Exercices - PPI – LabVIEW Design Pattern + Correction

Exercice 1

En LabVIEW, faire une machine d'état qui simule le feu d'un passage pour piéton.

Votre machine aura 3 états :

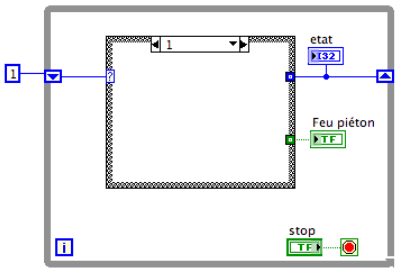
1. attendre bouton,
2. temporisation rouge,
3. traverser

Au démarrage l'état par défaut est le 1 et le feu est rouge. La machine reste dans cet état tant que l'utilisateur ne presse pas sur le bouton « traverser ». Dès que le bouton « traverser » est appuyé, va dans l'état 2 et attend trois secondes, met le feu à vert, puis va dans l'état 3. Dans l'état 3, attend trois secondes, met le feu à rouge, puis va dans l'état 1.

Etapes

En vous inspirant du slide 22, esquissez votre machine d'états à l'aide de 3 boites, une par état et de flèches (transitions).

Transposez votre code en une machine d'états LabVIEW (slide 23). La structure de votre code devra comporter une boucle *while* avec à l'intérieur une *case structure* avec un *case* par état. L'état sera sauvegardé dans un *shift register*. Une ébauche de la structure devrait ressembler à :



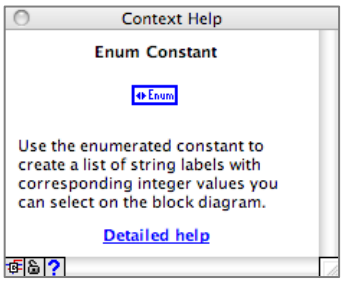
A vous d'ajouter le code pour calculer le nouvel état et allumer/éteindre le feu.

L'attente se fait à l'aide du VI **milliseconds to wait** 

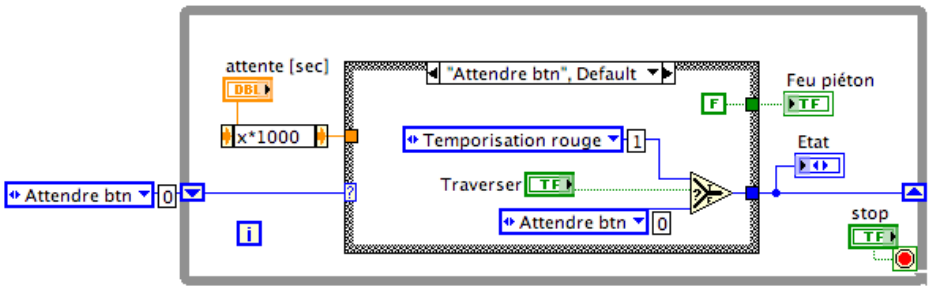
A l'aide du pinceau il est possible de définir les couleurs d'un bouton (boolean).

Testez et validez votre code.

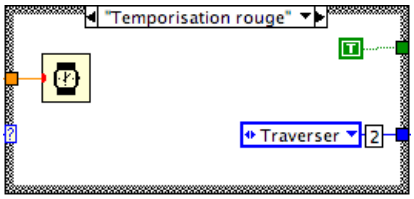
L'état est représenté à l'aide d'une énumération qui permet de « donner un nom à un nombre ». Vous pouvez sans problème employer un nombre à la place.



