envoie un signal lorsqu'elle est pressée ou relâchée,
- une pédale de frein qui envoie un signal lorsqu'elle est pressée ou relâchée,
- un switch moteur off,
- la vitesse courante du véhicule.

Le régulateur de vitesse possède une sortie unique, à savoir une consigne de couple, qui suit la loi de commande suivante :

$$\Delta T = \frac{1}{2} * \text{arctang}\,(k * (\text{Consigne de vitesse} - \text{Vitesse courante}))$$

Outre ces considérations, le régulateur doit :

- se lancer ou s'arrêter si le conducteur presse le bouton On/Off;
- s'arrêter lorsque la pédale de frein est pressée;
- se mettre en veille lorsque la pédale d'accélérateur est pressée, la régulation étant en marche;
- se remettre en route lorsque la pédale d'accélérateur est relâchée à partir de l'état de veille ;
- ne pas se mettre en marche si la vitesse est inférieure à 50 km/h ;
- s'arrêter si la vitesse courante descend en dessous des 50 km/h.

5.2 MODELISATION

5.2.1 Context Diagram

Le Context Diagram est la vue la plus haute du modèle. Le Context Diagram obtenu dans le cas du régulateur de vitesse est décrit ci-dessous. Vous pouvez remarquer que le fichier d'initialisation associé à ce système modélisé se nomme feat_v00_fcal. Aucune donnée ne transite via le Workspace. Le tableau 8 résume les différents signaux en entrée du système.

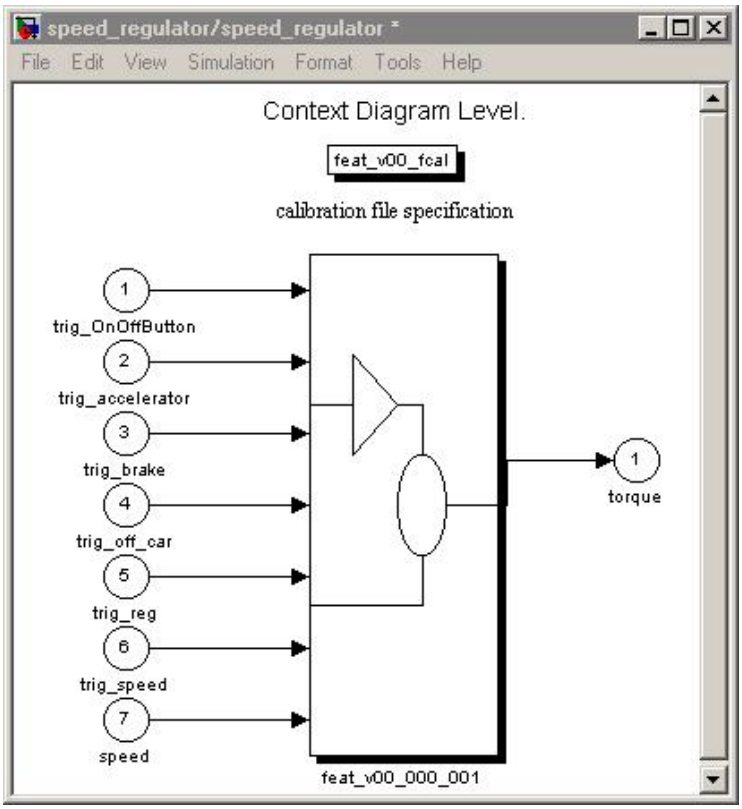


Figure 24 : Context Diagram

Nom du signal	Description
trig_OnOffButton	évènement généré lorsque le bouton On/Off est pressé.
trig_accelerator	évènement généré lorsque l'accélérateur est pressé ou relâché.
trig_brake	évènement généré lorsque le frein est pressée ou relâché.
trig_off_car	évènement généré lorsque le moteur est coupé.
trig_reg	tic d'horloge de la régulation de fréquence 2 Hz.
trig_speed	fréquence d'échantillonnage de la valeur de vitesse à 10 Hz.
speed	donnée continue représentant la vitesse courante.

