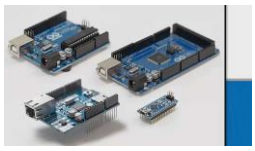
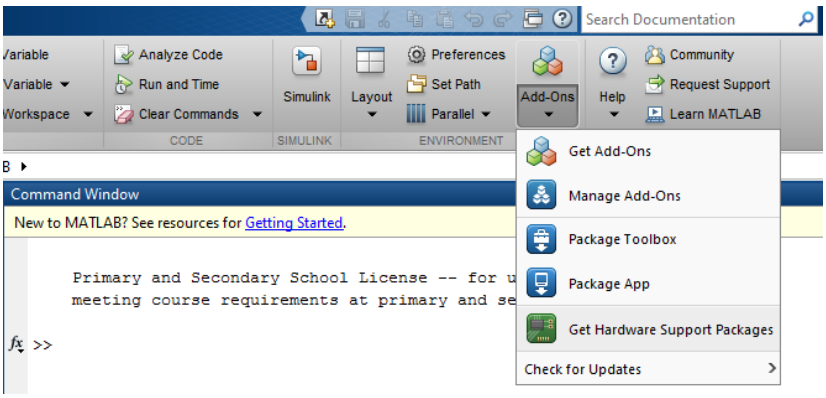
Exemple activité étudiant MEEF S2I "Prototypage d'une solution - Machine à Etats"



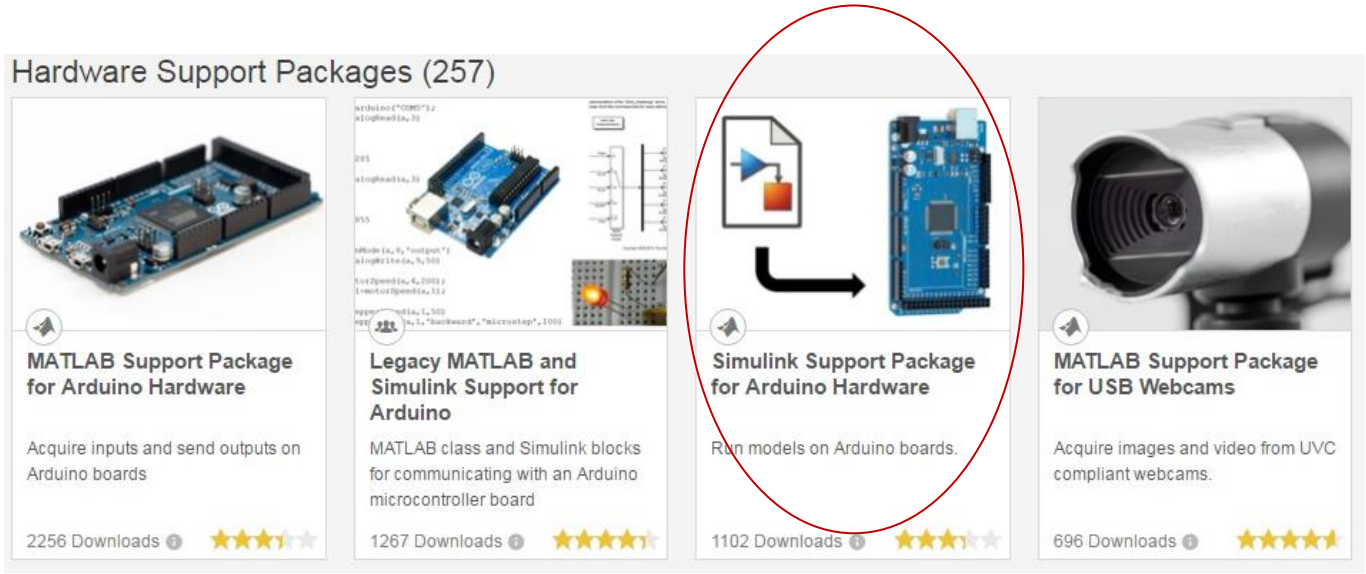
Plateforme Arduino et Simulink

Dans cette activité consacrée au prototypage, nous proposons de **développer une application** prenant comme **supports cibles une carte Arduino MEGA 2560 ou Arduino UNO**. Cette approche est très pertinente, car elle permet d'associer un puissant logiciel de simulation et une carte de prototypage permettant d'envisager des applications complexes.



Pour pouvoir communiquer avec les cartes Arduino, il faut au préalable télécharger et installer « **Simulink Support Package for Arduino Harware** » disponible gratuitement sur le site de Mathworks et accessible depuis le menu **Add-Ons** puis **Get Hardware Support Packages**.

Tous les modules sont disponibles avec la licence MATLAB FOURNIE POUR L'ANFS2I



Ce package vous permet de créer et d'exécuter des modèles Simulink/Stateflow sur des plateformes Arduino. Les modules présents dans cette bibliothèque vous permettront de configurer et d'accéder facilement aux entrées/sorties de la carte. Deux modes d'exécution de l'application sont alors possibles.

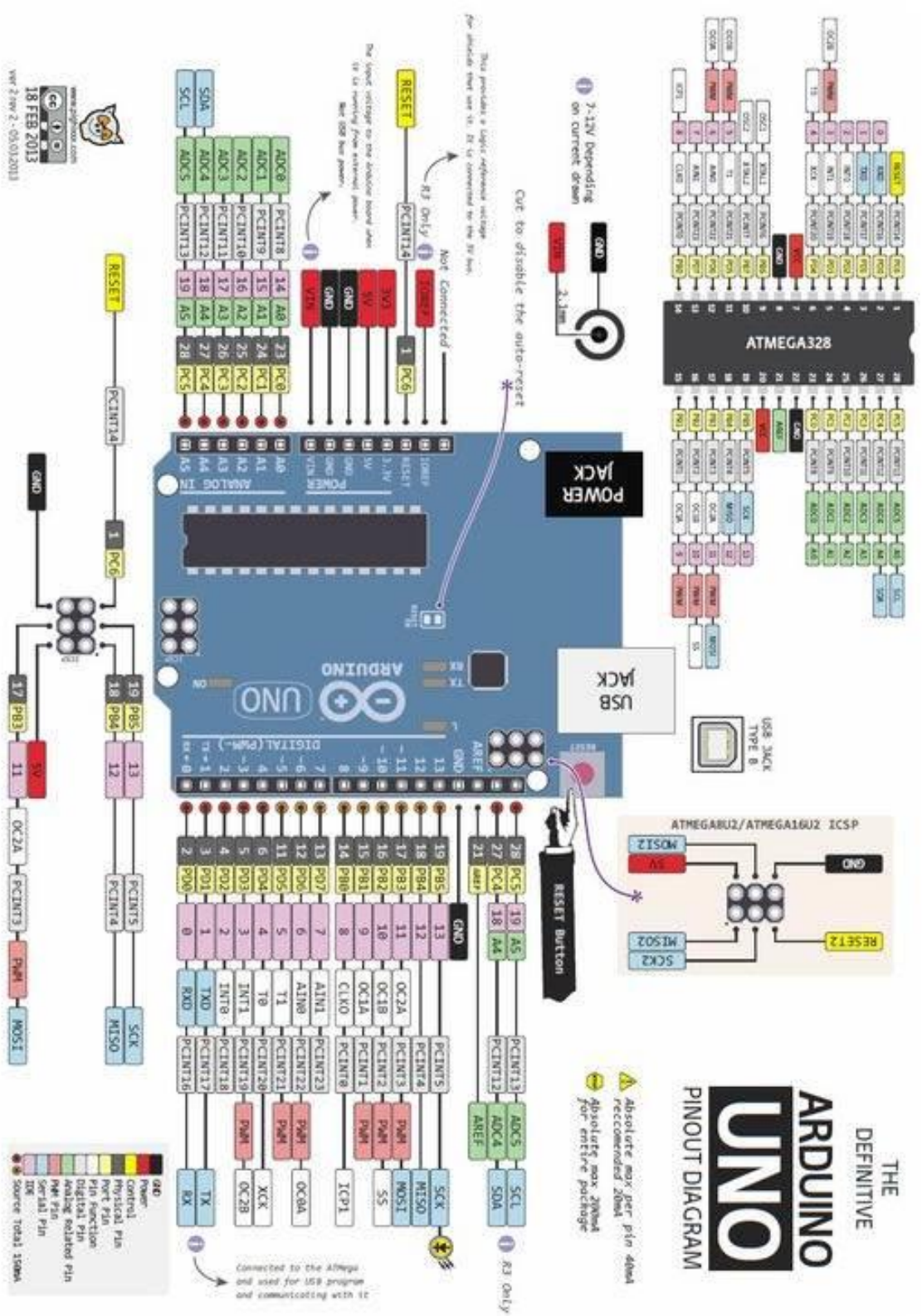
Exécution du modèle en mode externe : surveiller et ajuster de manière interactive les algorithmes développés dans Simulink lorsqu'ils sont exécutés sur la plateforme matérielle.

Exécution du modèle en mode normal *(standalone)* : transfert et exécution de l'algorithme dans la plateforme matérielle.

Il faudra également veiller à ce que le driver USB de la carte Arduino soit installé sur la machine concernée.

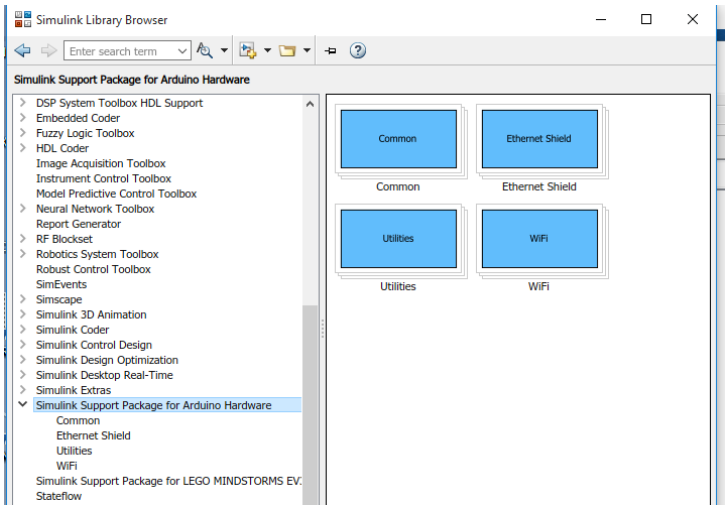
2. Objectifs:

- Etre capable de concevoir un modèle de comportement séquentiel à l'aide de diagrammes d'état dans un environnement Matlab/Simulink.
- Compiler puis implanter le programme obtenu dans une carte Arduino UNO ou MEGA 2560.
- Simuler le comportement avec une « **Partie Opérative »** modélisée.



