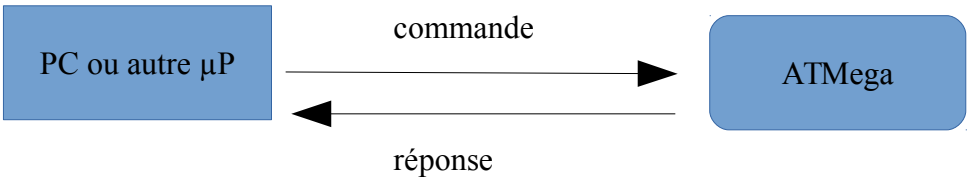
CNAM 2018
Mise en œuvre de capteurs
Dialogues sur la liaison série

Ce tp a pour but de mettre en œuvre différents capteurs (analogique, i2c, bus spécifique,…) et d'établir un protocole très simple pour dialoguer entre deux machines par une liaison série asynchrone.



On utilise une carte basé sur un microcontrôleur 8 bits ATMega328P qui comporte plusieurs capteurs (luminosité, température/humidité, télécommande IR), des boutons (utilisés en mode "analogique"), une led RGB et un relais utilisable comme interface de puissance. Le schéma est donné en annexe. On souhaite exploiter les différentes possibilités de cette carte en passant des commande par un port série asynchrone.

Description du protocole :

Une commande est un ordre envoyé vers l'ATMega.
Une réponse ou un trame d'information est une suite de caractères émise par l'ATMega.

Les commandes commencent par le caractère **'!'** et se terminent par le caractère **'\n'** (newline, 0x0A).
Les réponses commencent par le caractère **'#'** et se terminent par le caractère '**\n**' (newline, 0x0A).
La casse d'une commande peut être quelconque.
Si la commande est non reconnue ou si un paramètre est invalide, aucune action n'est effectuée et reçoit la réponse : **#?**
A la mise sous tension l'ATMega envoie la trame : "#ATCNAM 1.0\n".

Les commandes possibles sont :

Fonction	**Commande**	**Réponse**	**Commentaire**
Lire température	**!T**	**#T;25;ok**	valeur température en °C
Lire Humidité	**!H**	**#H;45;ok**	valeur humidité en %
Lire tension alimentation	**!V**	**#V;12004;ok**	valeur tension en mV
Lire luminosité	**!L**	**#L;123;ok**	valeur luminosité en lux
Led RGB Rouge	**!R0**	**#R;0;ok**	Allume ou éteint la led Rouge

	!R1	**#R;1;ok**	
Led RGB Verte	**!G0** **!G1**	**#G;0;ok** **#G;1;ok**	Allume ou éteint la led Verte
Led RGB Bleu	**!B0** **!B1**	**#B;0;ok** **#B;1;ok**	Allume ou éteint la led Bleue
Couleur led RGB	**!C;245;200;100**	**#C;245;200;100;ok**	Fixe la couleur de la led RGB dans l'ordre R;G;B entre 0 (mini) et 255 (maxi).

Fonction	**Commande**	**Réponse**	**Commentaire**
Relais	**!P0** **!P1**	**#P;0;ok** **#P;1;ok**	État du relais : 0 relâché / 1 collé
Éclairage automatique	**!M0** **!M1**	**#M;0;ok** **#M;1;ok**	Manuel : 0 / Automatique : 1
Seuil de déclenchement éclairage automatique	**!S200**	**#S;200;ok**	Valeur en lux de déclenchement de l'éclairage entre 0 et 10000
Information périodique	**!I30** **!I0**	**#I;30;25;45;123;12004;0;ok** **#I;0;25;45;123;12004;0;ok**	Demande d'information périodique de l'état des capteurs : période,température;humidité; luminosité;tension;boutons; La période est donnée en secondes (0 un seule fois puis arrêt).
Reset	**!Z**		Reset de la carte

NB : Pour la mise au point, il est commode d'afficher des valeurs sur la liaison série. On prendra la convention de commencer toutes les lignes de debug par les caractères //. Ainsi ces lignes pourront être facilement repérées et ignorées par les autres systèmes.

1. Stocker les données reçues et détecter la fin d'une commande

Étudier et charger l'exemple : **Exemples->Communication->SerialEvent**

Modifier l'exemple pour :

- pouvoir changer facilement la taille maximale d'un message et le caractère de fin de trame (CR ou LF) par des macros (#define)
- tronquer les données lorsqu'on dépasse le maximum réservé.
- ne pas stocker le caractère de fin de trame.
- être indépendant ou pas de la casse en fonction d'une macro (#define).