Tableau 8 : Signaux en entrées du système

5.2.2 Execution Context Diagram

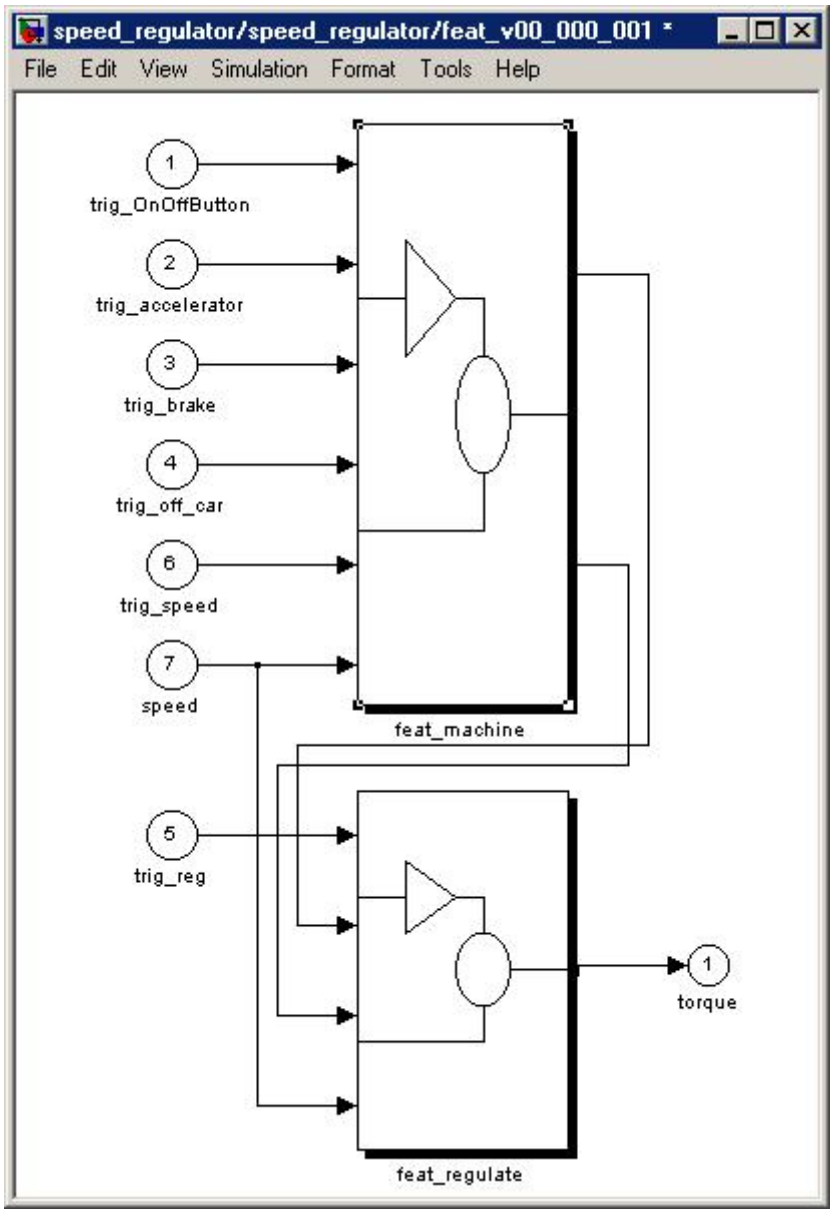


Figure 25 : Execution Context Diagram du régulateur de vitesse

Dans notre exemple, la vue Execution Context Diagram se décompose en deux sous-systèmes. Le sous-système feat_machine assure la gestion des évènements de l'environnement et en déduit la mise en route éventuelle de la régulation en vitesse.

Le second sous-système assure quant à lui la régulation proprement dite à une fréquence de 2Hz, en fonction des données envoyées par le sous-système feat_machine. Les deux sous-systèmes échangent donc des données.

Le découpage en sous-systèmes a été réalisé en vue de séparer au mieux les différents traitements associés à chaque événement.

5.2.3 Sous-système feat_machine

5.2.3.1 Vue générale

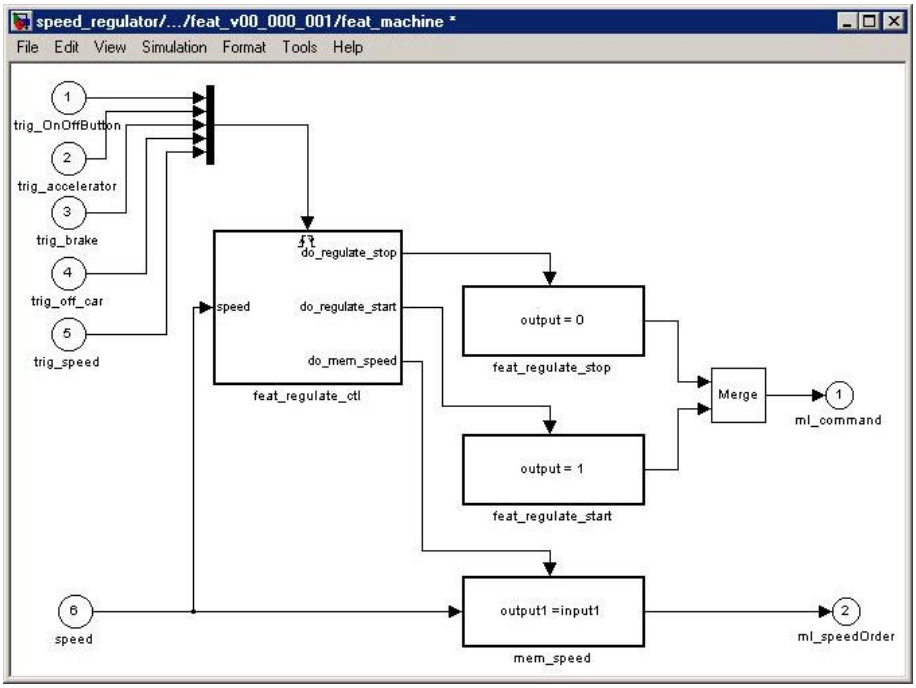


Figure 26 : Description du sous-système feat_regulate

Comme le stipule la méthode, les évènements réveillant le système modélisé sont multiplexés puis envoyés vers le Control-Specification **feat_regulate_ctl**.

