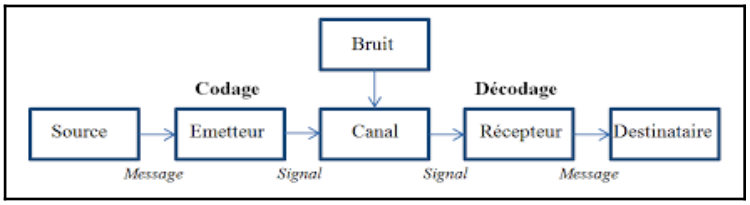
	Spécialité SIN	STI2D
		Terminale
Arduino – Défis de communication		TP

NOM : Vallet--Alff PRENOM : Remy DATE : 25/11/2022

Au cours de ce TP, quatre défis vous sont proposés – pour faire évoluer votre dispositif de communication entre deux Arduino. A vous de trouver les bonnes solutions techniques pour les relever. Vous pouvez les faire dans l'ordre que vous souhaitez.

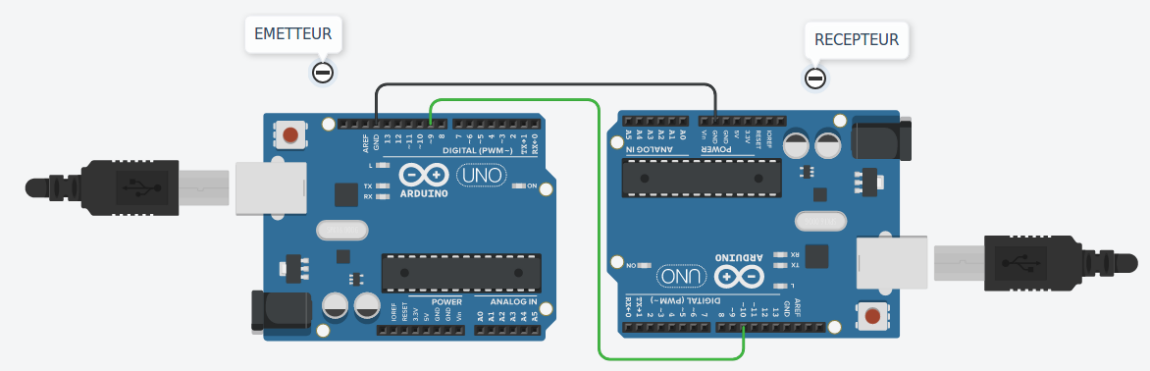
1 – Rappel du dispositif de communication.

Nous repartons du schéma de communication de base, sur lesquels vous avez travaillé au cours des deux précédents TPs.



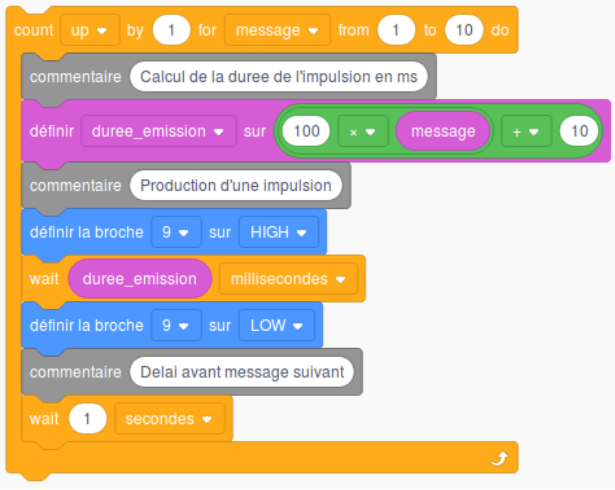
1. La source énonce un **message**
2. que l'émetteur **code** sous forme d'un **signal**
3. qui est acheminé par le **canal**
4. puis **décodé** par le récepteur pour retrouver le **message**
5. qui est reçu par le **destinataire**.

Vous devez vous baser sur le dispositif à deux Arduino reliés par deux fils, pour le faire évoluer. Vous disposez des outils de simulation (TinkerCAD), de prototypage (composants réels) ainsi que des ressources en ligne (Internet) nécessaires à votre travail.



Les programmes d'émission / réception de base sont rappelés ci-dessous :

Programme EMETTEUR (Arduino 1)	Programme RECEPTEUR (Arduino 2)




```
#define BROCHE  10
unsigned long duree_reception ;

void setup() {
  pinMode(BROCHE, INPUT);
  Serial.begin (9600) ;
}

void loop() {
  duree_reception = pulseIn(BROCHE, HIGH, 4000000) ;
  Serial.println (duree_reception / 100000) ;
}
```

Quatre évolutions vous sont proposées :

1. **modifier les programmes pour transmettre du texte (et non plus seulement un entier entre 1 et 10) d'un Arduino à l'autre**
2. **mettre en place un algorithme de chiffrement/déchiffrement pour empêcher un tiers « d'écouter » la communication entre les deux Arduino**
3. **utiliser le principe de la modulation en largeur d'impulsion (MLI ou PWM), non plus pour transmettre un message mais pour commander un servomoteur**
4. **remplacer la MLI par un codage binaire du message numérique à transmettre**

A vous de trouver les bonnes solutions techniques pour les réaliser. Vous pouvez les faire dans l'ordre que vous souhaitez.

2 – Défi : Transmettre du texte d'un Arduino à l'autre.

Le système de communication (par fil ou optique) vous permet de transmettre des « messages » numériques (nombres entiers), grâce à une modulation en largeur d'impulsions.

2.1 – Transmission d'un code ASCII.