3. Découverte de la bibliothèque « Simulink Support Package for Arduino »

Dans cette bibliothèque Arduino, quatre modules sont proposés : « Common », « Ethernet Shield », « Utilities » et « Wifi » ainsi que quelques exemples.



Avec ce package, vous pourrez par exemple réaliser les tâches suivantes :

Acquérir des données de capteurs analogiques et numériques depuis votre carte Arduino.

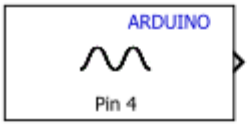
Contrôler d'autres appareils avec des sorties numériques et PWM.

Commander des machines à courant continu, des servomoteurs et des moteurs pas à pas (le Shield Adafruit Motor est également supporté).

Accéder aux appareils et capteurs connectés en I2C ou SPI.

Le module « Common » regroupe l'ensemble des constituants de la carte Arduino qui, associé à un diagramme d'état, permettra la réalisation d'un programme. Une fois compilé, cette application pourra être chargée dans la cible.

Mesurer et convertir sur 10 bits la tension analogique présente sur une broche.



Analog Input

Analog Output

Générer une tension analogique sur une broche de sortie.

Régler la vitesse d'un servo-moteur entre -90 et 90. Une valeur de 0 n'entraine pas de rotation.

Continuous Servo Write

Digital Input

Lecture du niveau logique présent sur une broche numérique.

Ecriture d'une valeur logique sur une broche numérique.

Digital Output

I2C Read

Lecture de donnée envoyée par un composant I2C esclave ou lecture d'un registre spécifique.

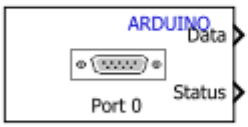
Ecriture de donnée vers un composant I2C esclave ou dans un registre spécifique.

I2C Write

PWM

Génère un signal PWM sur une broche de sortie spécifique. Valeur comprise en 0 et 255.

Lecture de donnée présente sur le port série.



Serial Receive

Serial Transmit

Ecriture de donnée sur le port série.

Ecriture/lecture de donnée depuis un composant SPI.

SPI WriteRead

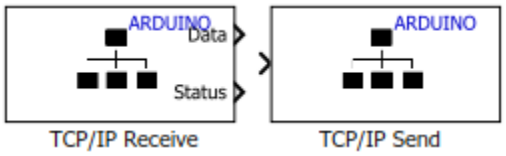
Standard Servo Read

Lecture de la position d'un servo-moteur en degrés.

Commande la position d'un servo-moteur standard entre 0 et 180.

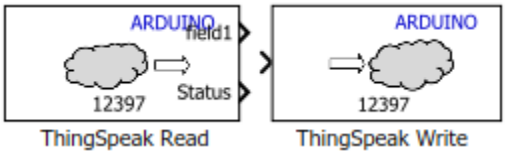
Le module « Ethernet Shield », regroupe l'ensemble des constituants concernant la réalisation d'application « câblée » en utilisant différents protocoles.

Configurer la pile TCP/IP pour recevoir les données d'un client sur un port « serveur » Arduino.



Configurer la pile TCP/IP pour envoyer les données depuis un serveur « Arduino » vers un port.

Lit les données enregistrées dans un canal ThingSpeak si la carte Arduino est connectée à internet.



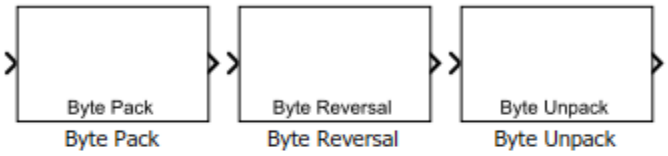
Ecrit les données dans un canal ThingSpeak si la carte Arduino est connectée à internet.

Configurer la pile TCP/IP pour recevoir les messages UDP.

Configurer la pile TCP/IP pour envoyer les messages UDP vers une interface identifiée par une IP.

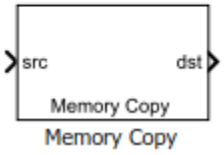
Le module « Utilities » regroupe l'ensemble des constituants qui permet la conversion entre différents formats de donnée.

Convertir en paquet d'octet les données présentes en entrée du bloc.



Extraire d'un paquet d'octet les données présentes.

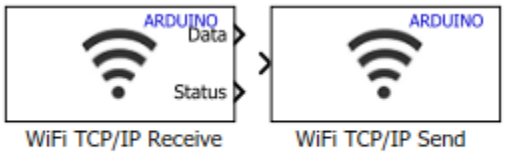
Lecture et écriture de donnée vers une mémoire cible.



Conversion de donnée vers le format big-endian.

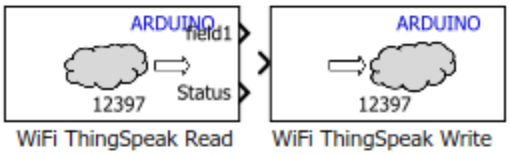
Le module « Wifi » regroupe l'ensemble des constituants concernant la réalisation d'application « sans fil »

Configurer la pile TCP/IP pour recevoir les données d'un client sur un port « serveur » Arduino.



Configurer la pile TCP/IP pour envoyer les données depuis un serveur « Arduino » vers un port.

Lit les données enregistrées dans un canal ThingSpeak si la carte Arduino est connectée à internet.



Ecrit les données dans un canal ThingSpeak si la carte Arduino est connectée à internet.

Configurer la pile TCP/IP pour recevoir les messages UDP.

Configurer la pile TCP/IP pour envoyer les messages UDP vers une interface identifiée par une

Utile et Pratique !

Le tableau de la page 5 liste les numéros des broches des cartes Arduino que l'on peut associer avec les modules précédents.

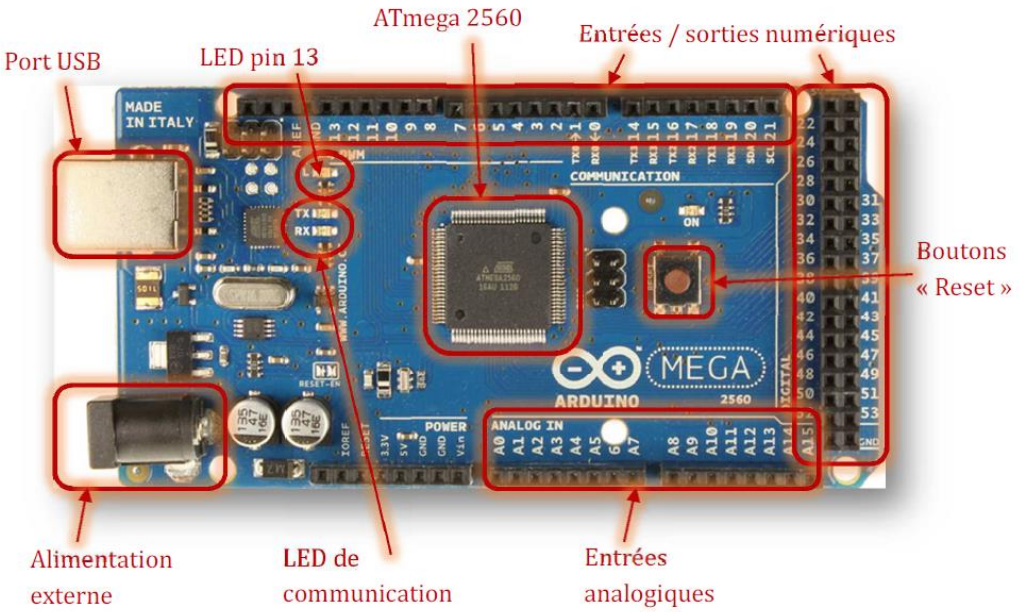
	Blocks	Target Hardware														
		Uno	Nano 3.0	Fio	Mini	Pro	Leonardo	Micro	Esplora	Mega 2560/ADK	Due	Robot Control Board	Robot Motor Board	Lilypad USB	Yun	MKR1000
User Selectable Pins	Digital Input	0-13	0-13	0-13	0-13	0-13	0-19	0-19	0-19	0-53	0-53	0-3,5-17,19-22	0-19	2,3,9,10,11,13	0-19	0-14
	Digital Output	0-13	0-13	0-13	0-13	0-13	0-19	0-19	0-19	0-53	0-53	0-3,5-17,19-22	0-19	2,3,9,10,11,13	0-19	0-14
	Analog Input	0-5	0-7	0-7	0-7	0-5	0-11	0-11	0-11	0-15	0-11	0,1,2,3,4,5,6,7,11	0,1,2,3,4,5,6,11	2-5	0-11	0-6
	Analog Output	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	DAC0, DAC1	N/A	N/A	N/A	N/A	DAC0
	PWM	3,5,6,9,10,11	3,5,6,9,10,11	3,5,6,9,10,11	3,5,6,9,10,11	3,5,6,9,10,11	3,5,6,9,10,11,13	3,5,6,9,10,11,13	3,5,6,9,10,11,13	2-13, 44-46	2-13	N/A	N/A	3,9,10,11	3,5,6,9,10,11,13	0-8,10,18,19
	Standard Servo Read	0-13	0-13	0-13	0-13	0-13	0-19	0-19	0-19	0-53	0-53	19,20,21,22	4,12,18,19	2,3,9,10,11	0-19	0-14
	Standard Servo Write	0-13	0-13	0-13	0-13	0-13	0-19	0-19	0-19	0-53	0-53	19,20,21,22	4,12,18,19	2,3,9,10,11	0-19	0-14
	Continuous Servo Write	0-13	0-13	0-13	0-13	0-13	0-19	0-19	0-19	0-53	0-53	19,20,21,22	4,12,18,19	2,3,9,10,11	0-19	0-14
	SPI Slave Select (SS)	0-10	N/A	N/A	N/A	N/A	0-19	0-19	0-19	0-49, 53	4,10,52	0-19	0-19	N/A	0-19	0-7,11-14
	SPI MOSI	11/ ICSP-4	N/A	N/A	N/A	N/A	ICSP-4	ICSP-4	ICSP-4	51/ ICSP-4	ICSP-4	ICSP-4	ICSP-4	N/A	ICSP-4	8
	SPI MISO	12/ ICSP-1	N/A	N/A	N/A	N/A	ICSP-1	ICSP-1	ICSP-1	50/ ICSP-1	ICSP-1	ICSP-1	ICSP-1	N/A	ICSP-1	10
	SPI SCK	13/ ICSP-3	N/A	N/A	N/A	N/A	ICSP-3	ICSP-3	ICSP-3	52/ ICSP-3	ICSP-3	ICSP-3	ICSP-3	N/A	ICSP-3	9
	I2C SDA	A4	D4	D4	A4	4	2/ SDA	2	N/A	20	20	D2	D2	2	2/ SDA	11
	I2C SCL	A5	D5	D5	A5	5	3/ SCL	3	N/A	21	21	D3	D3	3	3/ SCL	12
Fixed Pins	Serial Receive	Port 0: pin 0 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 0 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 0 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 0 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 0 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 0 Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 0 Port 2: N/A Port 3: N/A	Port 0: No pins Port 1: pin 0 Port 2: N/A Port 3: N/A	Port 0: pin 0 Port 1: pin 19 Port 2: pin 17 Port 3: pin 15	Port 0: pin 0 Port 1: pin 19 Port 2: Pin 17 Port 3: Pin 15	Port 0: No pins Port 1: pin 0 Port 2: N/A Port 3: N/A	Port 0: No pins Port 1: pin 0 Port 2: N/A Port 3: N/A	Port 0: No pins Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 0 Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 13
	Serial Transmit	Port 0: pin 1 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 1 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 1 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 1 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: pin 1 Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 1 Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 1 Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 1 Port 2: N/A Port 3: N/A	Port 0: pin 1 Port 1: pin 18 Port 2: pin 16 Port 3: pin 14	Port 0: pin 1 Port 1: pin 18 Port 2: pin 16 Port 3: pin 14	Port 0: No pins Port 1: pin 1 Port 2: N/A Port 3: N/A	Port 0: No pins Port 1: pin 1 Port 2: N/A Port 3: N/A	Port 0: No pins Port 1: N/A Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 1 Port 2: N/A Port 3: N/A	Port 0: No Pins Port 1: pin 14
	TCP/IP Receive	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	TCP/IP Send	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	UDP Send	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	UDP Receive	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	ThingSpeak Write	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	WiFi TCP/IP Receive	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	WiFi TCP/IP Send	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	WiFi UDP Send	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A
	WiFi UDP Receive	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,11,12,13	10,50,51,52,53	N/A	N/A	N/A	N/A	10,11,12,13	N/A