Vous pouvez voir qu'il existe trois Process-Specifications. Ces 3 P-Spec sont déclenchés par un appel de fonction demandé par le C-Spec. Ils effectuent respectivement :

- l'arrêt de la régulation (**feat_regulate_stop**),
- la mise en marche de la régulation (**feat_regulate_start**),
- la mémorisation de la consigne de vitesse (**mem_speed**).

Deux P-Spec écrivent dans la même variable **ml_command**, c'est pourquoi la présence du bloc Merge est obligatoire. La variable booléenne **ml_command** est la première donnée envoyée au second sous-système. Ce dernier assurera ou non la régulation en fonction de la variable booléenne. La seconde donnée envoyée au second sous-système est la consigne de vitesse **ml_speedOrder**.

Enfin, dans un souci de compréhension, tous les P-Spec arborent un masque qui décrit en pseudo code leurs traitements respectifs. Ainsi, il n'est pas nécessaire de rentrer dans le sous-système pour connaître le traitement qu'il effectue. Par exemple, le pseudo code du P-Psec **feat_regulate_start** nous renseigne parfaitement sur la nature de son traitement, à savoir qu'il envoie sur sa sortie la valeur 1 lorsque l'appel de fonction **do_regulate_start** est reçu.

5.2.3.2 C-Spec feat_regulate_ctl

Modèle Stateflow théorique

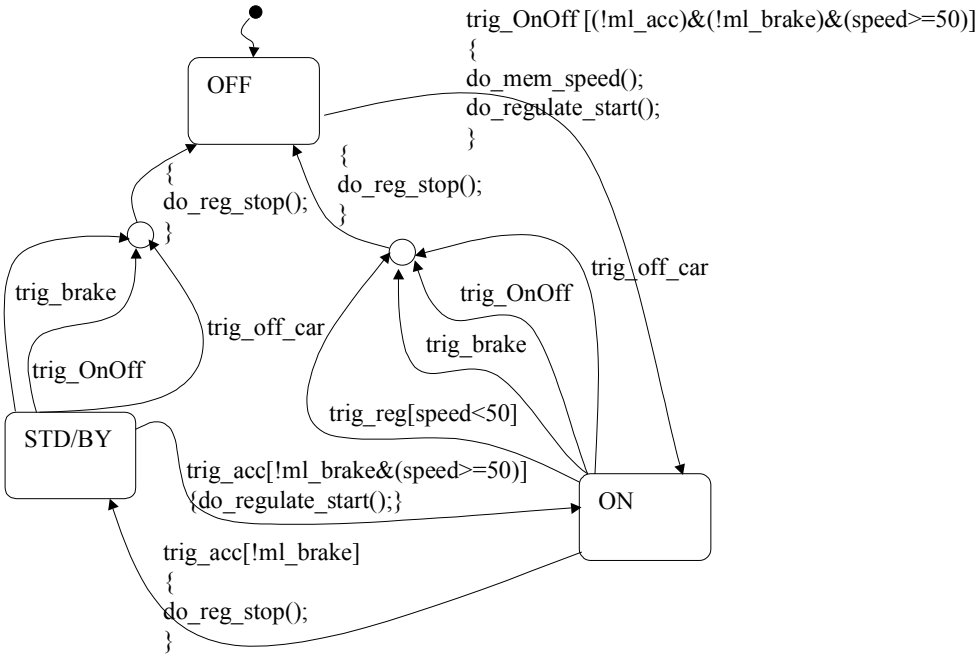


Figure 27 : C-Spec feat_regulate_ctl théorique

Par rapport aux signaux définis au chapitre 5.2.1, deux variables booléennes supplémentaires sont nécessaires à la modélisation (d'où le préfixe **ml_**):

- **ml_brake** : donnée mémorisant l'état de la pédale de frein (pressée ou relâchée),
- **ml_acc** : donnée mémorisant l'état de la pédale d'accélération (pressée ou relâchée).

La machine à état possède 3 états On, Off et Stand/by. L'état par défaut est l'état Off (point noir suivi de la flèche tout en haut du diagramme).

Le passage d'un état à un autre est toujours déclenché par un événement. Considérons la transition partant de l'état Off et reliant l'état On. Celle-ci est déclenchée par l'événement **trig_OnOff**. Si les conditions entre crochets sont vérifiées, à savoir que le frein et l'accélérateur ne sont pas pressés et que la vitesse est supérieure ou égale à 50 km/h, alors :

- la mémorisation de la consigne est effectuée,
- la régulation en vitesse est lancée,
- le nouvel état de la machine à état est On.