Vous allez modifier uniquement les programmes d'émission et de réception pour transmettre des valeurs pouvant coder des caractères alphanumériques. Le dispositif matériel (réel ou simulé) reste absolument inchangé.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
32	20	Space	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4A	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5A	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	DEL

2.1.1 - Modifiez les programmes d'émission et de réception pour qu'ils puissent transmettre des « messages » entiers entre 32 et 126 inclus.

2.1.2 – Conversion d'un caractère en code ASCII.

Cherchez dans la documentation de référence Arduino comment convertir un caractère alphanumérique (chiffre, lettre sans accent, ponctuation) en un entier représentant son code ASCII.

Indices :

- **consultez** la documentation relative aux types de variables « int » et « char ».
- **consultez** la documentation relative à l'opération de conversion explicite de type, nommée « cast » en anglais : https://www.arduino.cc/en/reference/cast
- **trouvez** comment exprimer une constante littérale de type « caractère » en l'entourant d'apostrophes.

Quelle instruction permet de convertir un caractère contenu dans une variable de type char vers un entier de type int ? :

Quelle instruction permet de convertir un code ASCII contenu dans une variable entière de type int vers un caractère de type char ? :

2.1.3 – Modifiez votre programme d'émission – pour lui faire transmettre le code ASCII d'un caractère en particulier (de votre choix).

2.1.4 – Modifiez le programme de réception pour faire afficher dans la console de l'IDE Arduino non pas le code ASCII reçu, mais le caractère correspondant.

2.1.5 - Optimisez ces deux programmes.

Optimisation	But de l'optimisation	Votre solution
Largeur d'impulsion	Dans le programme de base, la durée des impulsions est calculée en multipliant le « message » (entier) par 100 ms. Pour coder 126, il faut donc une impulsion d'une durée de _____ ms. Modifiez le programme pour que les durées de transmission soient plus courtes, sans que n'apparaissent d'erreurs de transmission.	
« offset »	Les « messages » à transmettre vont de 32 à 126. Réalisez un décalage (« offset ») des valeurs transmises pour transposer les messages allant de 32 à 126 vers des codes allant de 1 à _____.	
Période	Dans le programme initial, un délai de une seconde sépare la transmission de deux messages successif. Réduisez cette durée pour augmenter le débit de transmission.	

2.2 – Transmission d'une chaîne de caractères.

Pour transmettre un message constitué d'un texte de plusieurs caractères, il vous faudra transmettre les caractères les uns après les autres, et donc utiliser une boucle pour cela.

2.2.1 – Utilisation de la classe String.

Les librairies de base fournies avec l'IDE Arduino incluent un ensemble d'outils pour manipuler les « chaînes de caractères ». Ceux-si sont regroupés dans une « classe », nommée String.

Vous devez repartir du programme d'émission rapellé dans la section 1.

Déclarez une variable globale, nommée *messageTexte*, de type *String* :

```
String messageTexte ;
```

Modifiez votre programme pour qu'il lise *messageTexte* à partir du terminal de l'IDE Arduino, via la liaison série (USB), grâce aux instructions :

```
if (Serial.available()) {
    messageTexte = Serial.readString () ;
    // … a compléter avec la suite de votre programme d'émission
}
```

Une fois le message lu, **modifiez votre programme** pour qu'il transmette les caractères les uns après les autres. Pour cela, utilisez une boucle *for (… ; … ; …)* ainsi que les fonctions suivantes :

- messageTexte.length () → retourne la longueur de la chaîne de caractères
- messageTexte.charAt (i) → renvoie le caractère situé à la position i dans la chaîne

Pour approfondir :
La notion de « **classe** » est utilisée en « programmation orientée objet » (POO).
La POO est un « paradigme » de programmation qui permet de structurer les programmes en regroupant les informations (données, constantes) avec les portions de programmes (logiciel) qui servent à les manipuler.
En POO, on utilise les termes suivant :

- Un « **objet** » est une chose (quelconque) à propos de laquelle on manipule un certain nombre d'informations. Par exemple, on peut définir un objet « *napoleon* » qui contiendra les *nom*, *prenom*, *dateDeNaissance* et *dateDeDeces* de l'empereur Napoléon Bonaparte.
- Un « **attribut** » est l'une des informations mémorisée pour un objet. Dans notre exemple, *nom*, *prénom*, *dateDeNaissance* et *dateDeDeces* constituent 4 attributs de l'objet *napoleon*. On écrira : *napoleon.nom* pour désigner son nom ...
- Une « **méthode** » est une fonction (donc, une portion de programme) que l'on peut faire exécuter à un objet. Dans notre exemple, on pourra définir la méthode « *dureeDeVie* » qui calculera la durée de la vie d'un objet personne. On écrira : *napoleon.dureeDeVie ()* pour réaliser ce calcul.
- Une classe représente l'ensemble des objets de même type ainsi que les méthodes associées. Les objets qui appartiennent à une certaine classe sont nommés « **instances** » de cette classe. Dans notre exemple, on pourra définir une classe *PersonnageHistorique*, dont *napoleon* serait une instance.

L'usage habituel veut que l'on utilise des noms écrits en minuscules, avec majuscule à chaque changement de mot les composant ; en commençant par une minuscule pour les variables représentant des objets, des attributs ou des méthodes, et une majuscule pour les noms de classes.

2.2.2 – Débit de transmission

Calculez combien de temps il faut à votre système pour transmettre un caractère :

Cette durée dépend-elle du caractère transmis ?

Combien votre système peut-il transmettre de caractères par seconde ?

