Laboratoire 1 : USART. Débogage numérique

Comme dans tout travail d'ingénierie, des bogues peuvent apparaître et apparaîtront dans les systèmes embarqués. Dans des conditions de travail, il est important d'avoir un moyen de communiquer avec l'appareil embarqué. Pour un aperçu des méthodes de dépannage possibles, nous donnerons une brève introduction, puis étudierons en détail l'interface série USART, couramment utilisée pour la communication série entre deux appareils.

1. Qu'est-ce qui est différent du dépannage habituel ?

La raison pour laquelle le débogage intégré est plus difficile que le débogage logiciel classique découle de plusieurs problèmes :

- Un débutant est habitué aux outils de haut niveau génériques, pour la plupart gratuits (par exemple, les IDE, les débogueurs, etc.). Pour les systèmes embarqués, il est possible que l'utilisateur n'ait pas accès à des outils matériels et logiciels dédiés.
- Si vous n'avez pas de débogueur spécialisé et que vous essayez d'en utiliser un générique (par exemple Remote GDB), qu'est-ce qui vous fait penser que vous avez la pile réseau requise pour ce débogueur ? Si vous avez implémenté une telle pile, êtes-vous sûr qu'elle fonctionne correctement ?
- Même si vous aviez un débogueur spécialisé (par exemple, une sonde Lauterbach), vous aurez très probablement besoin de configurations de débogueur spécialisées pour fonctionner avec votre matériel (par exemple, Practice Scripting Language for Lauterbach T32)
- Le débogage invasif peut affecter le comportement de votre code - pensez RTOS (systèmes d'exploitation en temps réel), SMP (systèmes multiprocesseurs) ou votre circuit (par exemple, modifiez le circuit pour mesurer le courant).
- Votre matériel peut avoir des erreurs.
- Même l'impression de messages d'erreur peut ne pas fonctionner car vous devez parfois implémenter une telle fonction et elle peut contenir des bogues en elle-même.

Cependant, les principes de débogage sont les mêmes que dans les logiciels de haut niveau : il faut comparer ce que l'on attend du système (code/circuit) avec ce qu'il fait réellement, et pour cela il faut de la visibilité.

2. Outils nécessaires

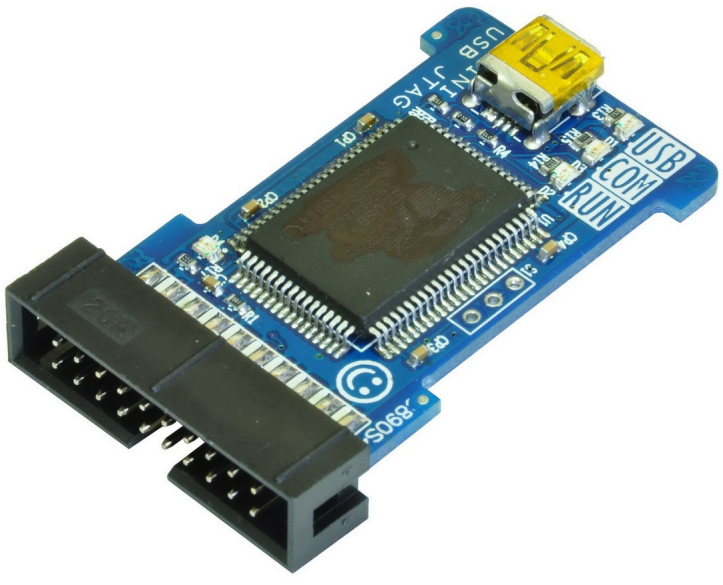
La visibilité au niveau matériel est obtenue grâce à une certaine forme d'entrée-sortie (si disponible) :

- Débogage des LED - vérification de l'état vrai/faux
- **messages sur l'interface série / USART - débogage via USART, bluetooth, etc.**
- Débogage avancé pour afficher et modifier la mémoire / les registres (voir JTAG ci-dessous)
- Le bouclage (connexion des sorties aux entrées) peut fournir des informations sur la manière dont les commandes sont envoyées aux périphériques externes.