Considérons maintenant la transition On vers Off. 4 transitions parallèles liées à 4 évènements peuvent déclencher l'arrêt de la régulation (vitesse inférieure à 50 km/h, moteur coupé, bouton On/Off ou frein pressés).
Tous les passages d'un état à un autre connaissent des transitions similaires.

Modèle Simulink

Le C-Spec a été construit sous Simulink du fait que nous ne disposions pas de Stateflow. Ce diagramme n'est que la transcription en Simulink de la machine à état vue précédemment.

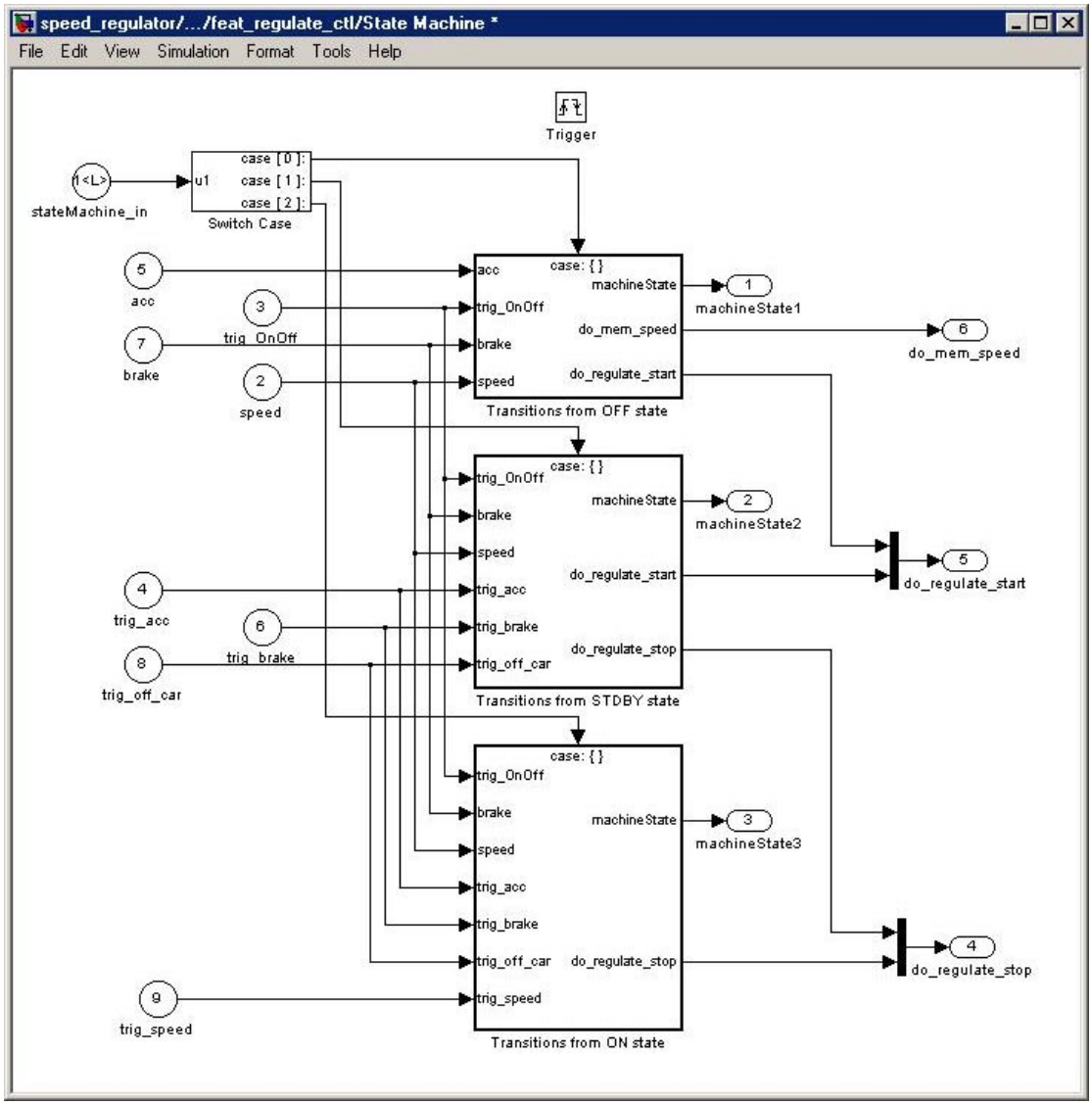


Figure 28 : C-Spec du sous-système feat_regulate_ctl

Pour obtenir les signaux de type trigger, un démultiplexage/filtrage a été effectué à un niveau plus haut. Ces signaux sont nécessaires pour constituer les gardes de type évènement sur les transitions puisque Simulink réveille la machine à état sur tous les évènements reçus.

Afin de traduire la machine à états, chaque état du modèle sous Stateflow s'est vu attribué un nombre sous Simulink : 0, 1 et 2 pour respectivement l'état Off, Stand/by et On. Ainsi, en fonction de l'état courant de la machine à état, le traitement va s'orienter vers le cas 0, 1 ou 2. Le nouvel état **machineState<number>** est alors mis à jour et l'appel de fonction **do_<function call Name>** associée à la transition est envoyé.