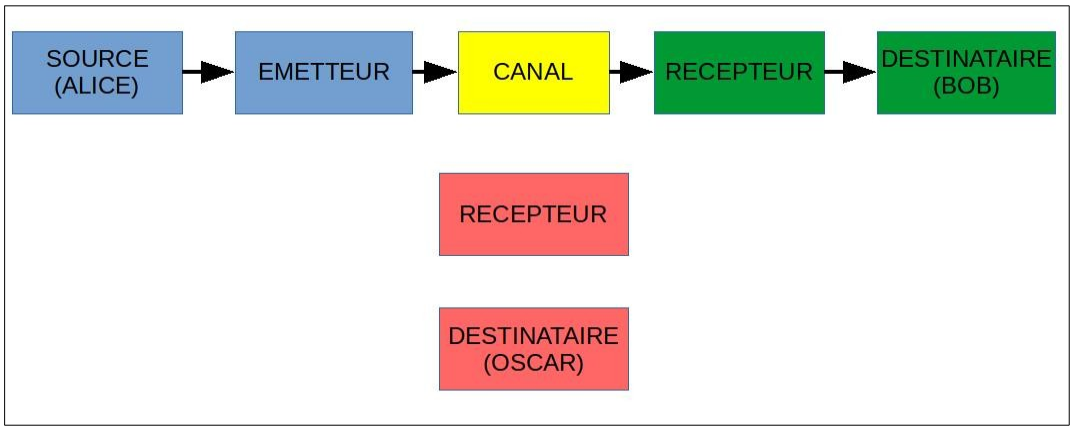
Pour approfondir :
On mesure le **débit** d'un système de communication (quantité de données transmises par unité de temps en bits par seconde (bit/s) ou en octets par seconde. Le débit maximal d'un système est souvent nommé « bande passante » par abus de langage.
Pour tirer le meilleur parti de la voie de transmission en atténuant l'impact du bruit de fond, le protocole de communication ajoute presque toujours à la donnée utile des informations auxiliaires à la transmission. Ces informations comportent presque toujours des signaux de synchronisation et des informations redondantes, déduites des données utiles, qui permettent la détection et la correction des erreurs. Les réseaux peuvent en outre passer des informations d'adresse et de routage, de format du paquet, de statut et de priorité des données. Toutes ces données auxiliaires doivent transiter sur la voie de transmission (canal). Elles ne sont pas transmises aux parties (source et destinataire) qui doivent communiquer à travers la voie.
Dans le contexte de la conception des voies de transmission, on différencie le débit binaire brut, en bits par seconde et le débit d'information (utile), souvent exprimé en octets par seconde. Le débit binaire est celui de la voie, y compris toutes les informations auxiliaires.
Le débit d'information est inférieur ou égal au débit binaire si la voie de transmission n'effectue aucune compression de données. Seules les données utiles comptent pour caractériser la voie de transmission du point de vue de ses utilisateurs.

3 – Défi : Chiffrer la transmission entre les deux Arduino.

La transmission entre l'Arduino émetteur et le récepteur peut être « écoutée » par un troisième intervenant. En cryptologie (qui est la discipline relative au chiffrement), on nomme traditionnellement :

- « Alice » la source du message
- « Bob » son destinataire
- « Oscar » le troisième intervenant qui écoute la communication sur le canal

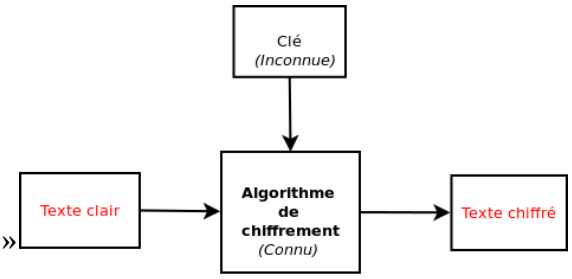
Complétez le schéma de communication où apparaissent le récepteur et le destinataire Oscar :



3.1 – Chiffrement et déchiffrement.

Le chiffrement est un procédé de cryptographie grâce auquel on souhaite rendre la compréhension d'un document impossible à toute personne qui n'a pas la clé de (dé)chiffrement.

On appelle « texte clair » le message non chiffré, « texte chiffré » ce même message une fois chiffré grâce à une « clé ».



Pour approfondir :
Les opérations de **chiffrement** et de **codage** font partie de la théorie de l'information et de la théorie des codes. Le « codage » consiste à préparer de l'information (des données) pour la mémoriser ou la transmettre sous forme de symboles. La compression est un codage : on transforme les données vers un ensemble de mots adéquats destinés à réduire la taille mais il n'y a pas de volonté de dissimuler. Dans le cas du « chiffrement », il y a une volonté de protéger les informations et d'empêcher des tierces personnes d'accéder aux données.
Le chiffrement doit résister à un adversaire « intelligent » qui peut attaquer de plusieurs manières alors que le codage est destiné à une transmission sur un canal qui peut être potentiellement bruité.
Un système de chiffrement est dit :

- symétrique quand il utilise la même clé pour chiffrer et déchiffrer.
- asymétrique quand il utilise des clés différentes : une paire composée d'une clé publique, servant au chiffrement, et d'une clé privée, servant à déchiffrer. Le point fondamental soutenant cette décomposition publique/privée est l'impossibilité calculatoire de déduire la clé privée à partir de la clé publique.

C'est le c

Vous devez choisir une méthode de chiffrement, c'est-à-dire définir les algorithmes de chiffrement et déchiffrement. Ces algorithmes seront « symétriques » : c'est-à-dire que l'opération de déchiffrement se fera par un algorithme effectuant le traitement « inverse » de celui de chiffrement.