Appareils de mesure :

- Multimètres (pour valeurs statiques)
  - Résistance : connectez simplement un composant de circuit entre les sondes.
  - Tension : connectez en parallèle - la sonde positive (rouge) au point de potentiel supérieur, la sonde négative (noire) au point de potentiel inférieur. Pour un potentiel de point unique : sonde négative à GND, sonde positive au point souhaité.
  - Courant : connecter en série.
  - Commencez toujours par régler l'échelle du multimètre au maximum, puis corrigez progressivement l'échelle jusqu'à obtenir la mesure la plus précise.
  - Pour vérifier la polarité des diodes : Sélectionnez Diode Check. Si vous placez la sonde négative sur l'anode et la sonde positive sur la cathode, le multimètre émettra un bip.
  - Pour vérifier les courts-circuits/la connectivité : utilisez la vérification des diodes - si l'appareil émet un bip, il y a un court-circuit/une connexion entre ces deux points.

- OSCILLOSCOPE
- Analyseurs logiques (pour signaux numériques)
- Analyseurs de protocole (pour les protocoles embarqués, comme I2C, SPI, etc.)
- Débogueurs basés sur JTAG - peuvent avoir des fonctions très avancées : lire/modifier la mémoire, interagir avec le noyau, arrêter l'horloge système, etc.



Exemple de flux de débogage

Voici un exemple de flux de débogage :

- Vérifiez la feuille de catalogue et le schéma. Accédez-vous aux bons registres ? Les périphériques sont-ils connectés aux broches que je pensais qu'ils étaient ?
- Est-il possible de déboguer via des messages de débogage (équivalent à printf) ? :
- Avez-vous une pile Ethernet? Si tel est le cas, envisagez SSH, NFS, etc. pour effectuer le débogage de printf.
- Si vous disposez d'une pile réseau, des protocoles plus simples sont peut-être disponibles : UART ? Pouvons-nous connecter quelque chose via UART ? Peut-être un PC, un module Bluetooth HC-05, un LCD ?
- Le débogage des LED peut être envisagé.
- Isoler le problème à l'aide d'outils de mesure : multimètres, oscilloscopes, analyseurs logiques.
- Essayez un débogueur JTAG ou tout autre débogueur dont vous savez qu'il fonctionne avec votre appareil.

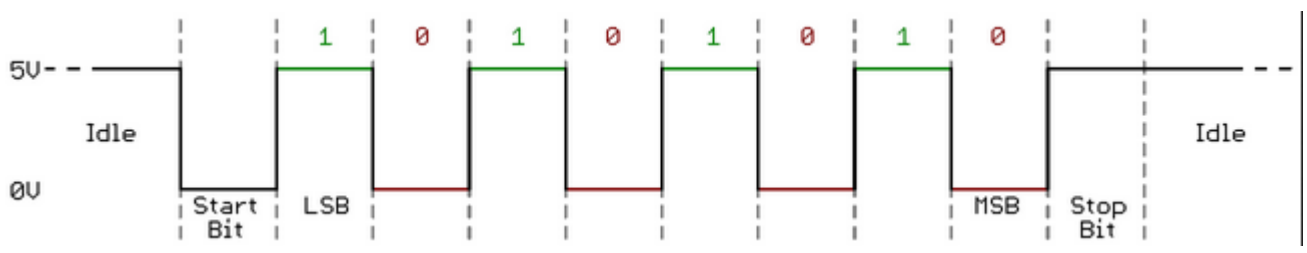
3. Interface série USART

L'interface série est le moyen le plus simple de communiquer avec votre microcontrôleur pour lire des données ou envoyer des commandes. Du point de vue du microcontrôleur, la communication série est basée sur seulement **deux lignes de données** :

- ligne de **transmission** , notée **Tx** ,
- ligne de **réception** , notée **Rx** .

La communication est en duplex intégral, elle peut être transmise simultanément à la réception.