2. Implémenter les premières commandes

Les commandes sont exécutées par une fonction **`int8_t parseCommandLine();`** qui analyse la chaîne reçue et déclenche le traitement éventuel.
Si la commande est reconnue et les paramètres sont corrects, la fonction retourne 0, elle retourne -1 dans le cas contraire. La réponse **ok** ou **?** sera ajoutée par la fonction appelante (loop) en fonction de ce code d'erreur.

- Implémenter les commandes **R**,**G**,**B** et **P**.

3. Lecture température et humidité

Installer une librairie permettant la lecture d'un capteur de type DHT11 et faire fonctionner l'exemple sur la carte de tp.
- Faire une lecture périodique (par exemple 30s pour la mise au point, plusieurs minutes dans une application réelle) de la température et de l'humidité et stocker les valeurs dans des variables globales.
- Implémenter les commandes **T** et **H.**

Attention, pour permettre une réponse la plus rapide possible, et aussi pour préparer l'ajout d'un mode faible consommation, la lecture des capteurs est effectuée périodiquement et pas lors de la réception d'une commande. La réponse à une commande se fait en lisant simplement la valeur de la dernière mesure disponible.

4. Lecture de la luminosité

- En utilisant les fonctionnalités de votre librairie de lecture du BH1750, implémenter la commande **L.**

5 . Éclairage automatique

On suppose ici que le relais commande un lampadaire. Dans le mode automatique, on souhaite allumer la lumière lorsque la luminosité est en dessous d'un seuil réglable, et l'éteindre lorsque le seuil est dépassé. Dans le mode manuel, le relais n'est pas commandé par cette fonction.
Cette fonctionnalité sera réalisée dans une fonction **`void switchLight() ;`** qui sera appelée périodiquement par la boucle principale. La période réelle serait de quelques minutes mais on la réduira à quelques secondes pendant la mise au point.

Pour éviter une commande intempestive sur un nuage ou flash lumineux, on prend la décision sur moyenne glissante des N dernières mesures (N=10 par exemple) et on choisi un hystérésis (± 10 % du seuil par exemple).

- Implémenter les commandes **M** et **S**.
- Réaliser la commande automatique de la lumière.

Pour la moyenne glissante, vous pouvez vous inspirer du programme
Exemples->Analog->Smoothing

6. Lire la tension d'alimentation

La tension d'alimentation de la carte est mise à l'échelle par un pont diviseur et est dirigée vers une entrée analogique.
-Exprimer la relation entre la tension d'alimentation (en mV) et la valeur numérique lue sur l'entrée analogique. Donner la valeur maximale de la tension d'alimentation.
- Écrire une fonction **`int16_t readPwrVoltage() ;`** qui retourne la valeur de la tension d'alimentation en mV.
- Implémenter la commande **V**.

7. Buttons "analogiques"

Très souvent lorsqu'on veut connaître l'état d'un bouton on utilise une entrée logique et une résistance de tirage (éventuellement interne en mode "INPUT_PULLUP"). Cette méthode utilise une broche du microcontrôleur par bouton.
Pour économiser des broches, on peut utiliser une même entrée *analogique* pour plusieurs boutons dans un réseau R/2R. La tension prend plusieurs valeurs différentes en fonction de l'état des boutons. C'est ce qui est fait sur la carte de TP.

- Calculer les valeurs théoriques fournit par le convertisseur analogique pour chaque combinaison possible de l'état des trois boutons.

B3	B2	B1	Tension (volt)	Valeur numérisée	Valeur retournée
0	0	0			0
0	0	1			1
0	1	0			2
0	1	1			3
1	0	0			4
1	0	1			5
1	1	0			6
1	1	1			7
				Trop éloignée des valeurs théoriques possibles	8 (erreur)

On considérera que l'état est indéterminé et on retournera la valeur 8 (erreur) si la valeur numérisée s'écarte de + ou – 20 bits de la valeur théorique.