Pour simplifier, **choisissez un système de chiffrement** qui travaillera sur un seul « message » (nombre entier) à la fois. Ce nombre entier sera compris entre 1 et 26.

Voici deux suggestions (mais vous pouvez en retenir une autre) :

- **Chiffre « de César »** : Le chiffre de César (ou chiffrement par décalage) est un algorithme de chiffrement très simple que Jules César utilisait pour chiffrer certains messages qu'il envoyait. Il s'agit d'une substitution mono-alphabétique car il remplace chaque lettre par une autre lettre de l'alphabet, toujours la même. Dans ce système, la clef est un entier qui indique le décalage à effectuer.
- **Chiffre « homophone »** : Dans ce système, chaque valeur possible du message sera remplacée par une autre valeur, prise au hasard parmi un ensemble de plusieurs possibilités.

Indiquez la méthode de chiffrement que vous choisissez :

Précisez le type de « clef » que vous utiliserez (par exemple, pour un chiffre de César, ce type est un nombre entier positif) :

Écrivez les algorithmes de chiffrement et déchiffrement dans le tableau ci-dessous.

Chiffrement	**Déchiffrement**
Variables : • messageClair : entier • clef : _____ • messageChiffre : entier Demander messageClair . . . . Renvoyer messageChiffre	Variables : • messageChiffre : entier • clef : _____ • messageClair : entier Demander messageChiffre . . . . Renvoyer messageClair

3.2 – Modification des programmes d'émission et réception.

Vous allez convertir les algorithmes de chiffrement / déchiffrement en instructions en langage C Arduino. Pour cela, vous devez repartir des programmes indiqués en partie 1, dans leur version textuelle. Vous y ajouterez des « fonctions » : l'une pour chiffrer, que vous écrirez dans le programme d'émission ; l'autre pour déchiffrer, que vous placerez dans le programme de réception.

Ces deux fonctions seront nommées chiffrer () et déchiffrer (). Elles prendront pour arguments un message et un clef, et retourneront respectivement les messages chiffrés / déchiffrés.

Émission

```
#define CLEF // A COMPLETER
int message ;

int duree_emission ;

int chiffrer (int messageClair, _________ clef) {
   int messageChiffre ;

   // A COMPLETER

   return messageChiffre ;
}

void setup () {
  pinMode (9, OUTPUT) ;
}

void loop () {
  int messageClair ;
  for (messageClair = 1 ; messageClair <= 10 ;
        messageClair ++) {
    duree_emission =
       (100 * chiffrer (messageClair, CLEF) + 10) ;
    digitalWrite (9, HIGH) ;
    delay (duree_emission) ;
    digitalWrite (9, LOW) ;
    delay (1000);
  }
}
```

Réception

```
#define BROCHE 10
#define CLEF // A COMPLETER
unsigned long duree_reception ;

int dechiffrer (int messageChiffre, _________ clef) {
   int messageClair ;

   // A COMPLETER

   return messageClair ;
}

void setup () {
  pinMode (BROCHE, INPUT) ;
  Serial.begin (9600) ;
}

void loop () {
  duree_reception = pulseIn (BROCHE, HIGH, 4000000) ;
  if ( (duree_reception >= 100000) &&
    (duree_reception < 2700000) ) {
     Serial.println (
       dechiffrer (duree_reception / 100000, CLEF) ) ;
  }
}
```

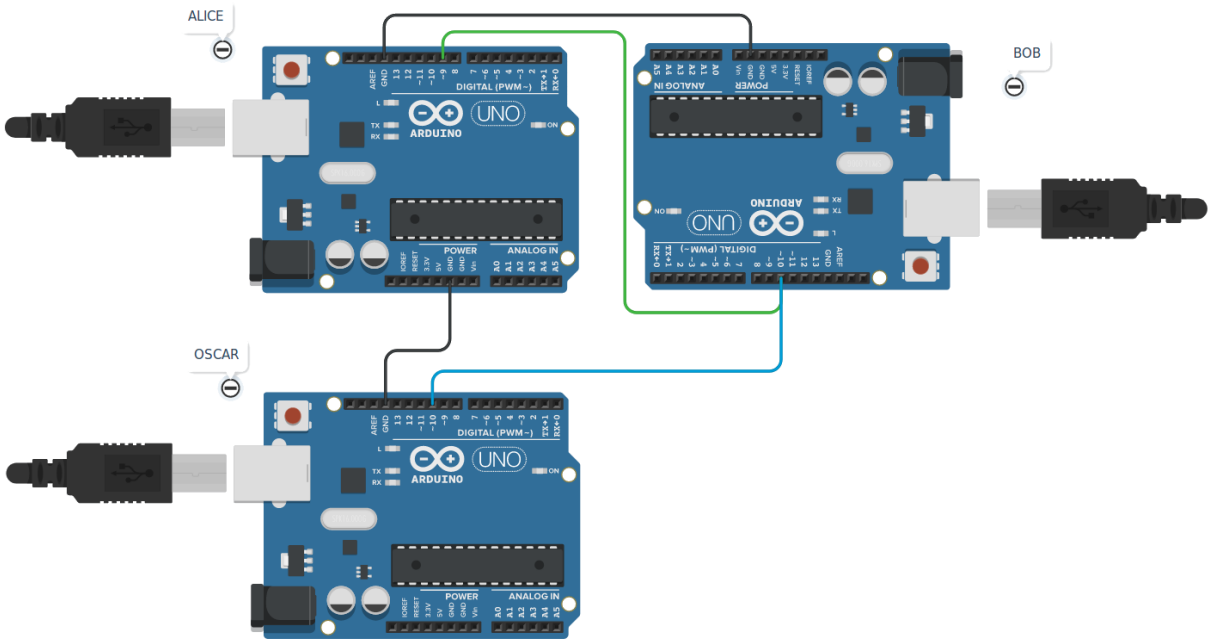
3.3 – Testez le fonctionnement en simulation.