La transmission asynchrone des données se fait au niveau des **trames** , chaque trame étant constituée de plusieurs bits, ayant le format décrit sur la figure.



Un bit de départ est transmis , puis un mot de données. Il est suivi d'un **bit de parité** facultatif , dont le rôle consiste à effectuer une simple vérification de l'exactitude des données, et **d'un ou deux bits d'arrêt** .

Le microcontrôleur ATmega328p comprend un périphérique **USART** (Universal Synchronous-Asynchronous Receiver/Transmitter) pour l'interface série. Dans la partie initialisation de ce périphérique, les étapes suivantes doivent être effectuées :

- choisir la vitesse de transmission des données - le débit en bauds (valeurs usuelles : 9600, 19200, 38400, 57600, 115200)
- choisir le format de la trame (combien de bits de données, de bits d'arrêt, si elle contiendra ou non un bit de parité)
- permettant l'émission et la réception de données sur les lignes RXet TX.

Le débit en bauds est le nombre de symboles/impulsions par seconde du signal. Il représente essentiellement la vitesse de transmission et il est très important que l'émetteur et le récepteur utilisent le même débit en bauds pour une transmission correcte des données. L'un des problèmes les plus courants avec l'USART configuré de manière asynchrone est le réglage différent du débit en bauds sur l'émetteur et le récepteur. Cette incohérence se manifeste par la réception de données erronées (l'émetteur envoie le caractère 'a', le récepteur reçoit le caractère '&')

Pour que deux appareils, dans notre cas le PC et la carte de laboratoire, puissent communiquer entre eux via USART de manière asynchrone, ils doivent être configurés de manière **identique** . Si la carte est configurée avec un débit en bauds de 115200, 9 bits de données, 1 bit d'arrêt et aucune parité, le PC doit être configuré **exactement de la même manière** pour communiquer.

3.1 S'inscrire

Description complète pour :

- les trois registres de contrôle
- registre de débit en bauds
- les tampons d'émission/réception

Il se trouve dans la fiche technique Atmega 328p au chapitre 19. Les registres ont un 'n' à la fin qui distingue plusieurs périphériques USART pouvant exister sur un microcontrôleur (sur l'ATmega328P 'n' ne prendra que la valeur 0 correspondant à USART0).

Registre de données USART n (UDRn)

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

RXBet TXBsont respectivement les tampons de réception et de transmission. Ils utilisent *la même adresse d'E/S* . RXBOn y accède donc en lisant depuis UDRn, TXBen écrivant dans UDRn. Le tampon de transmission ne peut être écrit que lorsque le bit UDRE(*USART Data Register Empty*) du port UCSRnAest à 1. Sinon, les écritures seront ignorées.

Registre de contrôle et d'état USART n A (UCSRnA)

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UCSRnAest le registre d'état du contrôleur de communication. Les éléments les plus importants sont :

- **RXCn** - **Receive Complete** - devient 1 lorsqu'il y a des données reçues et non lues. Lorsque le tampon de réception est vide, le bit est automatiquement réinitialisé
- **TXCn** - **Transmit Complete** - devient 1 lorsque le tampon de transmission devient vide
- **UDREn** - **Registre de données vide** - devient 1 lorsque le tampon de transmission peut accepter de nouvelles données

Registre de contrôle et d'état USART n B (UCSRnB)

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UCSRnBest un registre de contrôle. Éléments importants :

- **RXCIEn** - **Receive Complete Interrupt Enable** - lorsque 1, le contrôleur de communication génère une interruption lorsque les données ont été reçues
- **TXCIEn** - **Transmit Complete Interrupt Enable** - lorsque 1, le contrôleur de communication génère une interruption lorsque le tampon de transmission devient vide
- **UDRIEn** - **Activer l'interruption du registre de données vide** - lorsque 1, le contrôleur de communication génère une interruption lorsque le tampon de transmission ne peut plus accepter de données
- **RXENn** - **Receiver Enable** - si 0, aucune donnée ne peut être reçue
- **TXENn** - **Transmitter Enabler** - si 0, aucune donnée ne peut être transmise
- **UCSZn2** - avec UCSZ1et UCSZ0depuis le port UCSRC, sélectionne la taille d'un mot de données