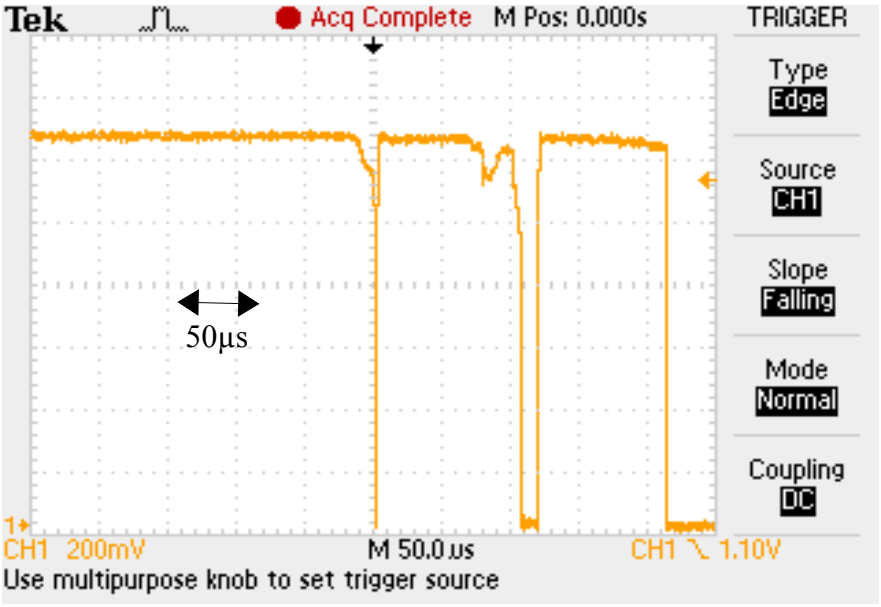
- Écrire une fonction **`uint8_t readButtons() ;`** qui retourne l'état des boutons.

Cette fonction sera appelée périodiquement par la boucle principale. Il faut une période assez faible de l'ordre de 100ms pour être réactif.

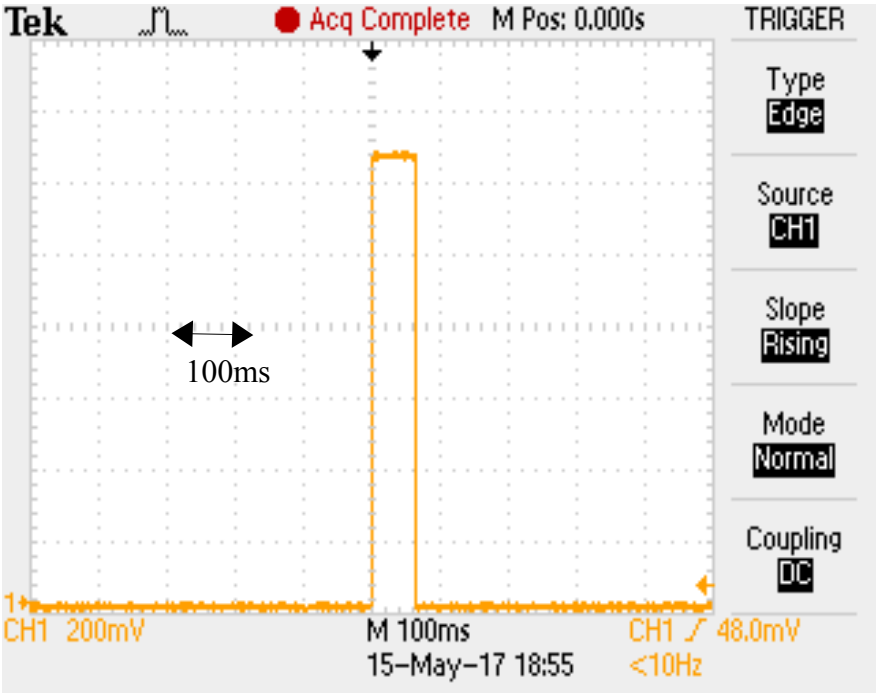
Lorsque la valeur renvoyée par la fonction change, et seulement dans ce cas, on enverra la trame : **#B;valeur_retournée;**

Par exemple, un appui bouton n°1 provoquera l'envoi de trame #B;1; puis le relâchement provoquera l'envoi de #B;0;
L'appui simultané sur les boutons n°1 et n°3 provoquera l'envoi des trames #B;5; puis #B;0; au relâchement si la mesure a lieu au moment ou les deux boutons sont enfoncés. Mais on a plus de chance d'obtenir la séquence #B;1; #B;5; #B;4; #B;0; ou une autre en fonction des instants exacts des appuis et des relâchements. La valeur maximale avant le retour à 0 donne l'état des boutons à prendre en compte.