Nous allons maintenant nous intéressés au cas 1, lorsque l'état courant de la machine est l'état Stand/by (figure ci-dessous).

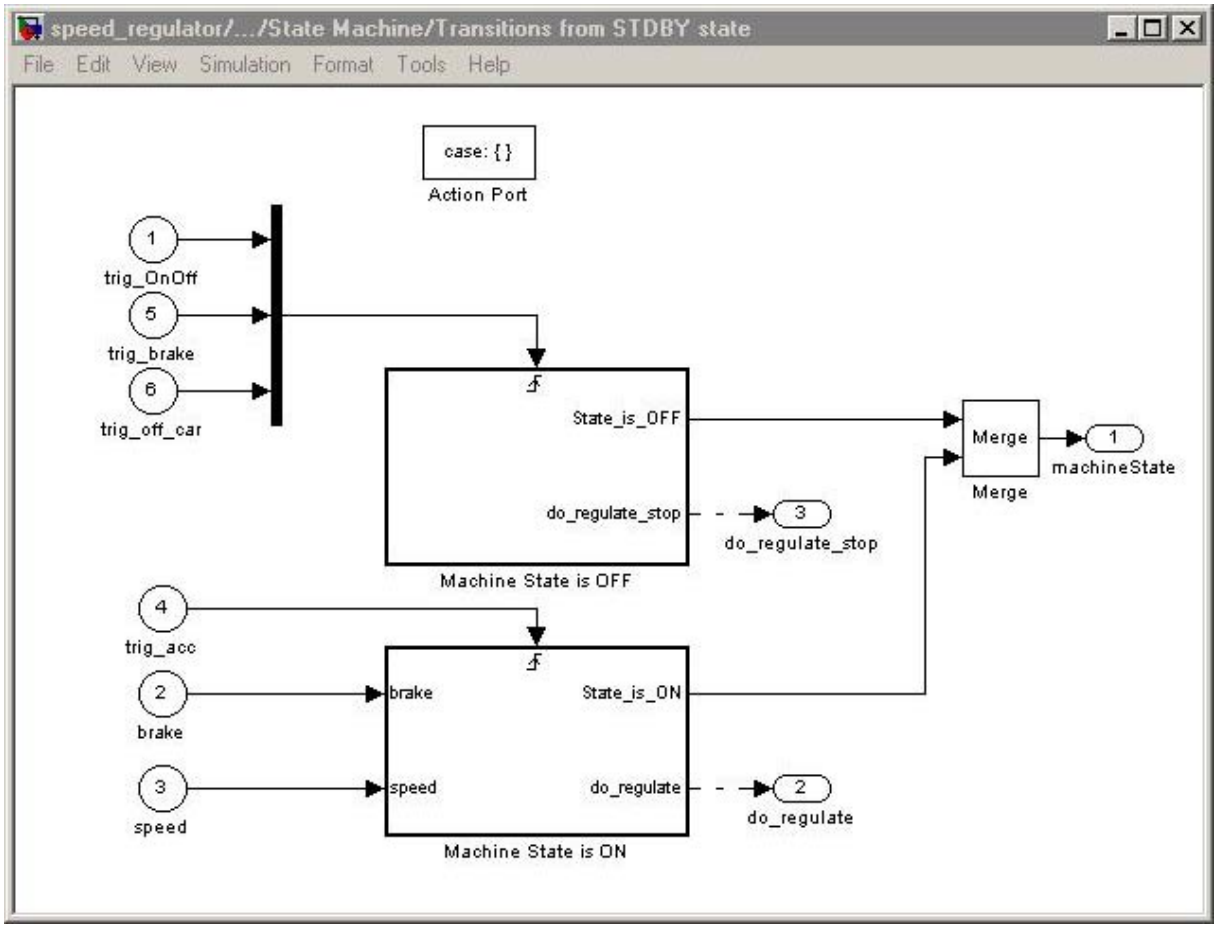


Figure 29 : Transitions à partir de l'état Stand/by

Vous pouvez observer sur ce diagramme qu'il existe deux transitions possibles à partir de l'état Stand/by vers :

- l'état Off si le frein ou le bouton On/Off sont pressés ou encore si le moteur est éteint (évènements **trig_OnOff**, **trig_brake** et **trig_off_car**);
- l'état On si l'accélérateur est relâché (**trig_acc**).

Dans les deux cas, le nouvel état **machineState** de la machine à état est mis à jour et l'appel de fonction associé est envoyé (**do_regulate_stop** ou **do_regulate**).