Registre de contrôle et d'état USART n C (UCSRnC)

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

UCSRnCc'est aussi un registre de contrôle. Éléments importants :

- **UMSELn** - **Mode Select** - 0 pour un fonctionnement asynchrone, 1 pour un fonctionnement synchrone
- **UPMn1, UPMn0** - **Mode de parité** - Étant deux bits, ils peuvent avoir ensemble 4 valeurs possibles, détaillées dans le tableau suivant :

UPMn1	UPMn0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **USBSn** - **Stop Bit Select** - 0 pour un bit d'arrêt, 1 pour deux bits d'arrêt

USBSn	Stop bit(s)
0	1-bit
1	2-bit

- **UCSZn1, UCSZn0** – avec UCSZn2from port UCSRnB, sélectionnez la taille du mot de données

UCSZn2	UCSZn1	UCSZn0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Registres de débit en bauds USART (UBRRn)

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRnH
	UBRR[7:0]								UBRRnL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

UBRRnest le registre qui sélectionne le **débit en bauds** . Il a 12 bits. Les 4 premiers sont en UBRRnH, les 8 autres en UBRRnL. La valeur que nous écrivons UBRRndépend de la fréquence du processeur et du débit en bauds souhaité. Dans le tableau suivant, vous trouverez les valeurs des vitesses de transmission les plus courantes pour une fréquence d'horloge de 16 MHz .

Baud Rate (bps)	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	–0.1%	832	0.0%
4800	207	0.2%	416	–0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	–0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	–0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	–0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	–3.5%	16	2.1%
230.4k	3	8.5%	8	–3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max.(1)	1Mbps		2Mbps	

Note: 1. UBRRn = 0, error = 0.0%

Il est souhaitable de choisir un débit en bauds qui puisse être obtenu exactement à partir de la fréquence d'horloge. Dans le cas contraire, une tolérance est définie (l'erreur maximale du débit en bauds) pour laquelle la communication peut se faire dans des conditions acceptables. Si vous souhaitez approfondir le sujet, vous trouverez de nombreuses informations ici [https://www.allaboutcircuits.com/technical-articles/the-uart-baud-rate-clock-how-accurate-does-it-need-to-be/]

3.2 Exemple d'utilisation

```
void USART0_init ( )
{
  /* définit le débit en bauds sur 9600 */
  UBRR0 =  103 ;

  /* démarrer l'émetteur */
  UCSR0B =  ( 1  << TXEN0 )  |  ( 1  << RXEN0 ) ;

  /* définit le format de trame : 8 bits de données, 1 bit d'arrêt, pas de parité */
  UCSR0C &= ~ ( 1  << USBS0 ) ;
  UCSR0C |=  ( 3  << UCSZ00 ) ;
}

void USART0_transmit ( données char non signées  ) { /* attendre que le tampon soit vide * / while ( ! ( UCSR0A & ( 1 << UDRE0 ) ) ;



    /* place les données dans le tampon ; la transmission démarrera automatiquement après l'écriture */
    UDR0 = data ;
}

char USART0_receive ( )
{
  /* attend tant que le tampon est vide */
  while  ( ! ( UCSR0A &  ( 1  << RXC0 ) ) ) ;

  /* renvoie les données du tampon */
  return UDR0 ;
}
```

écritures 16 bits

(3 <<x)

Pour les bits de configuration qui se retrouvent toujours les uns après les autres on utilise aussi un masque avec plusieurs bits décalés de l'index le plus à droite : `(3 << UCSZ00)`remplacer donc`(1 << UCSZ01) | (1 << UCSZ00)`

(1 <<x) | (1 << y) La plupart du temps, nous ferons des masques composites, que nous appliquerons à un registre d'E/S en même temps. **Minutieux!** Je ne peux composer que des masques pour la même opération, je ne peux pas appliquer un masque *OU* en même temps qu'un masque *ET* car le résultat serait complètement faux !