- Filtrer les états intermédiaires du fait des "rebonds" des buttons.
Lorsqu'on appuie ou qu'on relâche un bouton on constate que le signal ne se stabilise pas instantanément à la valeur finale mais passe par une suite d'états transitoires du fait de la mécanique du bouton. Voici un exemple de signal obtenu en relâchant le bouton n°1



La stabilisation est obtenue en 300 à 400µs environ. Par contre un appui même très court dure beaucoup plus longtemps. Voici le signal obtenu lors d'un appuie très bref (<100ms) :



A cette échelle les états intermédiaires n'apparaissent pas.

Comme on travaille ici avec des valeurs *analogiques* les états intermédiaires peuvent correspondre à des valeurs que l'on obtiendrait avec d'autres combinaisons de boutons.

Voici un algorithme qui permet de filtrer correctement les états intermédiaires :

Quand la fonction est appelée, on prend une successions de 10 mesures. Comme la fonction **`analogRead()`** prend environ 100µs, on aura accès à environ une milliseconde de signal.
On calcule alors la moyenne et la variance des valeurs. Si la variance est trop grande, on considère qu'on est en phase transitoire et on recommence une série de mesures jusqu'à ce que la variance soit assez faible.
On décide alors de la valeur à retourner en fonction de la moyenne de cette dernière série de mesures. Si la variance reste trop élevée même après un grand nombre d'essais, on retournera la valeur 8 correspondant à un état indéterminé.

-Implémenter le filtrage proposé dans la fonction **`uint8_t readButtons() ;`**

8. Trame d'information périodique

Il est parfois plus pratique d'avoir une émission périodique de toutes les informations disponibles plutôt que de devoir émettre une commande séparée pour chaque valeur.

- Implémenter la commande **I.**

9. Faire varier la couleur de la led RGB

Les broches R,G et B de la led RGB peuvent être commandées avec un signal pwm pour faire varier la couleur.

- Implémenter la commande **C**

10. Sécurisation du fonctionnement

Si on entre dans une boucle infinie par suite d'un bug logiciel ou si on attend en bouclant un événement qui n'arrive pas, tout le programme peut se bloquer. Pour éviter ce risque il faut d'abord y penser dès la conception en se posant à chaque instant la question de ce qui se arriverait si les choses ne se passaient pas comme prévu... En dernier recours, il existe la solution du *watchdog* qui resette le micro si on ne passe pas dans la boucle principale pendant trop longtemps.
Activer le watchdog pour une période de 8s et le remettre à zéro dans la boucle principale.
Vérifier son fonctionnement en mettant une boucle infinie sur l'appui d'un bouton.
Enlever ensuite cette portion de code.
Le watchdog permet aussi d'implémenter une commande de Reset (**!Z**). Cette commande ne sera d'aucun secours si le programme est planté mais elle permettra d'initialiser les capteurs en cas de défaillance de l'un d'eux. Sur réception de la commande **Z**, configurer le watchdog à 15ms et entrer dans une boucle infinie. Le wacthdog va entrer en action et faire le reset.

11. Stockage d'une configuration en mémoire non volatile

Pour éviter de devoir entrer les paramètres à chaque mise sous tension on peut utiliser la mémoire EEPROM.
- A chaque modification stocker en EEPROM les valeurs du seuil et du mode de fonctionnement de l'éclairage automatique, de la période de la commande I, ainsi que l'état des led et du relais.
- A l'initialisation reprendre les valeurs depuis l'EEPROM.
- Prévoir un retour à des valeurs par défaut ("réglage usine") par l'appui sur BP1 lors de la mise sous tension.

12. Télécommande infra-rouge

La carte comporte un récepteur de télécommande infra-rouge. Il est compatible avec la plupart des télécommandes infra-rouges commerciales (TV, luminaires,…).

Installer la libraire Irremote : https://github.com/z3t0/Arduino-IRremote
Avec les exemples, notez les codes de quelques touches de votre télécommande.

Implémenter les commandes l'équivalent des commandes R,G,B,P et M à partir des touches de votre télécommande.
Implémenter l'équivalent des commandes C et S à partir des touches de votre télécommande.

13. Thermomètre coloré

En vous aidant de cet article (http://arlotto.univ-tln.fr/arduino/article/thermometre-colore), faites varier la couleur de la led RGB en fonction de la température.
Implémenter une commande **X** pour entrer dans ce mode. Les commandes qui agissent sur la led (R,G,B et C) font alors sortir de ce mode particulier.