4. Les cartes Arduino UNO et MEGA 2560

L'utilisation de cette plateforme de prototypage est intéressante, d'une part, pour son faible coût et, d'autre part, pour un fonctionnement en mode externe. Ainsi, les « **Switches** » saisis dans le modèle Simulink permettent un contrôle **en direct** de la cible suite à leurs commutations.

Quelques caractéristiques de la carte MEGA 2560 :

Microcontrôleur	ATmega2560
Tension de fonctionnement	5 V
Tension d'alimentation (recommandée)	7- 1 2 V
Tension d'alimentation (limites)	6 - 20V
Nombre d'E/S	54 (dont 14 pouvant générer des signaux PWM)
Nb ports "Analogique/Numérique"	16
Courant max. par E/S	40 mA
Courant pour broches 3.3 V	50 mA
Mémoire Flash	256 KB (ATmega328) dont 8 KB utilisé par le bootloader
SRAM	8 KB (ATmega328)
EEPROM	4 KB (ATmega328)
Vitesse horloge	16 MHz



5. Cas d'étude : le portail SET

Le support choisi est l'ouvre-portail SET, présent dans la quasi-totalité des salles de SII. Nous allons modéliser, très simplement pour une première approche, puis implanter dans un système microprogrammé la partie « logicielle » de pilotage de l'ouvre-portail. Nous nous intéresserons plus spécifiquement à l'ouverture/fermeture des vantaux gauche et droite ainsi qu'au clignotement du feu jaune.

Pour le feu clignotant, il s'allume 5 secondes avant le mouvement du premier vantail et s'éteint 3 secondes après la fin du mouvement du dernier vantail.

On souhaite, pour cette activité, décomposer notre travail en 4 parties :

1- Piloter, depuis Simulink, l'allumage d'une led Jaune. Tester le mode de fonctionnement externe.

2- Programmer le clignotement du « Feu Jaune » puis implanter cette solution dans une carte Arduino et tester le mode de fonctionnement normal.

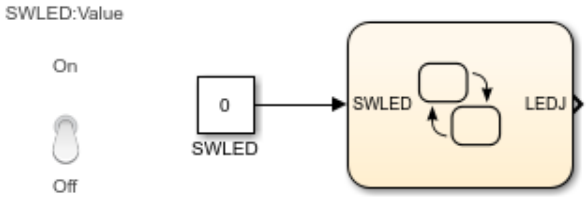
3- Déclencher le clignotement du feu suite à un appui sur un bouton poussoir « virtuel ».

4- Modéliser le cycle d'ouverture/fermeture des vantaux en y associant la gestion du feu jaune.

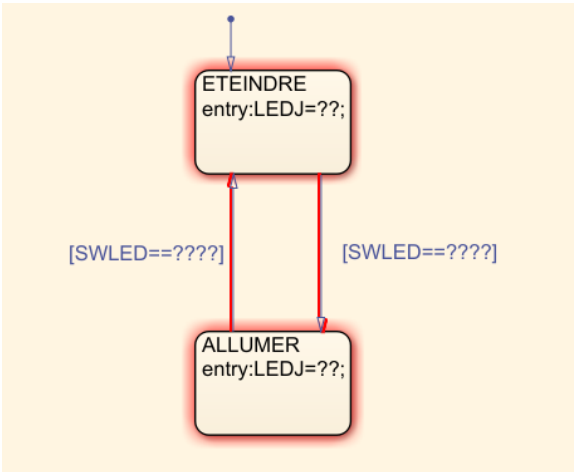
A/ Gestion manuelle du « Feu Jaune ».

Commençons par un montage simple qui consiste à allumer et éteindre la LED Jaune d'une carte Arduino au moyen d'un « Switch », un interrupteur virtuel.

- **Ouvrir** le fichier "LEDJ.slx" depuis la fenêtre « Current Folder » de Matlab. Vous devez obtenir sur votre écran une fenêtre Simulink conforme à l'image ci-dessous.



- **En double cliquant** sur le "Chart", vous ouvrez le diagramme d'état.



Dans le "Chart" nous avons 2 états : "**ETEINDRE**" et "**ALLUMER**". Dans notre cas, il faut que dans l'état « ALLUMER », la led jaune s'éclaire et dans l'état « ETEINDRE », la led jaune s'éteint.

On souhaite, pour débuter, programmer 2 actions :

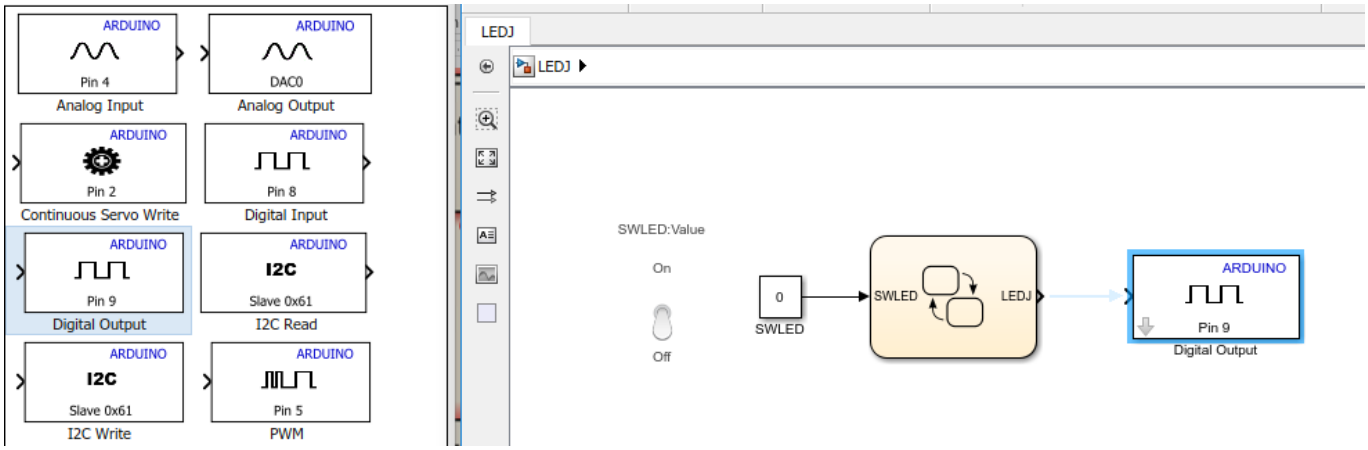
1- Eteindre la led jaune si le « Switch » est inactif *(Position OFF).*