3.3 Utilisation de l'interface série sur l'Arduino UNO

L'Arduino UNO se connecte au PC via l'interface série, mais utilise un convertisseur USB-UART intégré sur la carte. Grâce à cette interface et en utilisant l'IDE dédié à Arduino, le microcontrôleur peut être programmé, mais un canal de débogage peut également être fourni. Ainsi, grâce à de simples messages, l'état du système peut être trouvé, les valeurs des variables peuvent être affichées, ou même des commandes peuvent être envoyées, l'interface fonctionnant de manière bidirectionnelle. Plus de détails peuvent être trouvés ici [https://www.arduino.cc/reference/en/language/functions/communication/serial/]

La configuration par défaut de l'interface série USART utilise 8 bits de données, un bit d'arrêt, sans parité (8N1).

Le programme suivant peut envoyer des messages de l'Arduino au PC, via USB (ou en utilisant l'émulateur de Tinkercad)

```
void setup()
{
  Série.begin(9600);
  Serial.println("dans la configuration de la fonction");
}

boucle vide ()
{
  Serial.println("dans la boucle de fonction");
  retard(1000);
}
```

Le programme suivant peut recevoir des messages envoyés par le PC, via USB (ou en utilisant l'émulateur de Tinkercad)

```
void setup()
{
  Série.begin(9600);
  Serial.println("en attente de commandes");
}

boucle vide ()
{
  si (série. disponible ()) {
    char a = Serial.read();
    char buf[20] ;
    sprintf(buf, "%s: %c", "caractère primit", a);
    Serial.println(buf);
  }
}
```

4. Exercices

Tâche 1 (3p)

À l'aide de l'interface série et de la bibliothèque Arduino, envoyez les commandes suivantes à partir d'un terminal série :

- "on" - allume une led
- "off" - piqûre non led
- « clignotement » – clignotement d'une led à un intervalle de 200 ms
- "get" - affiche l'état d'un bouton via l'interface série

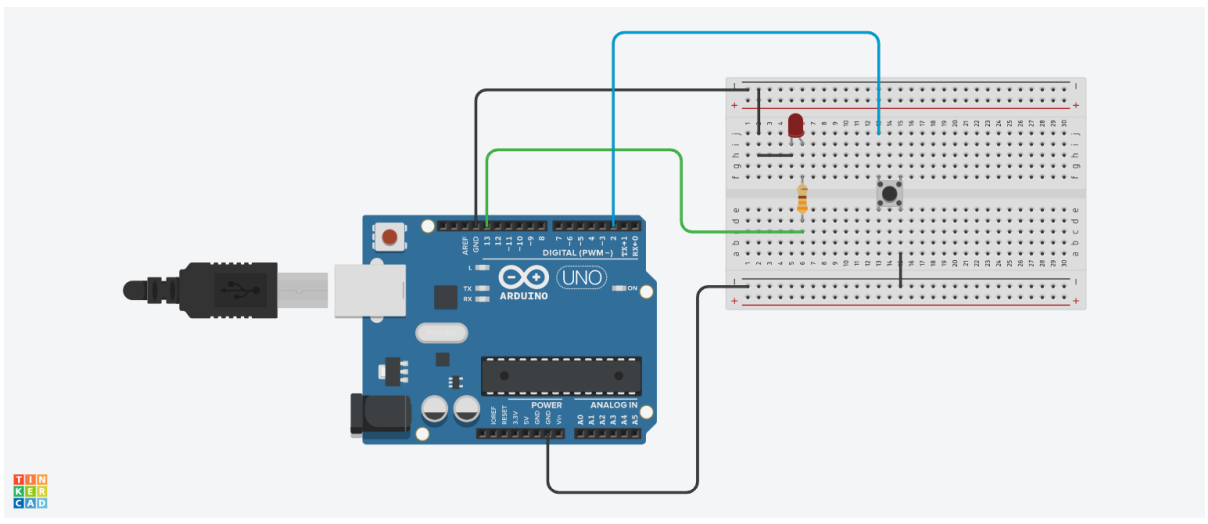
Chaque commande sera suivie du caractère de contrôle : "\n" (nouvelle ligne). Configurez le terminal série pour envoyer (uniquement) ce caractère après chaque message. Implémentez un programme dans Arduino qui reconnaît ces commandes et effectue les actions appropriées pour chacune d'entre elles. Utilisez la broche 13 (PB5) pour la LED et la broche 2 (PD2) pour le bouton.