Pour mémoire, une transition état/état sous Stateflow est de la forme :

event [Condition]
{traitement;}

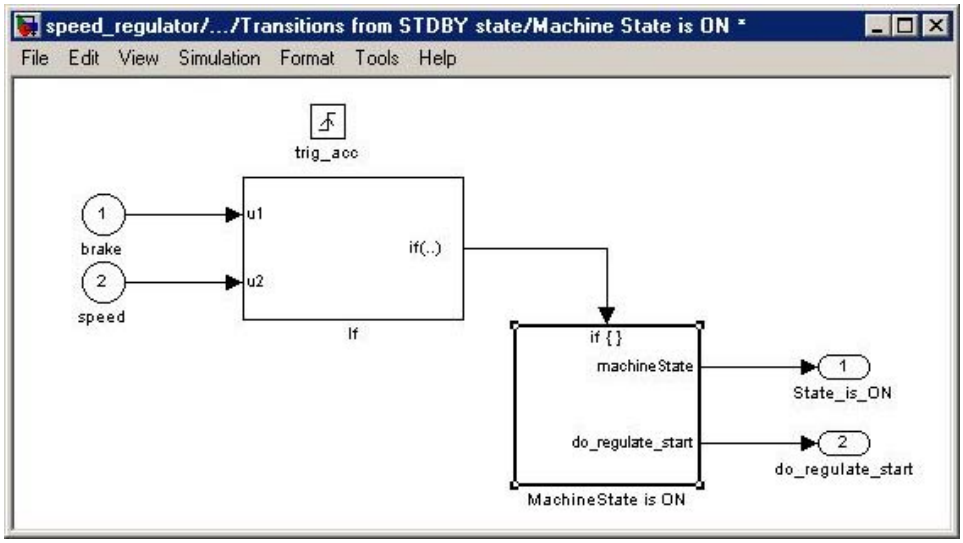


Figure 30 : Conditions sur transition état/état

Le diagramme ci-dessus montre comment ajouter une condition sur une transition état/état associé à un évènement. Une vérification sur les données en entrée **brake** et **speed** est en effet effectuée lorsque l'accélérateur est pressé (réception du signal **trig_acc**).

Par la suite, en fonction de ces entrées, l'état **machineState** de la machine à état est mis à jour et l'appel de fonction **f() do_regulate** est envoyé sur le port **do_regulate_start** (figure 31).

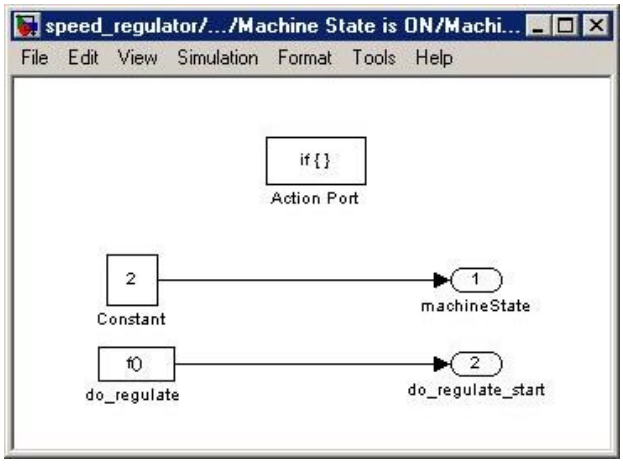


Figure 31 : Mise à jour de la machine à état et appel de fonction

5.2.3.3 Les P-Spec

La figure suivante présente en détail le P-Spec **feat_regulate_start**.

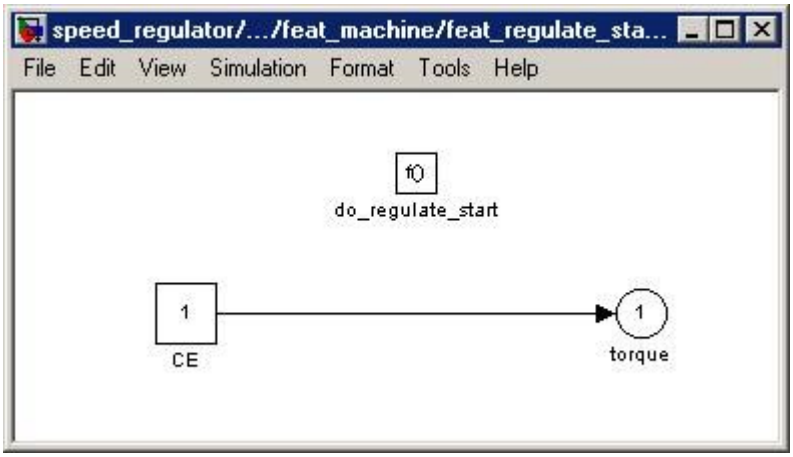


Figure 32 : P-Spec feat_regulate_start

Vous pouvez remarquer que ce sous-système possède comme tous les P-Spec un trigger de type function call **f()** en haut du diagramme.

Lorsqu'un appel de fonction **feat_regulate_start** est reçu, ce process envoie sur sa sortie la valeur 1.

La figure suivante présente en détail le P-Spec **mem_speed**.