2- Allumer la led jaune si le « Switch » est activé *(Position ON).*

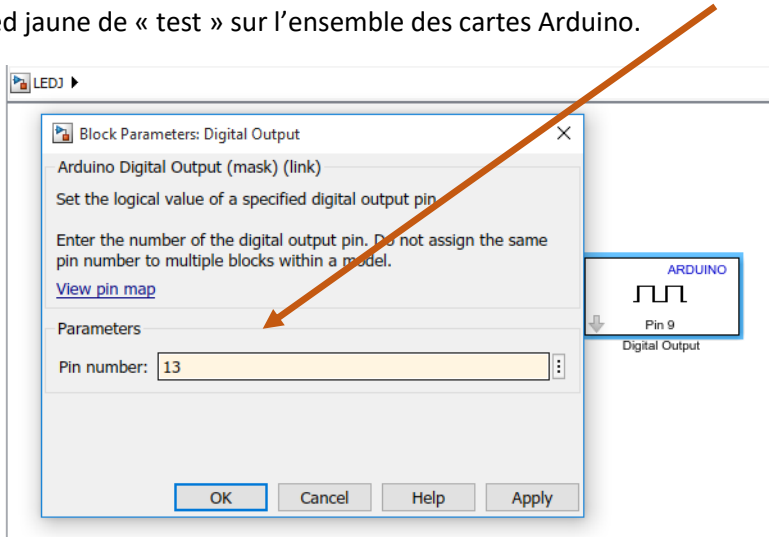
- ❑ **Renseigner** les actions associées aux états. Il s'agit de **remplacer** les points d'interrogation par le niveau logique souhaité *(1 ou 0).*
- ❑ Dans le "Chart", **inscrire les conditions** que l'on souhaite pour la transition. Il s'agit de **remplacer** les points d'interrogation par le niveau logique attendu *(1 ou 0).*

Une sortie « LEDJ » a été configurée dans le bloc « Chart ». Nous allons associer cette sortie avec une broche de la carte Arduino.

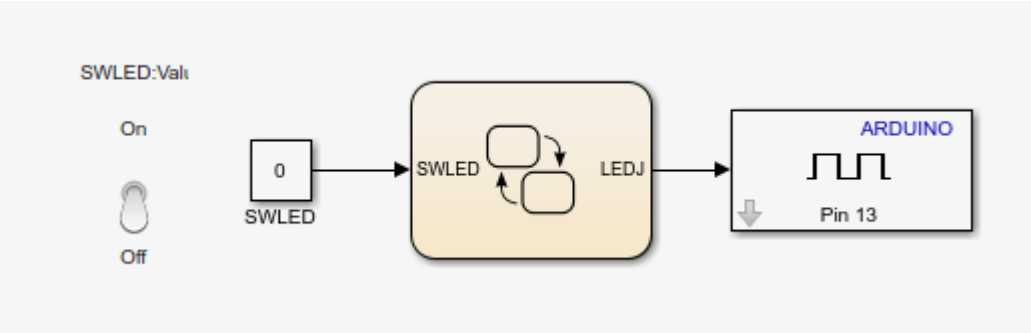
- ❑ **Glisser-déposer** depuis la bibliothèque *Simulink Support Package for Arduino* une sortie numérique *(Digital Output).*



- ❑ **Relier** la sortie LEDJ du bloc Stateflow avec la broche du bloc Arduino « Digital Output ».
- ❑ **Modifier** le numéro de la broche de sortie utilisée. On **choisira la broche n°13**, en effet celle-ci est équipée d'une led jaune de « test » sur l'ensemble des cartes Arduino.

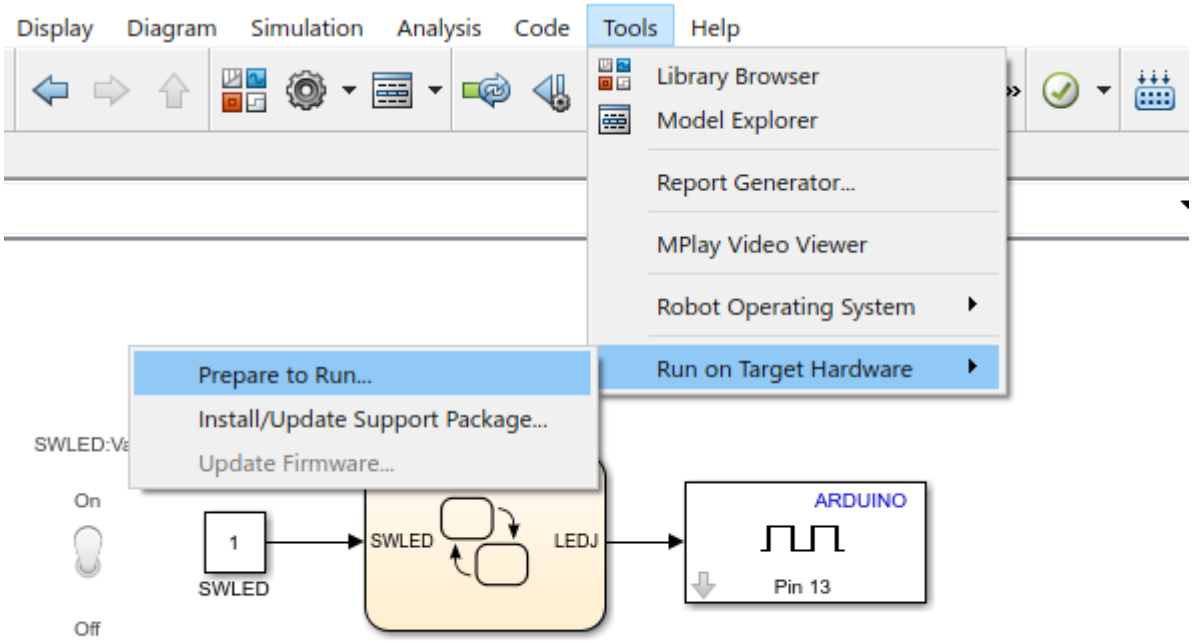


Vous devez obtenir le schéma ci-après :

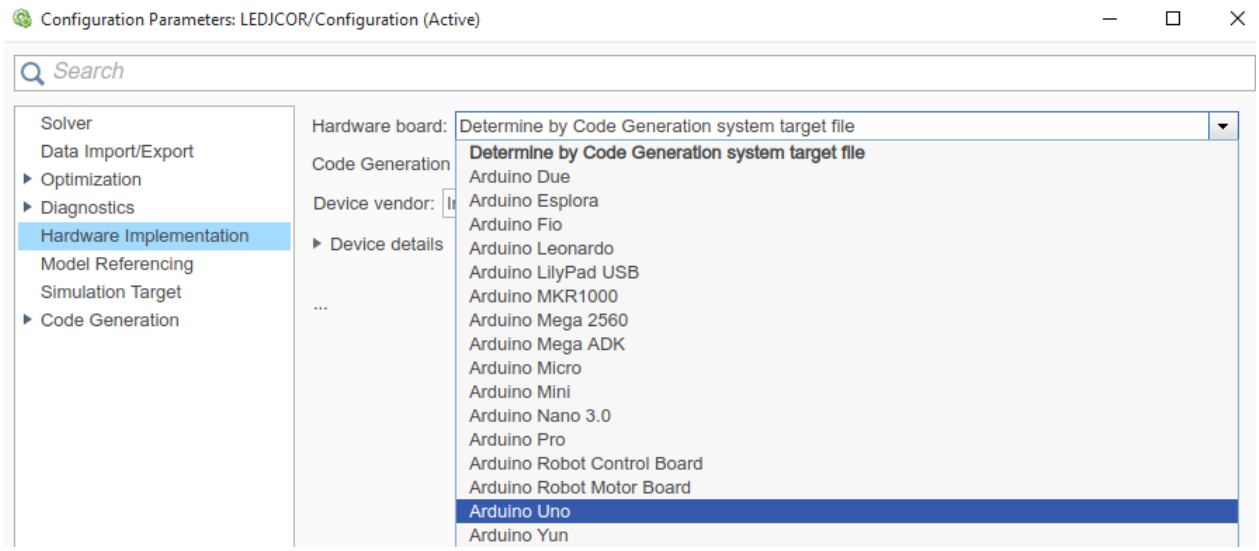


Il faut à présent implanter cette solution dans la cible. Pour cela il faut mettre en œuvre le processus de déploiement pour embarquer le modèle sur le processeur de la carte Arduino.