- La commande de clignotement lancera la fonction de clignotement (clignotement de la led "à l'infini")
- Les commandes marche/arrêt arrêteront la fonction de clignotement.

- pour recevoir une chaîne de l'interface série, vous pouvez utiliser un vecteur de caractères comme tampon
- pour vérifier si la chaîne reçue correspond à la commande, vous pouvez utiliser la fonction *strcmp()* , par ex. *strcmp(a, b) == 0*

Pour pouvoir recevoir des commandes série, il est souhaitable qu'il n'y ait pas de retards ou de fonctions de blocage dans le programme principal (dans la boucle). La fonction de clignotement doit être non bloquante. Nous utiliserons la fonction *millis()* pour mesurer la durée (au lieu de maintenir le programme jusqu'à l'expiration du délai) comme ceci :

```
long ts = 0 ; // la variable globale est votre amie
boucle vide() {
 // mon autre code ici
 si ((millis() - ts) >= 100) {
    ts = millis();
    // ma boucle temporisée non bloquante ici
 }
 // mon autre code ici
}
```



Tâche 2 (3p)

Exécutez l'exemple pour l'USART basé sur le registre. Pour les configurations en série, suivez le fichier usart.c dans le squelette.

Squelette

Capitole utile din Datasheet Atmega 328p

- 1. Configurations des broches - pag. 3
- 19. USART - pag. 143
  - sections 19.1 - 19.2 pour un aperçu
  - section 19.3 pour la génération d'horloge
  - section 19.4 pour le format de trame
  - sections 19.5 - 19.8 pour des exemples de code - initialisation et fonctionnement
  - la section 19.10 est la référence pour les registres d'E/S

Configurez USART0 avec les paramètres suivants : débit en bauds 19200, 8 bits de données, 1 bit d'arrêt, pas de parité. Envoyer un message au PC pour chaque événement d'appui/relâchement de bouton (ex. : PD2 est appuyé, « PD2 appuyé » est transmis, PD2 est relâché, « PD2 relâché » est transmis, une fois par appui).

Le terminal série de l'Arduino est configuré par défaut en mode 8N1, seul le débit en bauds étant paramétrable. Pour avoir un contrôle plus précis sur le format, un terminal série dédié tel que Putty [https://www.putty.org/] peut être utilisé .

Jusqu'à présent, nous avons utilisé l'interface série pour communiquer entre l'Arduino et le PC, via l'adaptateur USB-UART disponible sur la carte. Cependant, il existe des situations dans lesquelles on souhaite connecter un périphérique externe à l'Arduino (par exemple un module de communication radio, bluetooth, GPS, ou généralement un autre microcontrôleur).

Si nous n'avons pas besoin de la connexion PC, nous pouvons connecter un deuxième appareil sur les lignes RX et TX. Cependant, si on voulait déboguer ou envoyer des messages depuis le PC, la connexion ne fonctionne plus correctement, car l'interface série ne permet pas de connecter plus de 2 appareils (au moins sur la ligne TX).