Vérifiez grâce au simulateur TinkerCAD que vos nouveaux programmes de chiffrement / déchiffrement permettent bien la transmission des messages sans les altérer.

Qu'avez-vous observé ?

3.4 – Tentative d'attaque.

Réalisez – avec des composants réels ou en simulation, le système suivant. Celui-ci comporte trois Arduino, qui correspondent aux rôles d'Alice, Bob et Oscar.



Implantez les programmes émission chiffrée dans Alice ; réception chiffrée dans Bob. Vous implanterez le programme de réception non modifié (comme fourni partie 1) dans l'Arduino Oscar.

Décrivez les messages « interceptés » par Oscar :

Jeu de rôle : en binôme, ou avec un trinome.

- L'un des élèves joue le rôle d'Oscar. Celui-ci doit connaître la méthode de chiffrement utilisée, mais pas la clef.
- En observant les messages interceptés, **Oscar doit retrouver la clef**.

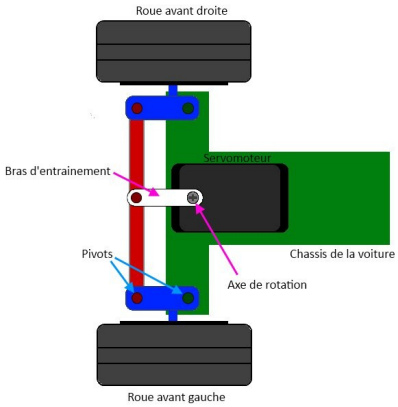
Faites un compte-rendu de vos observations lors de ce jeu de rôle.

	Spécialité SIN	STI2D
		Terminale
Arduino – Défis de communication		TP

4 – Défi : Commander la position d'un servo-moteur.

4.1 – Structure d'un servomoteur rotatif.

Un servomoteur est un système motorisé capable d'atteindre des positions prédéterminées, puis de les maintenir. La position est, dans le cas d'un moteur rotatif, une valeur d'angle.

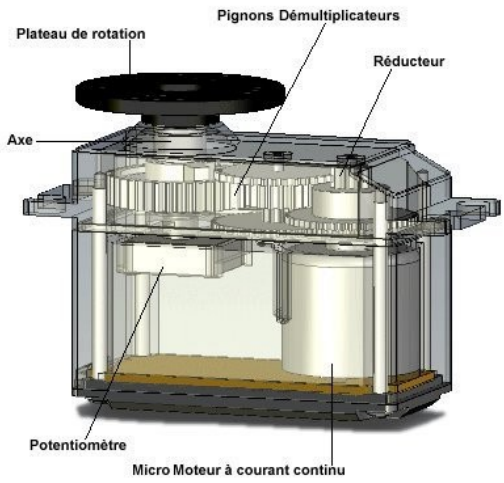


Les servomoteurs sont des actionneurs très utilisés dans les systèmes radio-commandés ou robotisés.

Il peuvent par exemple servir à contrôler l'orientation des roues d'un véhicule radio-commandé, en fonction de la consigne envoyée par l'utilisateur de la radio-commande.

Les servo-moteurs couramment utilisés reçoivent un consigne grâce à un signal modulé en largeur d'impulsions (MLI ou PWM). Il est donc possible de les commander directement depuis un micro-contrôleur comme l'Arduino, à travers une de ses sorties numériques.

Pour approfondir : fonctionnement d'un servo-moteur

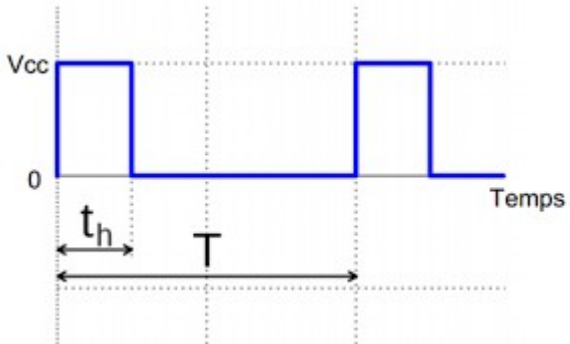


Un servomoteur (ou « servo » en abrégé) est un appareil mécanique complexe qui comporte un système d'asservissement. Il se compose :

- d'un moteur à courant continu
- des engrenages pour former un réducteur (en plastique ou en métal)
- d'un capteur de position de l'angle d'orientation de l'axe (un potentiomètre bien souvent)
- d'une carte électronique pour le contrôle de la position de l'axe et le pilotage du moteur à courant continu

Mécaniquement , il s'agit d'un moteur avec réducteur . Le mouvement de sortie est une rotation . Lorsque le moteur tourne, l'axe du servo change de position, ce qui modifie la résistance du potentiomètre. Le rôle de l'électronique interne est alors de commander le moteur pour que la position de l'axe de sortie soit conforme à la consigne reçue. Le « débattement » d'un servomoteur courant est de 180°.

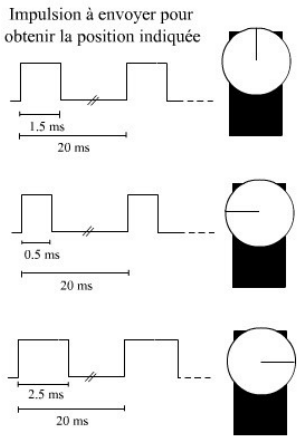
4.2. Commande d'un servo-moteur rotatif.