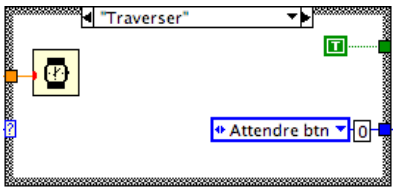
Commence par l'état **Attendre btn (0)** et attend que l'utilisateur presse **Traverser** La LED **Feu piéton** est éteinte



Quand **Traverser** est pressé va dans l'état **temporisation rouge**, attend quelques secondes et allume la LED **Feu piéton** et va dans l'état **Traverser**

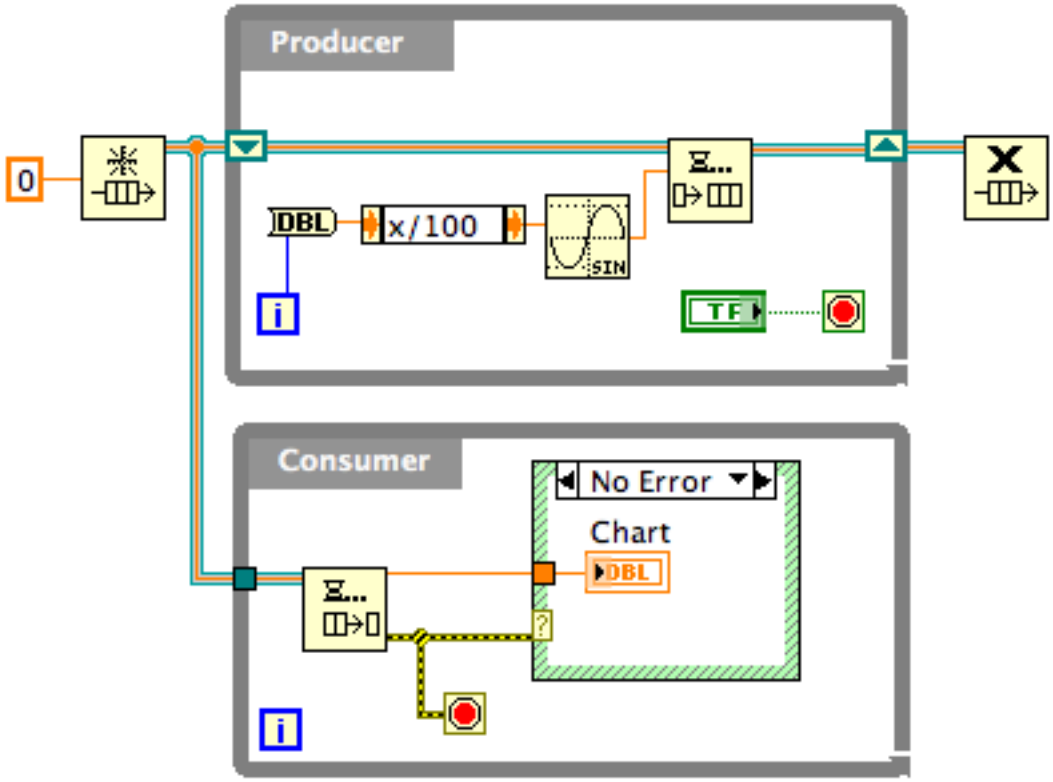


Dans l'état **Traverser**, attend quelques secondes et éteint la LED **Feu piéton** et retourne dans l'état initial **Attendre btn**



Exercice 2

Câblez l'exemple du *slide* 36 – producer/consumer.



