# Laboratoire 6 et 7: Machines à états et communications

## Durée:

2 semaines auquel s'ajoute la semaine sans cours ni évaluation

## Remise:

* 9 octobre à 23h59 au plus tard: diagrammes d'états et pseudo-code
* 23 octobre à 16h59 au plus tard: démonstration du fonctionnement et code

## Objectifs:

* Améliorer sa capacité à créer un diagramme d'états à partir de la description d'un problème (sommatif);
* Se familiariser avec la conception de machines à états qui interagissent entre elles (sommatif);
* Améliorer sa capacité à traduire les diagrammes d'états en pseudo-code (sommatif);
* Mettre en œuvre des machines à états de communications (sommatif);

## Matériel:

* Ensembles de développement Dallas et STM32F4-Disco;
* Matériel requis pour implanter la solution retenue;
* Programmes *247-637 S-0005* et *247-637 S-0001*;
* Vos programmes modulaires développés jusqu'à maintenant.
* Les fichiers mis en ligne à titre de point de départ pour compléter le laboratoire.

## Déroulement:

* Introduction faite par l'enseignant;
* Conception de diagrammes d'états avec intervention de l'enseignant au besoin;
* Rédaction de pseudo-code avec intervention de l'enseignant au besoin;
* Conversion de pseudo-code en programmes;

## Introduction:

La transmission et la réception de données se prêtent bien à une implémentation qui fait appel à des machines à états puisque les transmetteurs devront généralement attendre une requête avant de procéder à l'envoi de données alors que les récepteurs attendront de recevoir des octets avant de confirmer la réception d'un message. Ce laboratoire vous demande d'ajouter des services de communications à des programmes qui doivent permettre à un ensemble de développement Dallas de communique avec un ensemble STM32F-disco.

## Partie 1 : Conception de diagrammes d'états

Concevez les 2 diagrammes d'états des services de transmission et de communication par port série qui dépendent des points suivants:

* La trame à transmettre est constituée dans l'ordre:
  + D'un octet de départ dont la valeur vaut "$";
  + D'un octet dont la valeur correspond au nombre d'octets qui forment les données du message;
  + Des octets qui forment le message;
  + D'un octet dont la valeur permet de vérifier si le message a été bien reçu (*i.e.* *checksum*).
* Le nombre d'octets qui forment les données du message vaut au moins 1 et au plus 16;
* La présence dans le message d'un octet qui vaut "$" entraîne la transmission de la valeur 0 tout de suite après la transmission de cet octet;
* La valeur de l'octet de vérification de l'intégrité du message (*i.e.* le *checksum*) est calculée en additionnant les valeurs des données seulement;
* Il n'y aura pas de transmission si le nombre d'octets à transmettre vaut 0 ou est plus grand que 16;
* Le service de transmission offrira aux autres modules les variables suivantes:
  + Un octet qui servira à formuler des requêtes de transmission (*ex.* unsigned char requete\_tx);
  + Un tampon formé de 16 octets qui serviront à mémoriser les messages à transmettre (*ex.* unsigned char donneesATransmettre[16]);
  + Un octet qui servira à indiquer s'il y a eu erreur de transmission (*ex.* unsigned char statut\_tx);
  + Un octet qui servira à indiquer s'il y a eu mise à jour de l'information relative aux erreurs de transmission (*ex.* unsigned char information\_tx);
* C'est dire que le service de réception:
  + Attendra d'abord de recevoir l'octet de départ;
  + Attendra par la suite de recevoir l'octet qui donne le nombre d'octets qui forment le message;
  + Attendra ensuite de recevoir chacun des octets qui forment les données en s'assurant d'attendre de recevoir un octet qui vaut zéro après chaque réception d'un octet qui vaut la même chose que l'octet de départ (*i.e.* "$");
  + Attendra enfin de recevoir un octet de vérification (*i.e.* *checksum*) une fois que les données auront été reçues;
  + S'assurera de ne pas inclure les octets insérés (*i.e.* les "0x00") dans ce qui représentera les messages reçus;
  + Pourra attendre indéfiniment l'octet de départ sans pour autant rapporter une erreur de réception;
  + Indiquera qu'il y a erreur de réception causée par un délai trop long (*i.e. timeout*) quand il s'écoulera plus de 500ms entre:
    - La réception de l'octet de départ et celle de l'octet qui donne le nombre de données;
    - La réception de l'octet qui donne le nombre de données et la première donnée du message;
    - La réception d'un octet du message et celle de l'octet suivant;
    - La réception du dernier octet du message et celle de l'octet de vérification (*i.e.* *checksum*);
  + Indiquera qu'un message valide a été reçu quand ce sera le cas;
* Le service de réception offrira donc aux autres modules les variables suivantes:
  + Un tampon formé de 16 octets qui serviront à mémoriser les messages reçus (*ex.* unsigned char donneesRecues[16]);
  + Un octet qui servira à indique qu'un message valide a été reçu (*ex.* unsigned char information\_rx);
  + Un octet qui servira à indiquer s'il y a eu erreur de réception ou pas (*ex.* unsigned char statut\_rx);
  + Un octet qui servira à indiquer s'il y a eu mise à jour de l'information relative aux erreurs de réception (*ex.* unsigned char information\_rx);

## Partie 2 : Rédaction du pseudo-code pour la réception et la transmission

Vous devez maintenant utiliser les notions vues en classe pour rédiger le pseudo-code qui correspond aux deux diagrammes d'états que vous avez conçu.

## Partie 3 : Rédaction de code

Vous devez créer les programmes LAB6-Dallas et LAB6-STM32 qui implémentent le protocole qui vient d'être défini. Utilisez au besoin les modules du programme S-0005 et S-0001 pour y parvenir et créez les fichiers "serviceProtocole637.h" et "serviceProtocole637.c" pour mettre en place vos solutions. En principe, les mêmes fichiers pourront servir pour réaliser les deux programmes demandés.