FLT

La consigne envoyée au servomoteur n'est autre qu'un signal électronique de type PWM. Il dispose cependant de deux caractéristiques indispensables pour que le servo puisse comprendre ce qu'on lui demande. À savoir : une fréquence fixe de valeur 50Hz (comme celle du réseau électrique EDF) et d'une durée d'état HAUT elle aussi fixée à certaines limites.

Le « chronogramme » du signal de commande est représenté si dessus. Pour les servomoteurs grand-public (issus du monde du modélisme), les valeurs suivantes sont utilisées :



- T est la « période » du signal PWM ; elle est fixée à 20 ms. **Calculez** la fréquence de ce signal : _____ Hz

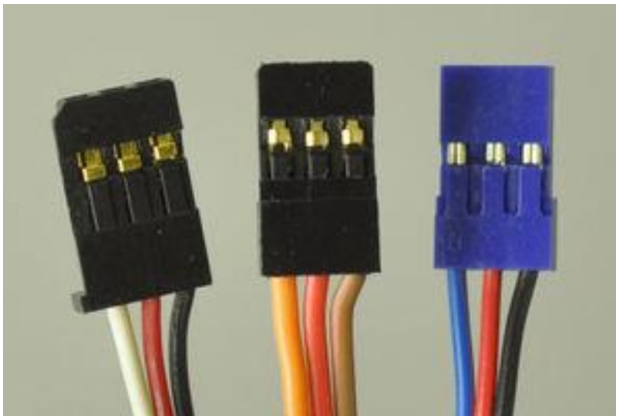
- t_h représente la durée pendant laquelle le signal de commande est à l'état haut. Cette durée dépend de la consigne que l'on souhaite transmettre au servo, c'est-à-dire de la position angulaire que l'on souhaite qu'il prenne (cf figure à gauche). Relevez les valeurs de t_h pour les positions angulaires suivantes :

Position (°)	**t_h (ms)**
0	
90	
180	

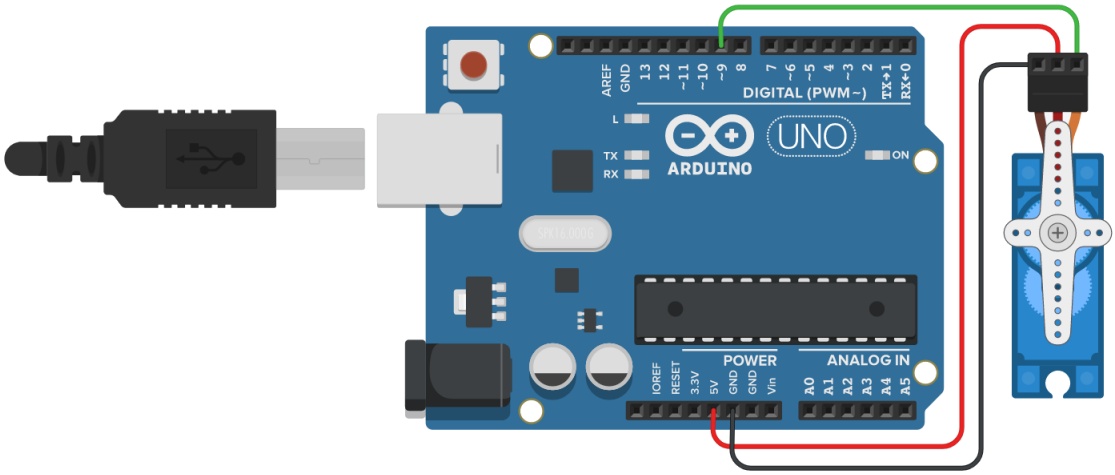
4.3 – Production du signal PWM de commande par programme.

Les servomoteurs sont équipés de prises à 3 contacts. En fonction de la marque, les couleurs de fils peuvent différer. Il n'y a pas de norme à ce sujet. Il faut donc se reporter à la documentation du fournisseur.
En général on trouve:

- Brun ou noir = MASSE/GND (borne négative de l'alimentation)
- Rouge = alimentation du servomoteur (Vservo, borne positive de l'alimentation)
- Orange, jaune, blanc ou bleu = Commande de position du servomoteur



Réalisez le dispositif ci-dessous dans TinkerCAD.



Ecrivez l'algorithme permettant de produire un signal PWM sur la broche 9 de l'Arduino, pour une consigne angulaire indiquée dans une variable nommée « *angle* ».

Variable : angle : entier

Dans TinkerCAD, créez le programme (en langage C) correspondant à votre algorithme. Pour cela, vous utiliserez les deux fonctions suivantes :

delayMicroseconds (d) → attente d'une durée d, d étant en micro-secondes.
ATTENTION : cette commande n'accèpte pas de valeur de d supérieure à 16384. Pour générer une délai d'attente supérieur, il convient d'appeler plusieurs fois cette commande.

map (x, fromLow, fromHigh, toLow, toHigh) → **Cherchez** la description de cette fonction dans la documentation de référence Arduino. **Expliquez** à quoi elle sert :