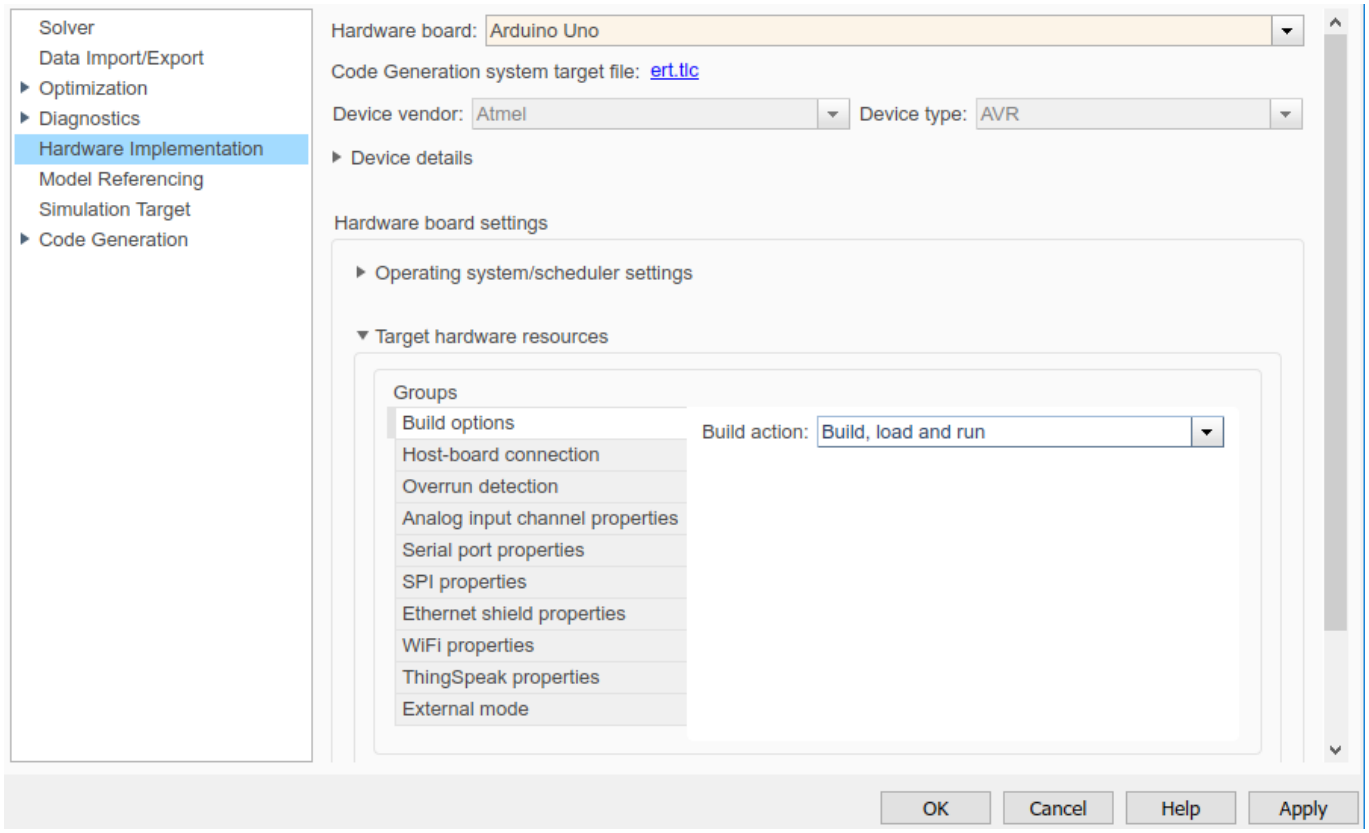
- ❑ Depuis le menu Tools, **sélectionner** Run on Target Harware **puis** Prepare to Run.



- ❑ **Sélectionner** la plateforme de prototypage concernée : Arduino UNO ou MEGA 2560.

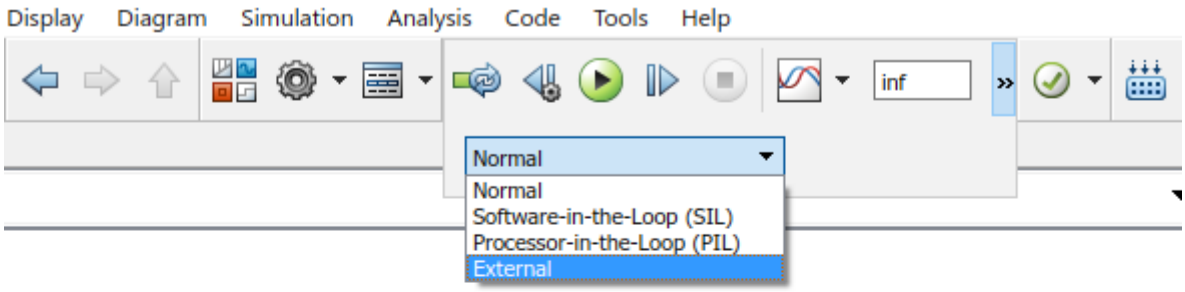


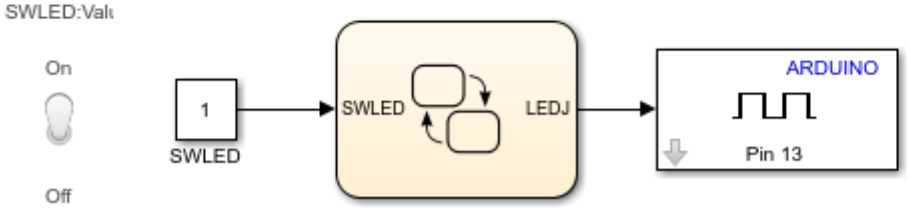
- ❑ Avant de lancer la compilation du programme, il faut **vérifier** les paramètres de configuration de la cible et notamment **l'affectation automatique du port série** de programmation dans la zone Host-board connection. **Cliquer** sur Apply **puis** OK pour valider.



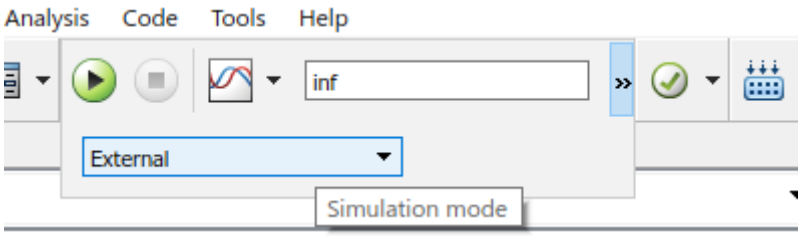
Le temps de simulation doit être infini pour profiter du mode « externe » en continu. Le mode externe (« External ») est sélectionné dans le menu déroulant *Simulation mode*, à droite du temps de simulation.

- ❑ **Configurer**, en tenant compte des remarques précédentes, les zones « Simulation stop time » et « Simulation mode ».

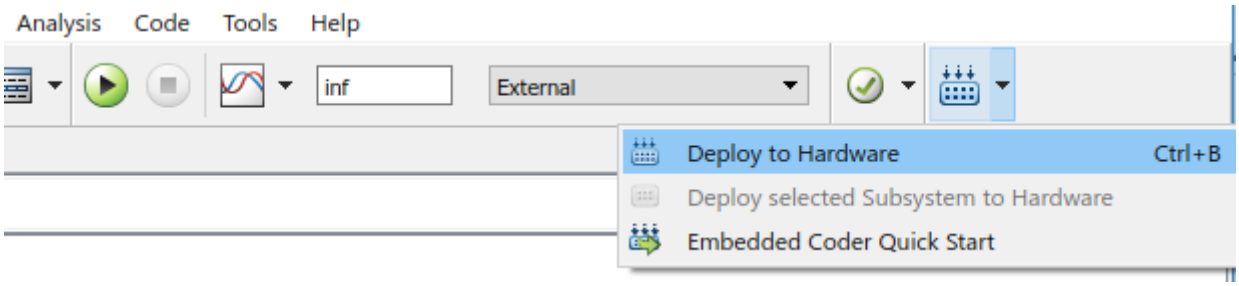




- Vous devez obtenir sur votre écran une fenêtre Simulink conforme à l'image ci-dessous.

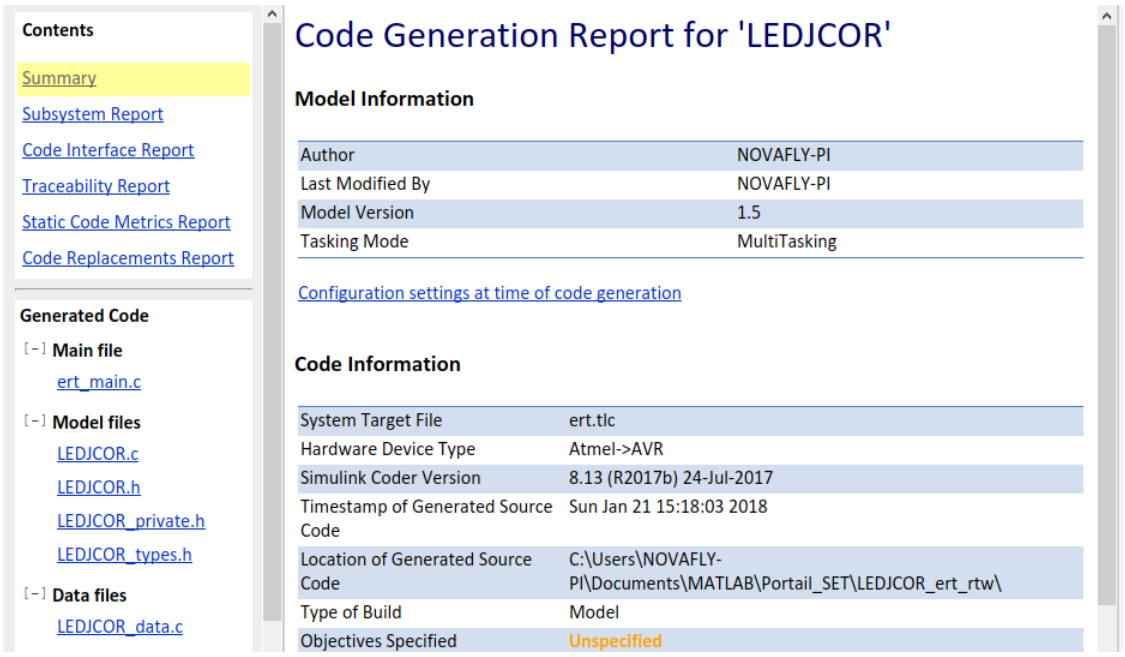


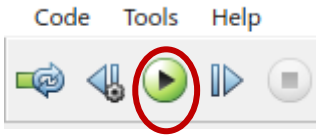
- Compiler et générer le code en **cliquant sur l'icône** « Deploy to hardware » du bandeau supérieur ou en faisant **Ctrl+B**.



Au moment du téléchargement, les leds Tx et Rx de la carte Arduino clignotent.

Le rapport de compilation apparait quelques secondes plus tard…ou minutes suivant la puissance de calcul de votre machine !