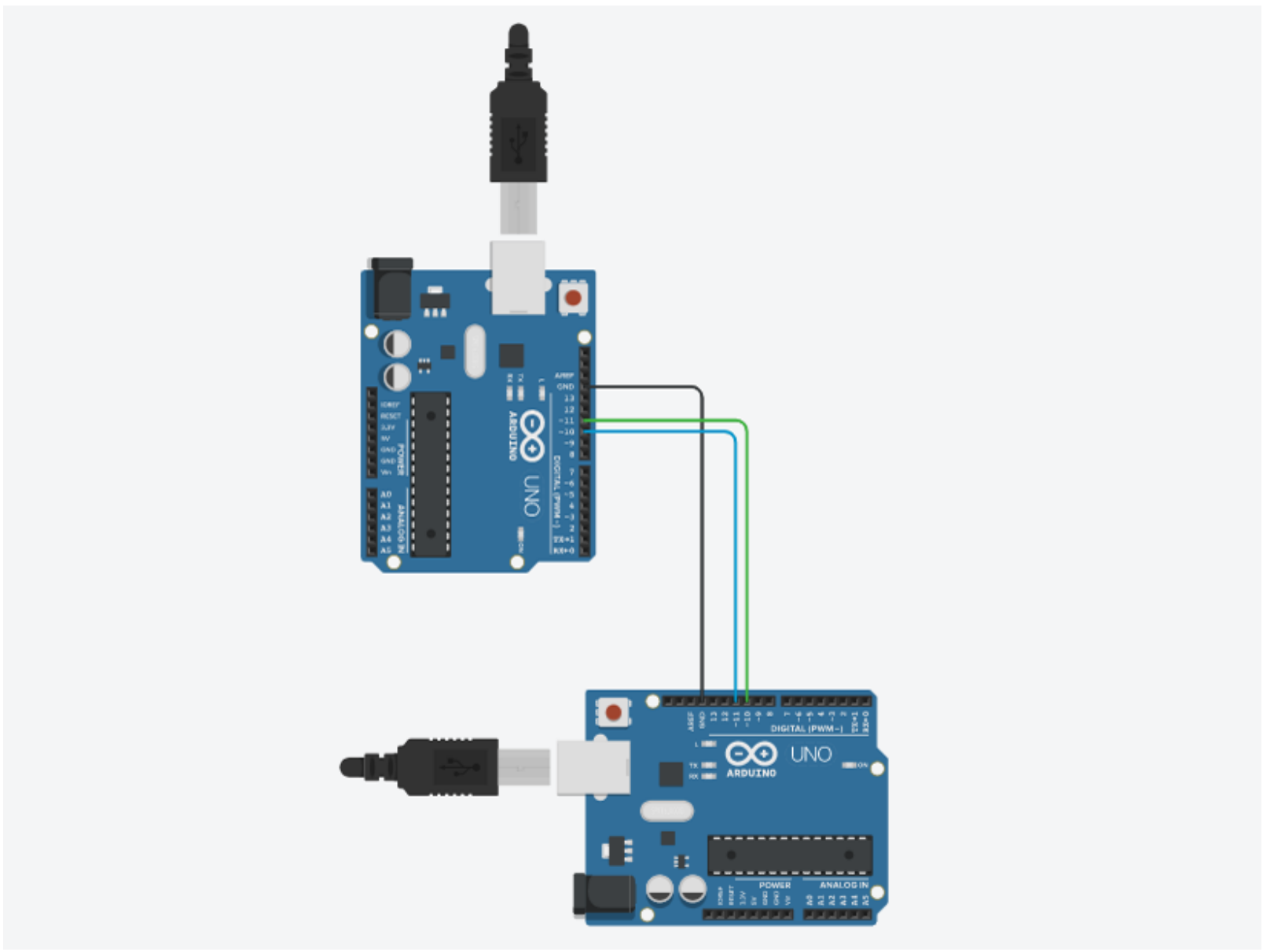
Une autre contrainte est que nous n'avons qu'une seule interface USART sur l'ATmega328p, pour laquelle nous devrons simuler une deuxième interface série dans le logiciel.

Trouvez un exemple dans l'IDE Arduino : Fichiers > Exemples > SoftwareSerial > SoftwareSerialExample

Tâche 3 (4p)

Collaborez avec des collègues à côté de vous pour connecter deux cartes Arduino ensemble et envoyer des messages, du premier Arduino au second, et vice versa.