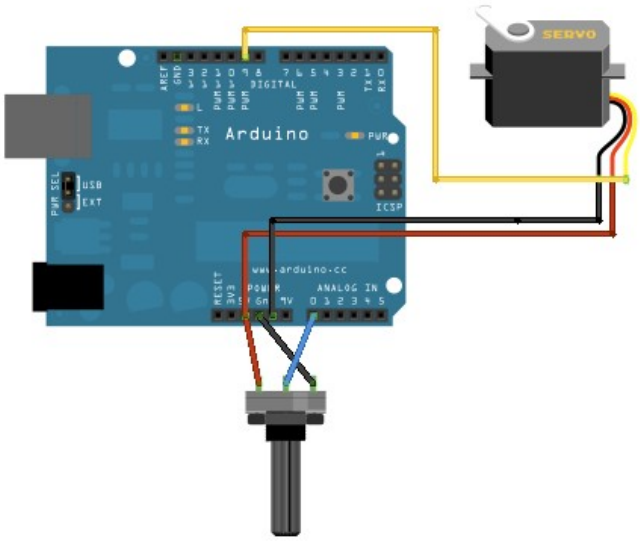
Saisissez votre programme dans l'éditeur de code de TinkerCAD.

Lancez la simulation pour plusieurs valeurs de angle entre 0 et 180.

Une fois votre programme mis au point en simulation, **réalisez le montage réel.**

Qu'observez-vous ?

4.4 – Utilisation de la librairie Servo.



Les bibliothèques logicielles fournies avec l'IDE Arduino comportent un ensemble d'outils permettant de gérer facilement les servomoteurs courants, sans devoir programmer soi-même la production de PWM. Il s'agit de la classe *Servo*.

Réalisez le montage ci-contre :

(la valeur du potentiomètre a peu d'importance ; toute valeur supérieure à 1kΩ conviendra).

ATTENTION : faites vérifier votre montage par le professeur avant de le brancher.

Dans l'IDE Arduino, chargez l'exemple « knob » dans la section « Servo ». Téléversez-le dans l'Arduino.

Knob

```
/*
 Controlling a servo position using a potentiometer (variable resistor)
 by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>

 modified on 8 Nov 2013
 by Scott Fitzgerald
 http://www.arduino.cc/en/Tutorial/Knob
*/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 0;  // analog pin used to connect the potentiometer
int val;    // variable to read the value from the analog pin

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
}

void loop() {
  val = analogRead(potpin);            // reads the value of the potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 180);     // scale it to use it with the servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo position according to the scaled value
  delay(15);                           // waits for the servo to get there
}
```

Qu'observez-vous ?

Expliquez à quoi servent les instructions suivantes :

myservo.attach(9);

val = analogRead(potpin);

val = map(val, 0, 1023, 0, 180);

myservo.write(val);

5 – Défi : coder le message en binaire.

5.1 – Codage en binaire.

La modulation en largeur d'impulsion présente plusieurs inconvénients :

- elle ne permet pas d'augmenter significativement la quantité d'information véhiculée par chaque « impulsion », car cela conduirait à en allonger considérablement la durée
- si l'on cherche à réduire la durée des impulsions, la précision de la mesure de leur durée devient critique, et parfois délicate à obtenir

Calculez la durée de l'impulsion MLI qui coderait l'entier 255 selon le protocole de base (rappelé section 1) : 255 000 ms soit 255 s.

Dans la pratique, la MLI est donc utilisée soit pour la commande d'actionneurs simples (par exemple, des servo-moteurs), soit pour coder des valeurs binaires elles-mêmes combinées pour former des messages plus longs.

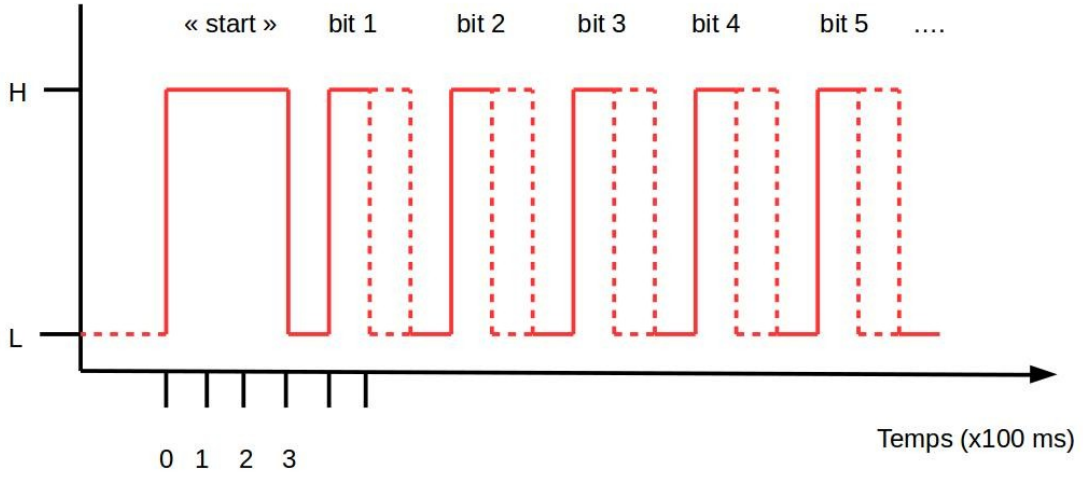
Dans cette partie, il vous est proposé de remplacer la transmission de messages en MLI par un codage en binaire.

Chaque message à transmettre sera un entier compris entre 0 et 255 inclus. Cet entier sera converti en son expression binaire : 8 bits (soit un octet).

Chaque bit sera codé en MLI :
- 0 sera représenté par une impulsion à l'état haut de 100 ms
- 1 sera représenté par une impulsion à l'état haut de 200 ms

Pour permettre au récepteur de savoir lorsqu'un message commence, une impulsion plus longue (300 ms), dite « start », sera transmise avant le premier des 8 bits du message.