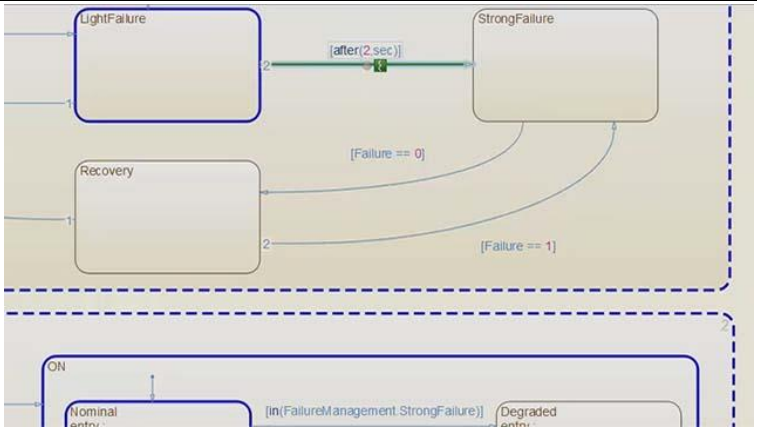
- Une fois le programme chargé, **cliquer** sur le bouton RUN. Vous allez découvrir la capacité de Simulink pour « monitorer » et régler les paramètres de façon interactive avec la carte de prototypage.

- La simulation se déroule sur une durée infinie. **Vérifier en agissant sur le switch**, si le résultat obtenu correspond au comportement attendu.

Vous pouvez visualiser, de **manière interactive**, les états actifs.

Pour cela, il suffit de faire un double clic dans le « chart » afin d'observer le comportement du diagramme d'état.

Les **états actifs apparaissent en bleu**. Ils changent en fonction du déroulement du programme.



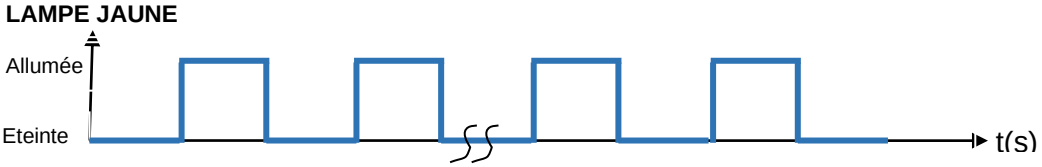
Vous venez de découvrir le **mode de simulation « externe »**. Ce mode de fonctionnement vous permet de communiquer de façon interactive avec votre carte de prototypage. On peut considérer, dans ce cas, que la carte Arduino est utilisée comme une carte d'entrée/sortie pour Matlab/Simulink.

B/ Clignotement continu du feu jaune.

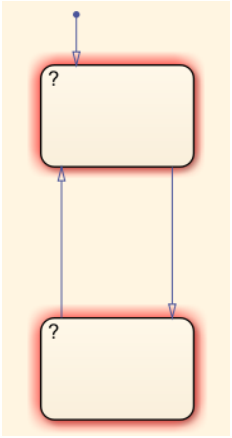
La deuxième application consiste à faire clignoter, en permanence, la led jaune présente sur la carte Arduino. On modélise ainsi, simplement, le clignotement de l'avertisseur lumineux de l'ouvre-portail.

Lorsque, avec la télécommande, on demande une ouverture du portail, on remarque que le feu se met à clignoter 5 secondes avant le début du mouvement. Le clignotement stoppe 3 secondes après le mouvement du dernier vantail.

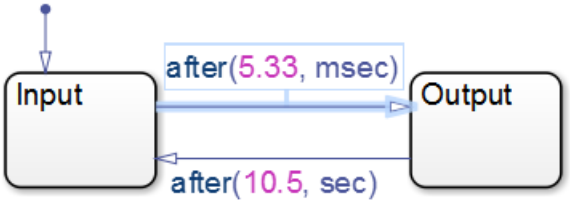
Le rythme du signal lumineux, périodique *(T=1s et t_{on}=0.5s)*, est conforme au chronogramme suivant :



- **Ouvrir** le fichier "LEDJCLI.slx" depuis la fenêtre « Current Folder » de Matlab. Vous devez obtenir sur votre écran un diagramme d'état conforme à l'image ci-dessous.



- **Renseigner** le nom des états ainsi que les actions associées aux états. On utilisera une variable LEDJ pour contenir l'état logique de la led jaune.
- Dans le "Chart", **inscrire les conditions** que l'on souhaite pour les transitions. Vous pouvez vous aider de la représentation ci-contre.



Une sortie « LEDJ » doit être configurée dans le bloc « Chart ». Nous allons associer cette sortie avec la broche 13 de la carte Arduino. Pour cela vous devez :

- Par un **clic droit** n'importe où dans le diagramme et en **choisissant** "Explore", vous accédez à la liste des variables qui seront utilisées dans le "Chart". **Déclarer** « LEDJ » comme une sortie.
- **Glisser-déposer**" depuis la bibliothèque *Simulink Support Package for Arduino* une sortie numérique *(Digital Output)* puis **relier** la sortie LEDJ avec le bloc Arduino « Digital Output ».

Il faut implanter cette solution dans la cible. Pour cela, il faut mettre en œuvre le processus de déploiement pour embarquer le modèle sur le processeur de la carte Arduino.

- Depuis le menu Tools, **sélectionner** Run on Target Harware **puis** Prepare to Run.

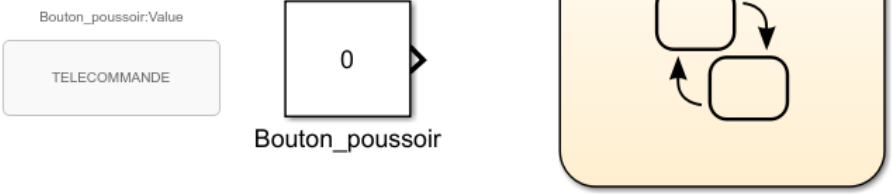
Le mode normal est sélectionné dans le menu déroulant *Simulation mode*, à droite du temps de simulation. La valeur indiquée dans la durée de simulation n'est pas prise en compte pour le déploiement de l'application, en mode normal.

- **Sélectionner** la plateforme de prototypage concernée : Arduino UNO ou MEGA 2560.
- Compiler et générer le code en **cliquant sur l'icône** « Deploy to hardware » du bandeau supérieur ou en faisant Ctrl+B.
- Une fois le transfert terminé, l'application s'exécute automatiquement. **Vérifier** si le résultat obtenu est conforme à nos attentes.

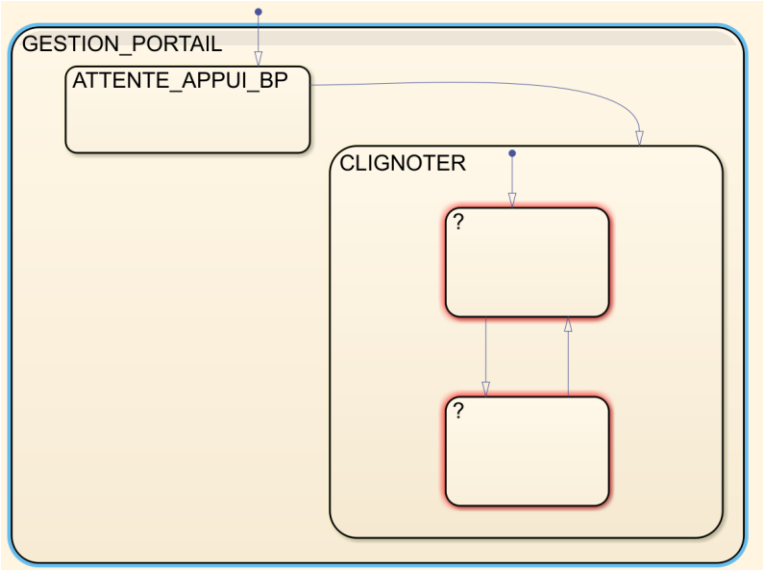
A/ Déclencher le clignotement du feu jaune suite à un appui sur un poussoir virtuel.

Cette troisième application consiste à faire clignoter en permanence la led jaune présente sur la carte Arduino, suite à un appui sur un bouton poussoir « virtuel ». L'objectif est ici de modéliser le déclenchement de l'avertisseur lumineux de l'ouvre-portail, suite à une action sur la télécommande.

- **Ouvrir** le fichier "LEDJBP.slx" depuis la fenêtre « Current Folder » de Matlab. Vous devez obtenir sur votre écran une fenêtre Simulink conforme à l'image ci-dessous.



- Le diagramme d'état « Chart » proposé par défaut est incomplet *(voir illustration ci–dessous).* A vous de **renseigner** le nom des états ainsi que les actions associées aux états. **Inscrire les conditions** que l'on souhaite pour les transitions. **Ne pas oublier d'enregistrer votre travail !**



- Par un **clic droit** n'importe où dans le diagramme et en **choisissant** "Explore", vous accédez à la liste des variables qui seront utilisées dans le "Chart". **Déclarer** « LEDJ » comme une sortie et « BP » comme une entrée.

- **Glisser-déposer**" depuis la bibliothèque *Simulink Support Package for Arduino* une sortie numérique *(Digital Output)* puis **relier** la sortie LEDJ avec le bloc Arduino « Digital Output » **et** l'entrée « BP » avec la constante « Bouton_poussoir ».

- Mettre en œuvre le processus de déploiement pour embarquer le modèle sur le processeur de la carte Arduino. Depuis le menu Tools, **sélectionner** Run on Target Harware **puis** Prepare to Run.

- **Choisir** le mode « externe » et un temps de simulation infini. Compiler et générer le code en **cliquant sur l'icône** « Deploy to hardware » du bandeau supérieur ou en faisant Ctrl+B.