Dans chaque boucles ajoutez une temporisation variable à l'aide du VI

milliseconds to wait

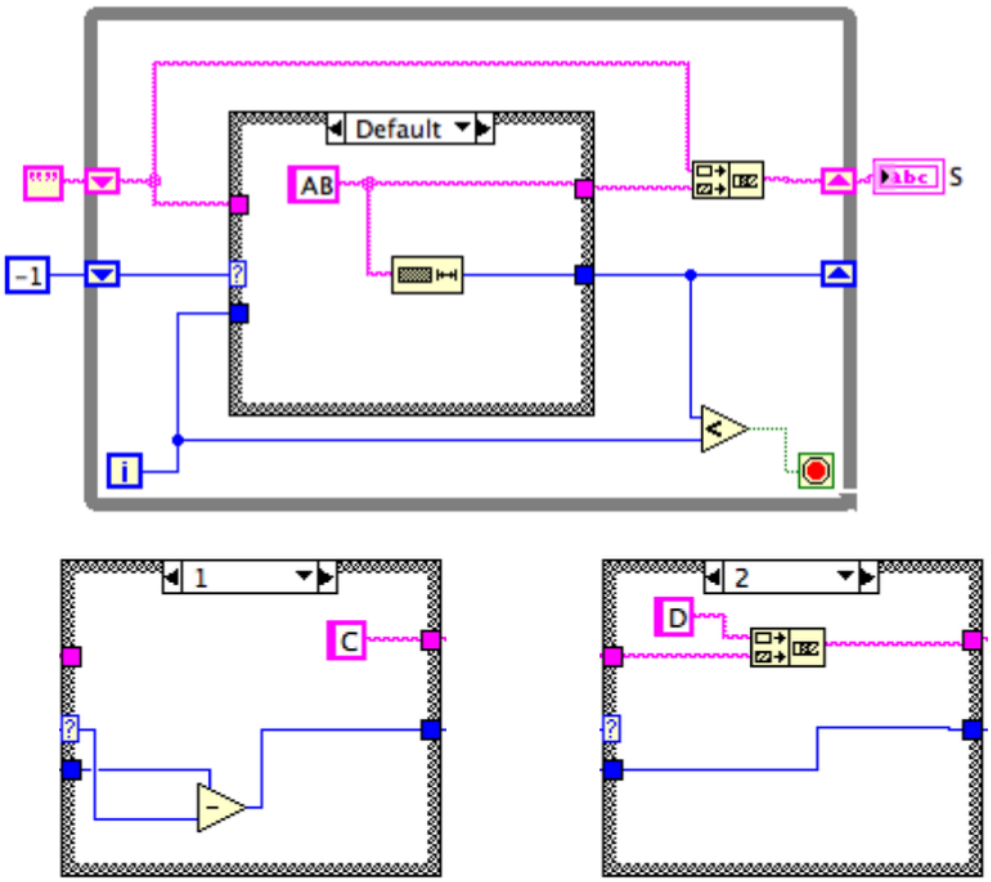
Que ce passe-t-il si la temporisation du *Producer* est plus grande que celle du *Consumer* ? Et inversément ?
Le buffer/queue se rempli jusqu'à ce que la mémoire de votre ordinateur soit *pleine*. Le buffer/queue se vide et le Consumer attend des données.

Comment et pourquoi la boucle *while* du Consumer s'arrête ?
Le bouton **stop** arrête la boucle *Producer*, le buffer/queue est détruit quand le VI **Release Queue** est appelé. La référence sur le buffer/queue est alors invalide. Cela à pour effet de générer une erreur lors de l'appel au VI **Dequeue Element**. L'erreur est connectée sur la condition de l'arrêt de la boucle *Consumer.*

Ajouter une 3e boucle *Producer* générant une sinusoïdale, qu'observez-vous ?
Les signaux sont mélangés de manière plus ou moins aléatoire en fonction de l'attente de chaque boucle.

Exercice 3 (Q. examen 2016)

Que vaut **S** après l'exécution du *diagram* ci-dessous ?



S = ABDABC

La boucle while est executée 3 fois.

SR1 : shift register avec la chaine de caractères

SR2 : shift register numérique

i=0; Case (-1) -> default;	SR1 = empty + '**AB**';	
	SR2 =length(AB) = 2;	2<0 -> false -> continue
i=1; Case (2);	SR1 = empty + AB + **'D'+'AB'**;	
	SR2 = i = 1;	1<1 -> false -> continue
i=2; Case(1);	SR1 = empty + AB + D + AB + **'C'**;	
	SR2 = i - current case = 2 – 1 = 1;	1<2 -> true -> stop