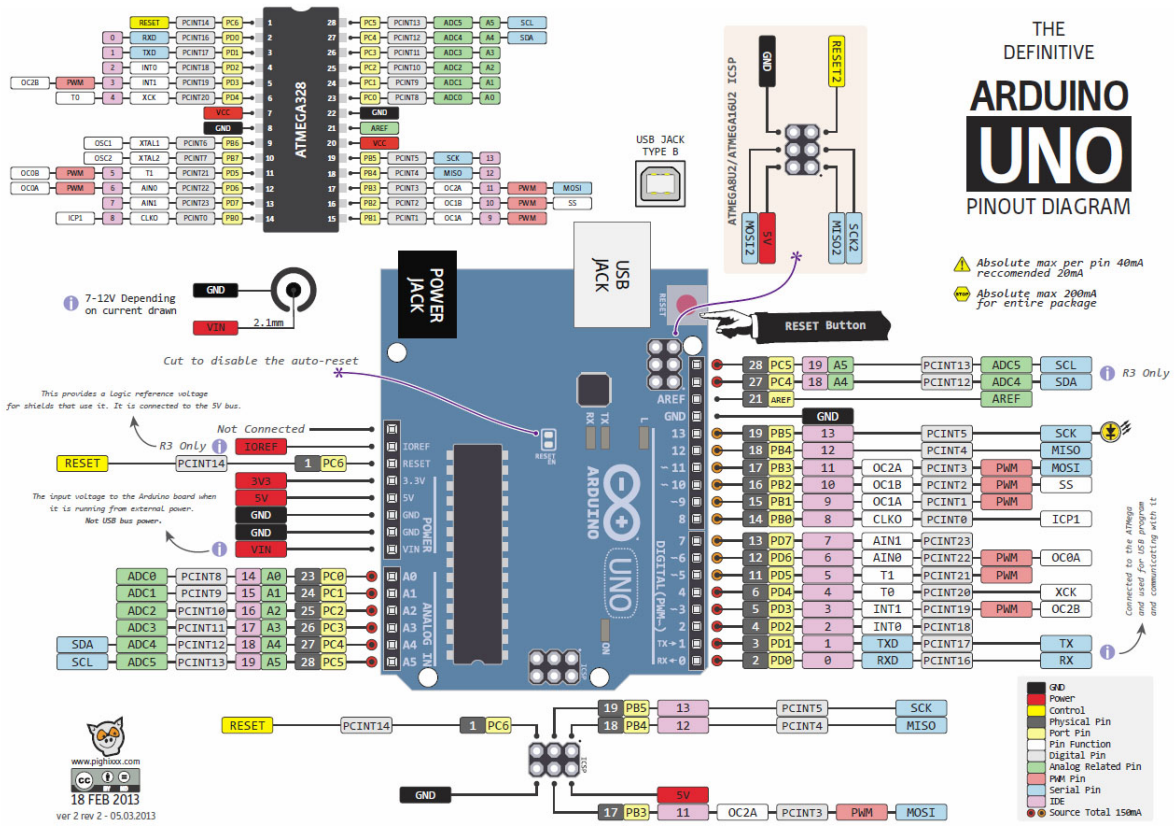
- Les messages envoyés par le premier Arduino seront affichés dans le terminal série connecté au deuxième Arduino, et vice versa.
- Allumer la led (PB5) pendant 500ms lorsqu'un message est reçu. Afin de ne pas bloquer le programme (et implicitement la réception des messages), n'utilisez pas la fonction de retard.
- Faites attention à faire les connexions RX/TX (RX1 à TX2, TX1 à RX2) et GND (GND1 à GND2, si les cartes Arduino sont connectées à des PC différents)
- Dans le cas où les fils ne sont pas assez longs pour connecter directement les deux Arduinos (connectés en même temps à 2 PC via des câbles USB), utilisez les planches à pain et effectuez les connexions via plus de fils.



Bonus (2p) Effectuez l'implémentation basée sur le registre de la tâche 1.

5. Ressources

- Fiche technique Atmega 328p
- Brochage Arduino UNO



- Responsable : Alexandru Predescu [mailto:alexandru.predescu@upb.ro]

RÉSOUDRE