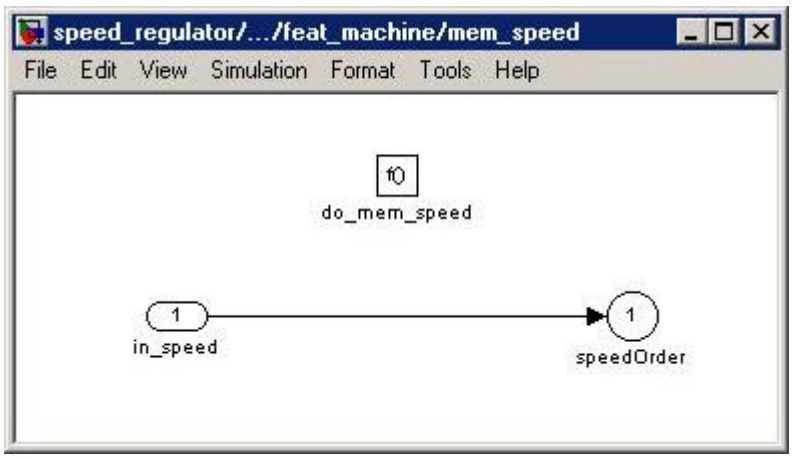


Figure 33 : P-Spec mem_speed

Ce P-Spec envoie sur sa sortie **speedOrder** l'entrée **in_speed** reçue lorsqu'un appel de fonction est reçu.

5.2.4 Sous-système feat_regulate

5.2.4.1 Vue générale

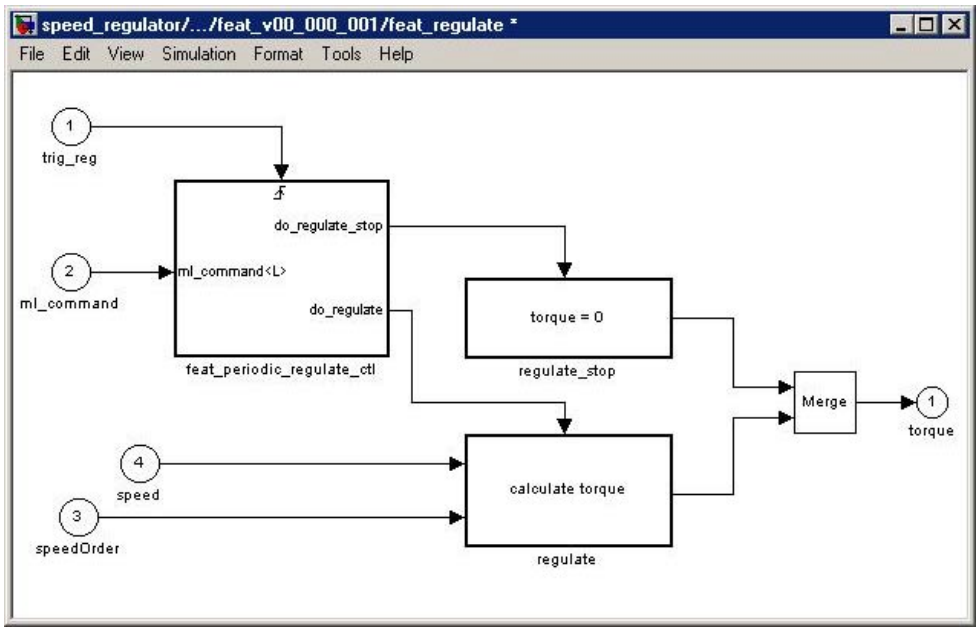


Figure 34 : Détail du sous-système feat_regulate

Ce sous-système est composé du C-Spec **feat_periodic_regulate_ctl** et de 2 P-Spec **regulate_stop** et **regulate** qui assurent respectivement l'arrêt de la régulation (couple nul) et le calcul du couple suivant la loi de commande.

5.2.4.2 C-Spec feat_periodic_regulate_ctl

Modèle Stateflow théorique

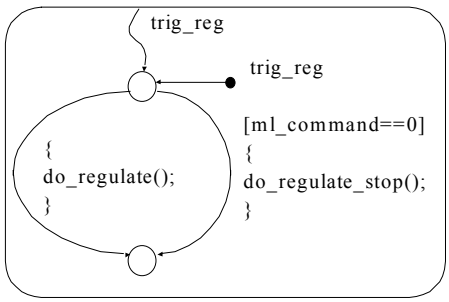


Figure 35 : C-Spec feat_periodic_regulate_ctl théorique

Le C-Spec théorique est décrit ci-dessus. Il possède un état unique. Ce C-Spec scrute à la fréquence de 2 Hz sur réception du signal **trig_reg** si **ml_command** (mis à jour par le premier sous-système feat_machine) est égal à 0. Si c'est le cas, alors l'appel de la fonction **do_regulate_stop** est envoyé, sinon c'est l'appel de la fonction **do_regulate** qui est envoyé.