Le codage est représenté sur le chronogramme ci-dessous :



Le début de chaque impulsion correspondant aux 8 bits sera espacé de manière constate de 300 ms, quelle que soit la valeur du bit transmis.

Calculez la durée totale nécessaire à transmettre une « trame », c'est-à-dire les 8 bits d'un message : 2 400 ms.

Combien votre système peut-il transmettre de messages d'un octet par seconde ?

5.2 – Programme d'émission.

Complétez le programme d'émission présenté ci-dessous.
Exprimez à côté l'algorithme correspondant :

Programme émission	Algorithme

```
void setup () {
  Serial.begin (9600) ;
  pinMode (9,        ) ;
}

void loop () {
  int message, one_bit, counter ;
  if (Serial.available ()) {
    message = Serial.parseInt () ;
    digitalWrite (9, HIGH) ;
    delay (300) ;
    digitalWrite (9,    ) ;
    for (counter = 0 ;          < 8 ; counter++) {
      one_bit = message % 2 ;
      delay (100) ;
      digitalWrite (9, HIGH) ;
      if (one_bit == 1) {
        delay (    ) ;
        digitalWrite (9, LOW) ;
      } else {
        delay (100) ;
        digitalWrite (9, LOW) ;
        delay (100) ;
      }
      message = message / 2 ;
    }
  }
}
```

Cherchez l'instruction Serial.parseInt () dans la documentation. **Expliquez** à quoi elle sert :

elle renvoie le premier nombre valide antier a partir de la position courante

Indiquez le « poids » du premier bit transmis (après l'impulsion de start) :

c'est un 1

Indiquez le « poids » du dernier bit transmis :
C'est le 0

Faites « tourner » l'algorithme « à la main » pour déterminer, pour les 3 « messages » suivants, les durées des 9 impulsions correspondant à chacun :

Message		**Durée des impulsions (ms)**								
Décimal	**Binaire**	**Start**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
54	110110	300	200	200	100	200	200	100		
136	10001000	300	200	100	100	100	200	100	100	100
168	10101000	300	200	100	200	100	200	100	100	100

5.3 – Algorithme de réception.

Voici deux algorithmes de réception exprimés en pseudo-code :

Algorithme 1

Variables : duree_reception, one_bit, message : entiers

Initialiser la liaison série à 9600 bauds
Configurer la broche digitale 10 en entrée

Répéter indéfiniment :
- Attendre sur la broche 10 une impulsion ;
  stocker sa durée en microsecondes dans duree_reception
- Si duree_reception / 100000 est égal à 3 alors faire :
  - message ← 0
  - Répéter 8 fois :
    - Attendre sur la broche 10 une impulsion ;
      stocker sa durée en microsecondes dans duree_reception
    - one_bit = (duree_reception / 100000) - 1
    - message ← (message * 2) + one_bit
  - Envoyer message sur la liaison série (vers le moniteur de l'IDE)

Algorithme 2

Variables : duree_reception, one_bit, message, poids : entiers

Initialiser la liaison série à 9600 bauds
Configurer la broche digitale 10 en entrée

Répéter indéfiniment :
- Attendre sur la broche 10 une impulsion ;
  stocker sa durée en microsecondes dans duree_reception
- Si duree_reception / 100000 est égal à 3 alors faire :
  - message ← 0
  - poids ← 1
  - Répéter 8 fois :
    - Attendre sur la broche 10 une impulsion ;
      stocker sa durée en microsecondes dans duree_reception
    - one_bit = (duree_reception / 100000) - 1
    - message ← message + one_bit * poids
    - poids ← poids * 2
  - Envoyer message sur la liaison série (vers le moniteur de l'IDE)

Comparez ces deux algorithmes. **Expliquez** leurs différences :

A partir du tableau des durées d'impulsions calculé en 5.2, **déterminez le résultat du décodage** selon les deux algorithmes. Lequel de ces deux algorithmes permet-il un décodage correct des trames ?

Durée des impulsions (ms) – reporté de 5.2									**Décodage**	
Start	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Par algo 1**	**Par algo 2**
300	200	200	100	200	200	100				
300	200	100	100	100	200	100	100	100		
300	200	100	200	100	200	100	100	100		

5.4 – Programme de réception.

Complétez le programme de réception sur la base de l'algorithme que vous avez retenu :

```
#define BROCHE  10
unsigned long duree_reception ;
int 

void setup() {
  pinMode (BROCHE,      ;
  Serial.begin (9600) ;
}

void loop() {
  duree_reception = pulseIn (BROCHE, HIGH, 4000000) ;

  if ( (duree_reception / 100000)     ) {
    message = 0 ;
    
    for (int i = 0 ; i < 8 ; i++) {
      duree_reception = pulseIn (BROCHE, HIGH, 4000000) ;
      
      
    }
  Serial.println (      ) ;
  }
}
```

5.5 – Tests et mise au point.

Testez votre système en simulation ou avec des compsants réels.

Fonctionne-t-il comme attendu ?

Si non, comment procédez-vous pour le mettre au point ?

5.6 – Optimisation.

Pour optimiser le débit, en réduisant la durée d'une trame, il est possible d'utiliser l'impulsion « start » pour transmettre le premier bit de données (il n'y a alors que 7 autres bits à transmettre en plus). Pour cela, l'on peut par exemple convenir que l'impulsion start aura la durée suivante :

- 300 ms si le premier bit vaut 0
- 400 ms si le premier bit vaut 1

Modifiez vos programmes pour implémenter cette optimisation.
C'est le 0