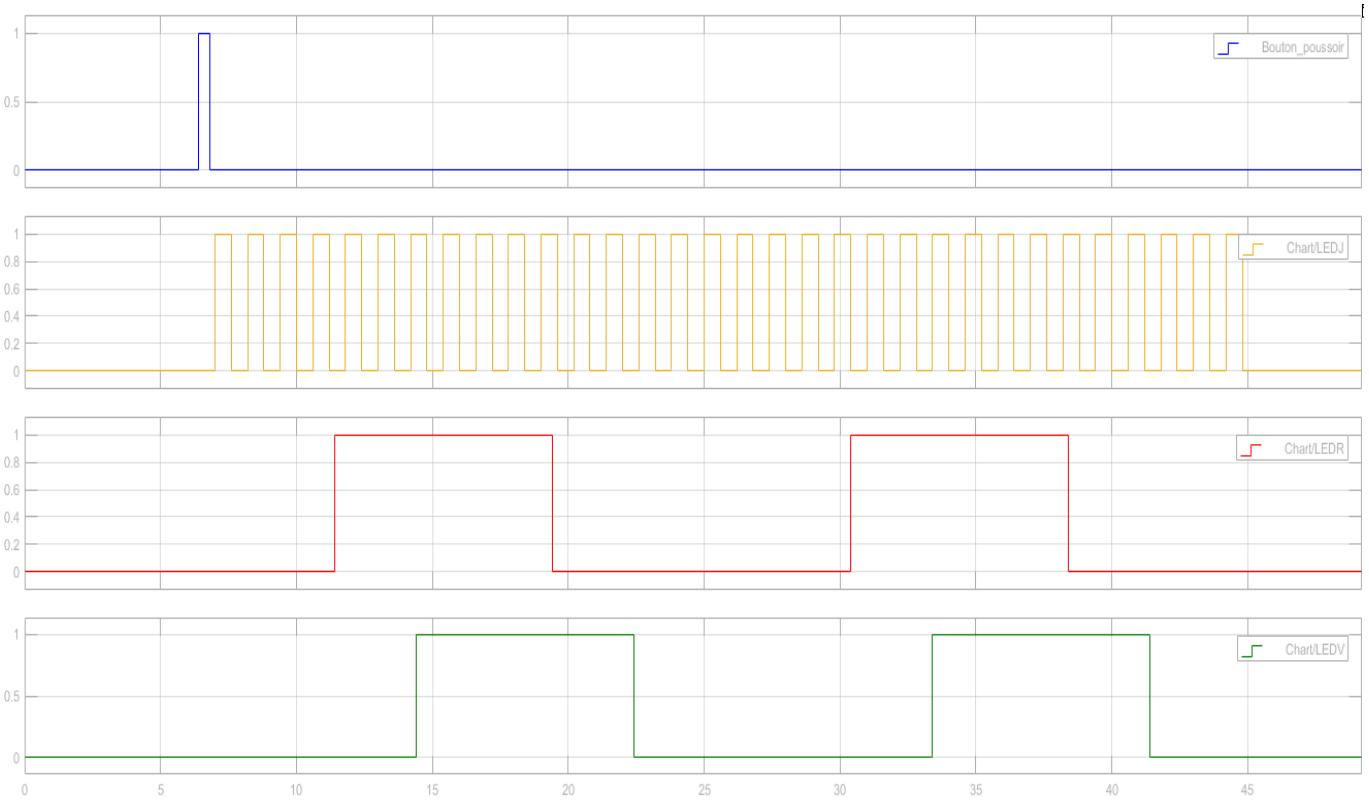
- Une fois le programme chargé, **cliquer** sur le bouton RUN. La simulation démarre. **Vérifier en agissant sur le poussoir**, si le résultat obtenu correspond au comportement attendu.

Pour aller Plus loin ! Une évolution possible consiste à modifier la solution précédente afin de quitter automatiquement, au bout de 10 secondes par exemple, la fonction clignoter pour revenir dans l'état ATTENTE_APPUI_BP. A vous de modifier l'application.

A) Modélisation et implémentation du cycle de fonctionnement de l'Ouvre-portail.

Cette quatrième application consiste à modéliser le comportement de l'Ouvre-portail lors d'une demande d'accès, suite à une action sur la télécommande. Il s'agit ici, de reproduire le fonctionnement du vantail gauche et droit lors d'un cycle d'ouverture/fermeture, ainsi que de l'avertisseur lumineux. La carte Arduino sera associée avec des leds qui seront câblées sur une carte de prototypage (type breadboard).

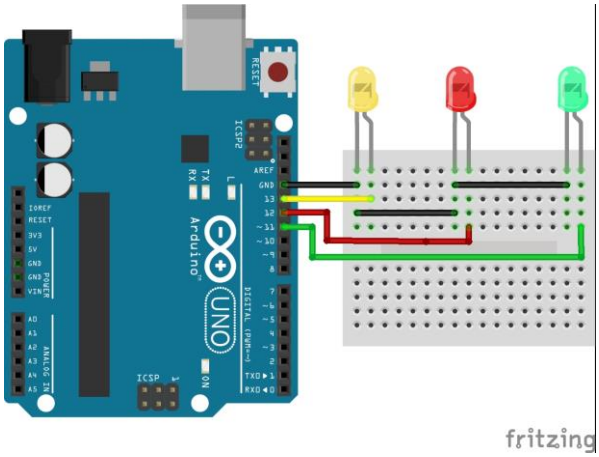
Le fonctionnement attendu doit être conforme aux chronogrammes suivants :



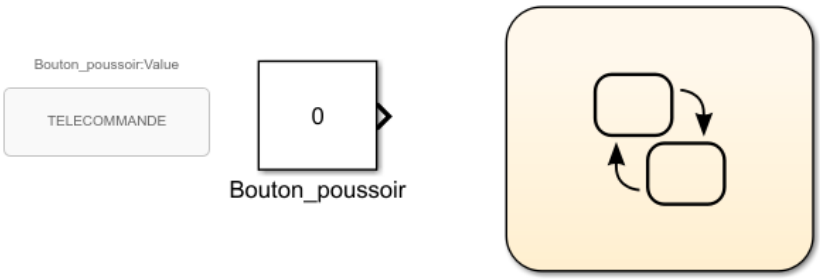
Suite à l'appui sur le bouton poussoir de la télécommande *–signal bleu*, l'avertisseur lumineux clignote *–signal jaune*. Puis, 5 secondes après, le vantail gauche s'ouvre *–signal rouge* suivi 3 secondes plus tard du vantail droit *–signal vert*. Une pause de 8 secondes après la fin de l'ouverture du vantail droit permet au propriétaire de sortir avec son véhicule. Le cycle de fermeture prend ensuite le relais. L'avertisseur lumineux cesse de clignoter 3 secondes après la fermeture du dernier vantail.

Le câblage retenu est proposé ci-contre. La led jaune, *connectée sur la broche 13 de la carte Arduino*, modélise l'avertisseur lumineux. La led rouge, *connectée sur la broche 12*, correspond au vantail gauche. La led verte, *connectée sur la broche 11*, au vantail droit.

Lorsque les leds rouges ou vertes sont allumées, elles indiquent que les vantaux correspondants sont en mouvements (on ne tient pas compte du sens dans cette activité).

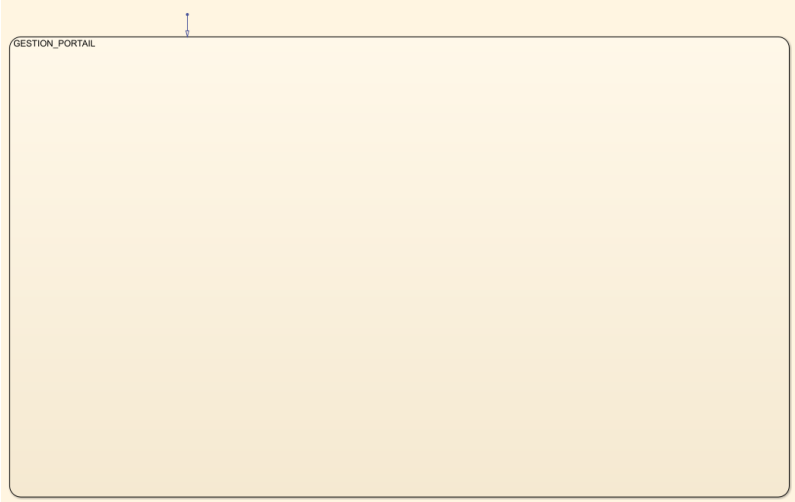


- **Ouvrir** le fichier "GESTPOR.slx" depuis la fenêtre « Current Folder » de Matlab. Vous devez obtenir sur votre écran une fenêtre Simulink conforme à l'image ci-dessous.

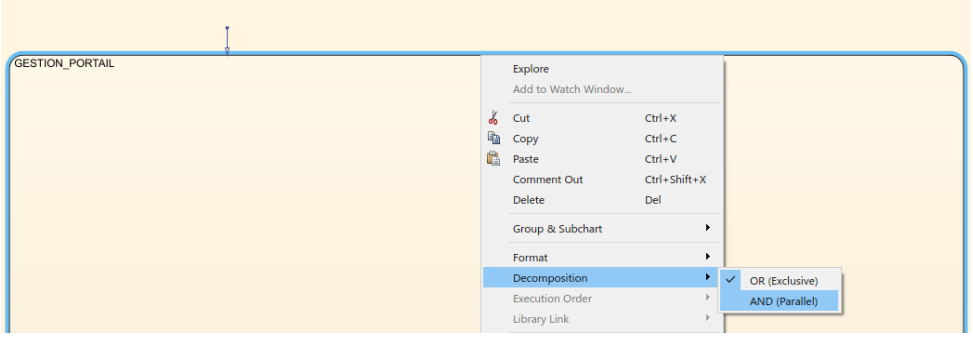


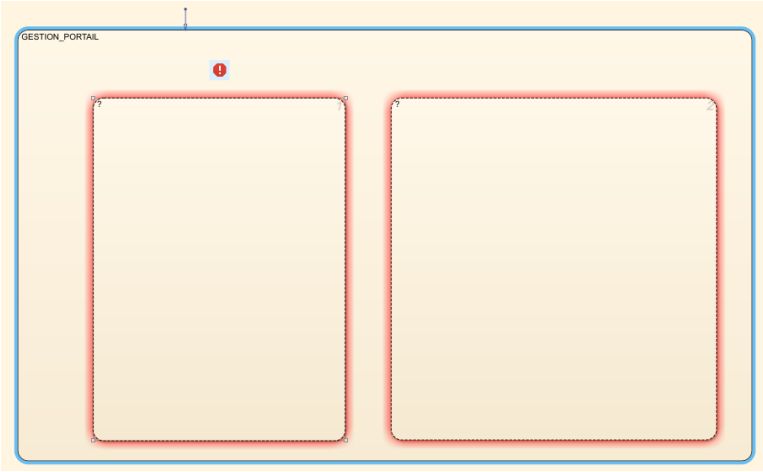
Ouvrir le « Chart » par un double-clic. Comme vous pouvez le constater ci-contre, il est « presque » vide.