Modèle Simulink

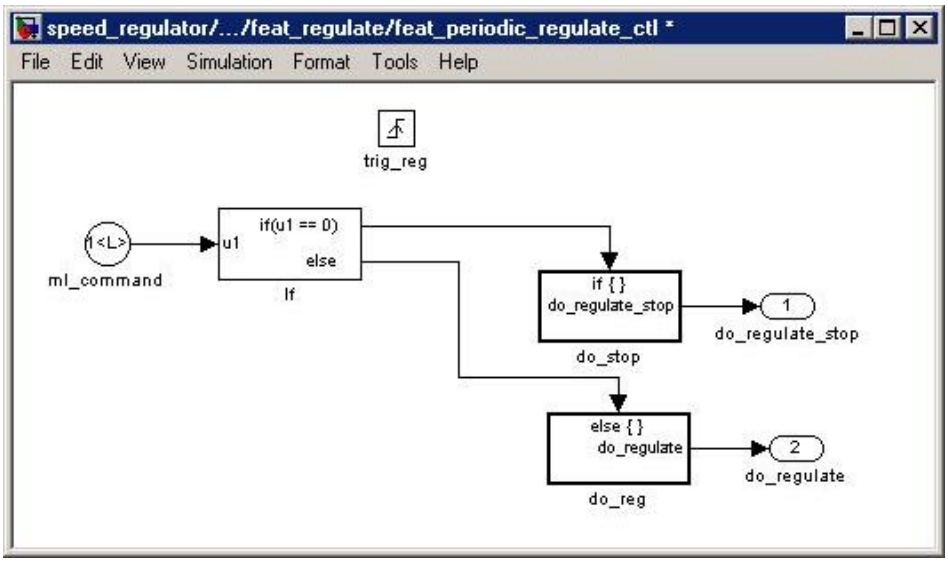


Figure 36 : C-Spec feat_periodic_regulate_ctrl sous Simulink

Le modèle Simulink est très simple et assure trait pour trait les mêmes traitements que le modèle Stateflow.

5.2.4.3 P-Spec do_regulate

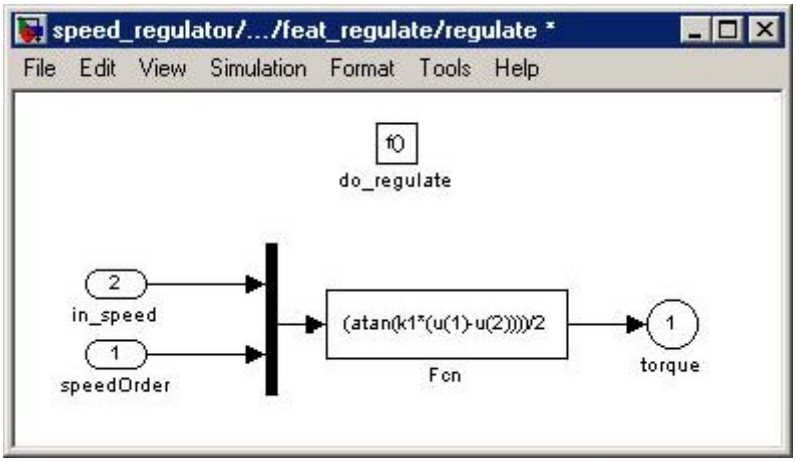


Figure 37 : process do_regulate

Vous pouvez remarquer que ce diagramme possède comme tous les P-Spec un trigger de type function call **f()** en haut de la figure.

Lorsqu'un appel de fonction **do_regulate** est reçu, ce process envoie sur sa sortie le nouveau couple **torque** calculé à partir de la loi de commande en fonction de la consigne de vitesse **speedOrder** et de la vitesse courante **in_speed**.

6. CONCLUSION

Nous avons présenté dans ce document des conventions concernant la modélisation sous Matlab, Simulink et Stateflow. Au fur à mesure que la méthode sera utilisée et appliquée à des systèmes particuliers, des questions seront soulevées. Ce retour sera très intéressant dans la mesure où elle permettra d'améliorer la méthode.

Au-delà de la méthode, des pistes de conversion Simulink/Stateflow ont été explorées. Elles permettent ainsi de représenter, sans une certaine difficulté et une certaine complexité, une machine à états sous Simulink. L'outil Stateflow s'avère donc nécessaire voire vital pour la modélisation des machines à états.

7. GLOSSAIRE

Calibration File Specification	Fichier de configuration des variables globales du modèle.
Context Diagram	Diagramme de niveau le 0 (niveau le plus haut).
C-Spec	Le Control Specification traite principalement des évènements.
Data Dictionnary	Dictionnaire de données permettant d'interfacer les variables Simulink et Stateflow.
Donnée	Il existe deux types de données : les données de flux (continues ou discrètes) et les données de contrôle (évènements). Par abus de langage, nous considérons que les données seront uniquement des données de flux.
Event	Evènement.
Execution Context Diagram	Diagramme de niveau 1.
Function call	Appel de fonction.
Matlab	Outil permettant en autre de simuler des systèmes physiques.
P-Spec	Le Process Specification traite uniquement des donnés.
Signal	Un signal est une donnée.
Simulink	C'est l'outil graphique associé à Matlab.
Stateflow	C'est l'outil associé à Simulink, intégrant les évènements et permettant la simulation de machine à état.
Trigger	Un trigger est un événement.
Workspace	C'est l'espace de travail sous Matlab.