Avant de définir les états ainsi que les actions associées, il est judicieux de décomposer le super état « GESTION_PORTAIL » en un état parallèle. En effet, nous pouvons scinder notre application en deux sous-parties. Une partie relative à la gestion du clignotement lumineux et l'autre partie relative au « pilotage » des vantaux. Ces sous-états présentent un contour en pointillés.

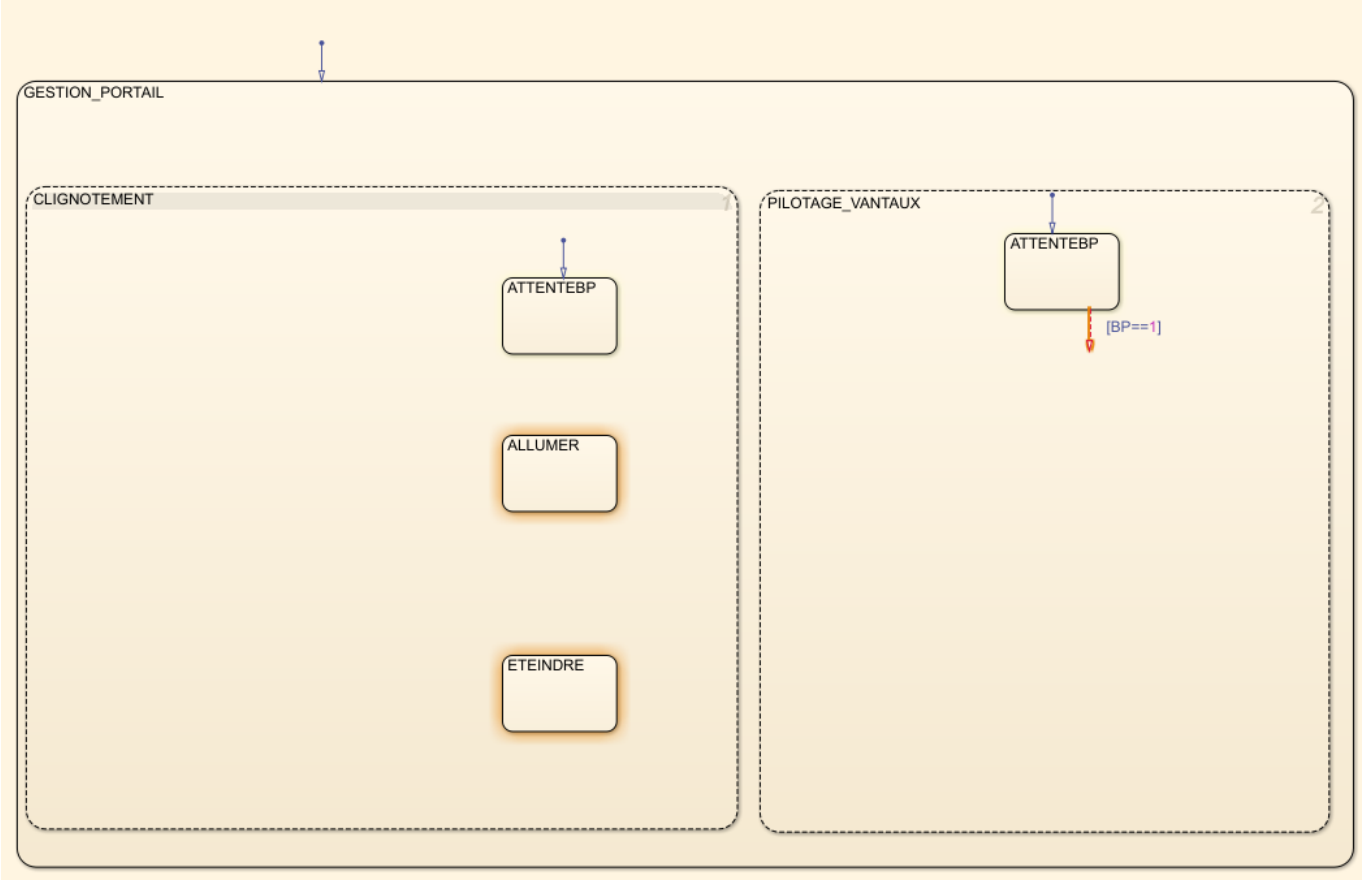


- On accède à la décomposition en **sélectionnant le super–état par un clic gauche**, puis **clic droit**, « Decomposition ». **Sélectionner** AND (Parallel). **Tracer cote à cote** les deux états parallèles en faisant « **glisser-déposer** » deux états « State ».





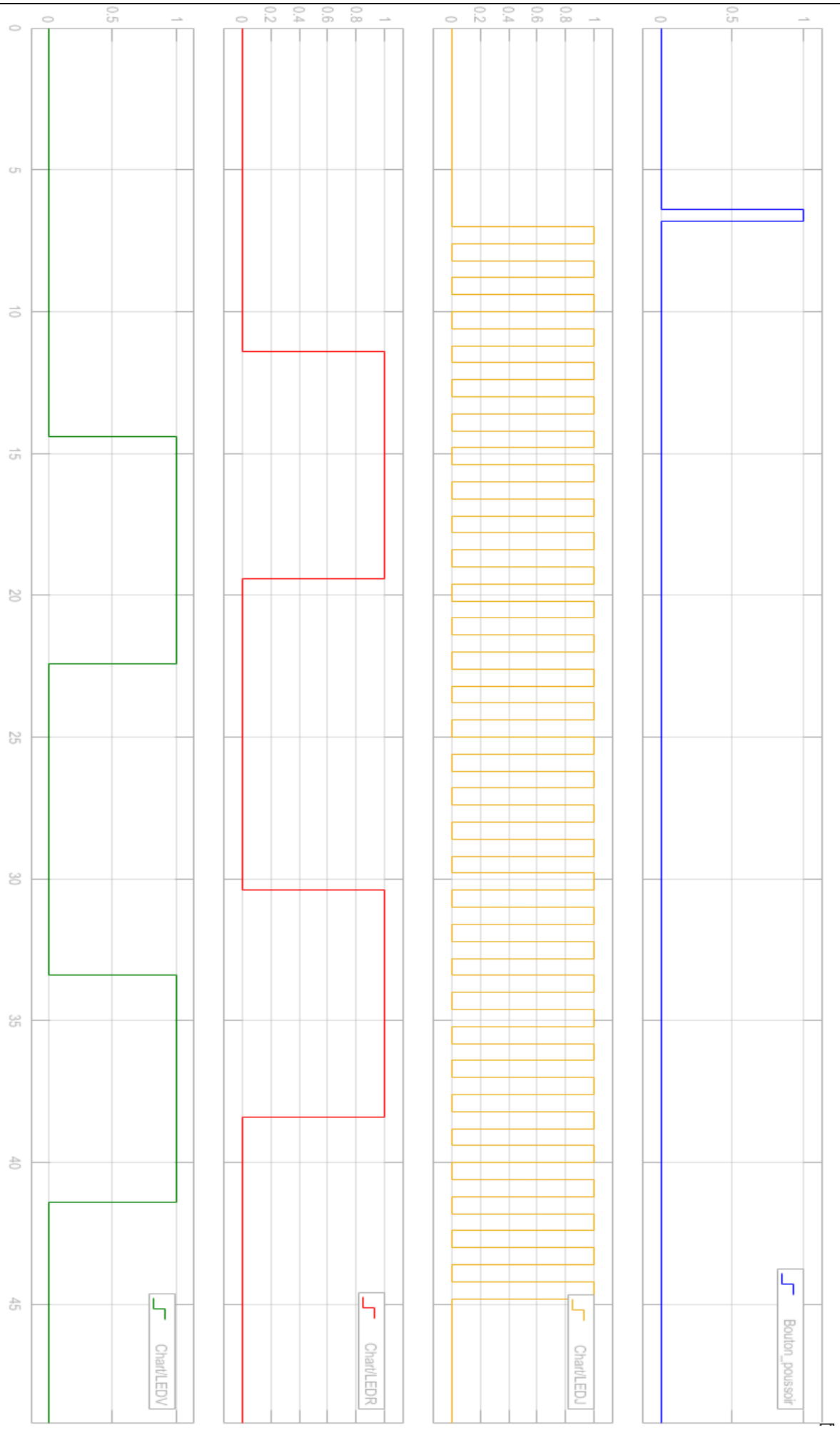
- **Positionner** les différents états, **dessiner** les transitions puis **inscrire** les conditions. Ne pas oublier de **déclarer** les entrées et les sorties du modèle. *Une piste de départ est proposée ci-dessous…*



- **Glisser-déposer** depuis la bibliothèque *Simulink Support Package for Arduino* les sorties numériques. Un « Scope » devra être **positionné** pour observer l'évolution des différents signaux.

- **Configurer** le mode de fonctionnement attendu ainsi que le processeur utilisé puis **implanter** l'application dans la cible. **Vérifier** si le résultat obtenu est conforme à vos attentes.

Une évolution possible consiste à supprimer le « poussoir virtuel » dans simulink puis à implanter un bouton poussoir sur la plaque de prototypage breadboard. On pourra le câbler sur la broche numérique 7 de la carte Arduino par exemple. **A vous de modifier le montage ET l'application…**


0
0.5
1
0
0.2
0.4
0.6
0.8
1
0
0.2
0.4
0.6
0.8
1
0
0.5
1
0
5
10
15
20
25
30
35
40
45
Chart/LEDV
Chart/LEDR
Chart/LEDJ
Bouton_poussoir