Livrets B4x

B4x Premiers pas

 Edition 1.1

Dernière mise à jour : 2017.10.12

5 Premiers pas avec B4R

B4R – Le moyen le plus simple pour développer des applications natives et puissantes pour des microcontrôleurs Arduino.

B4R suit les mêmes concepts que les autres outils B4X (B4A, B4i, B4J), fournissant un outil de développement simple et puissant.

Les applications compilées fonctionnent sur des circuits compatibles Arduino.

5.1 Installation de l'EDI Arduino

B4R nécessite l'installation de l'EDI Arduino version 1.6.7+.

Téléchargez l'EDI Arduino depuis ce lien : https://www.arduino.cc/en/Main/Software et installez-le.

N'installez pas Arduino 1.6.12 car il a un défaut.

Ne téléchargez pas l'EDI Arduino depuis ce site : http://www.arduino.org/downloads
Il ne fonctionne pas.

5.2 Installation de Microsoft .Net Framework

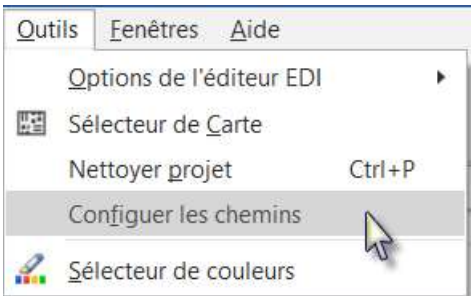
Sur la majorité des ordinateurs Microsoft .Net Framework est déjà préinstallé.

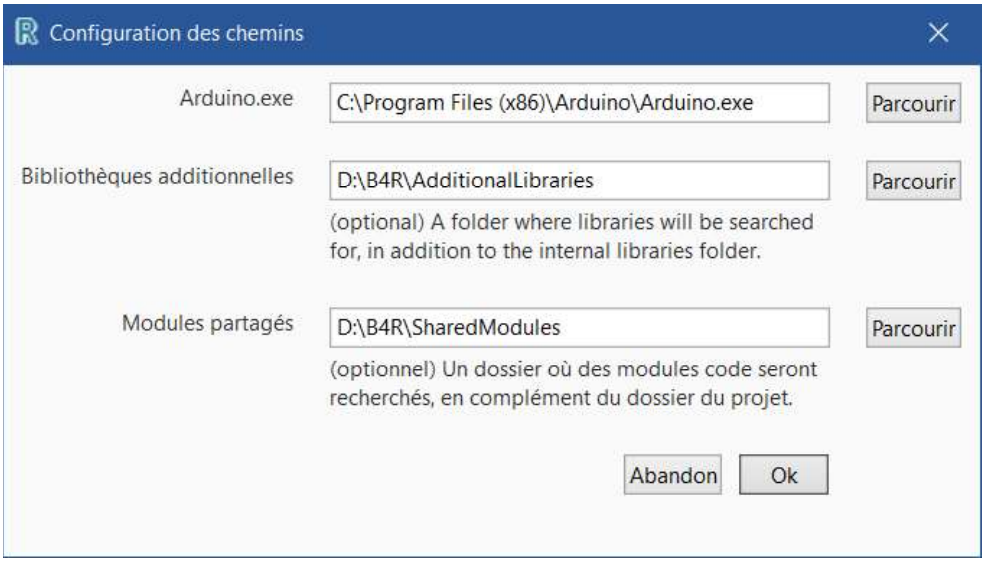
Si ce n'est pas le cas, vous devez installer soit :

- .Net Framework 4.5.2 pour Windows Vista +
- .Net Framework 4.0 pour Windows XP

5.3 Installation et configuration de B4R

- Téléchargez et installez B4R.
- Lancez B4R.
- Cliquez dans le menu Outils sur Configuer les chemins.





Utilisez les bouton Parcourir pour localiser le dossier contenant le fichier "arduino.exe".
Le chemin dépend de l'endroit où vous avez installé l'EDI Arduino.

Il est recommandé de créer un dossier spécifique pour les bibliothèques additionnelles.
B4R utilise deux types de bibliothèques :

- Bibliothèques standard, qui sont fournies avec B4R et se trouvent dans le dossier Libraries de B4R.
  Ces bibliothèques sont automatiquement mises à jour lorsque vous installez une nouvelle version de B4R.
- Bibliothèques additionnelles, qui ne font pas partie de B4R, et doivent être enregistrées dans un dossier spécifique différent de celui de B4R.
  Plus de détails dans le chapitre *Bibliothèques additionnelles* du livret *B4x Langage Basic*.

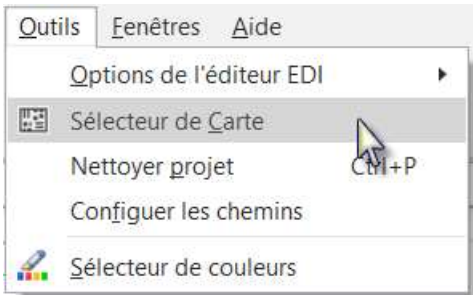
Vous devriez aussi créer un dossier spécifique pour les modules partagés (Shared Modules).
Un même module code peut être utilise par différents projets sans avoir à l'enregistrer dans le dossier du projet.

5.4 Connection d'un circuit

Lorsque vous connectez un circuit au PC avec un câble USB, Windows charge le pilote et affiche le port série utilisé.

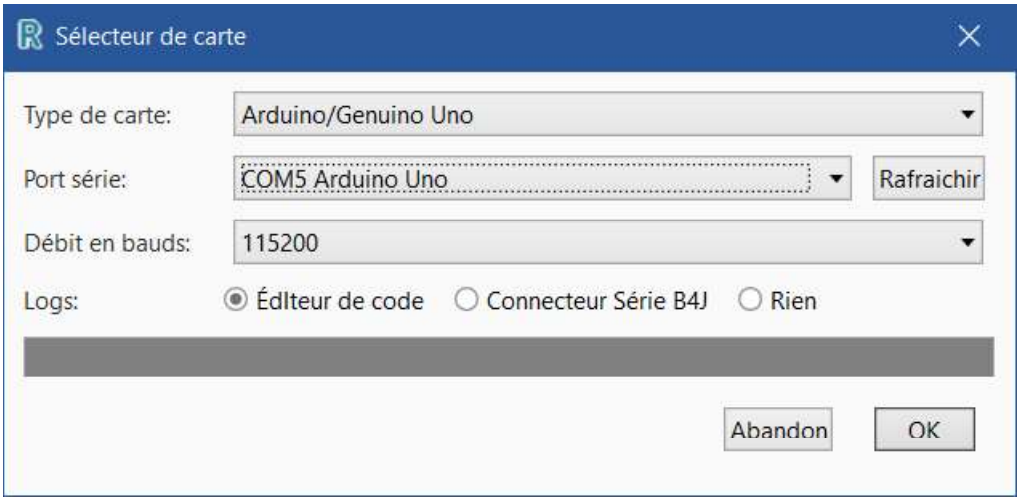
5.5 Sélection d'un circuit

Lancez B4R.



Dans le menu Outils, cliquez sur Sélecteur de Carte.

Une fenêtre similaire à celle ci-dessous sera affichée.

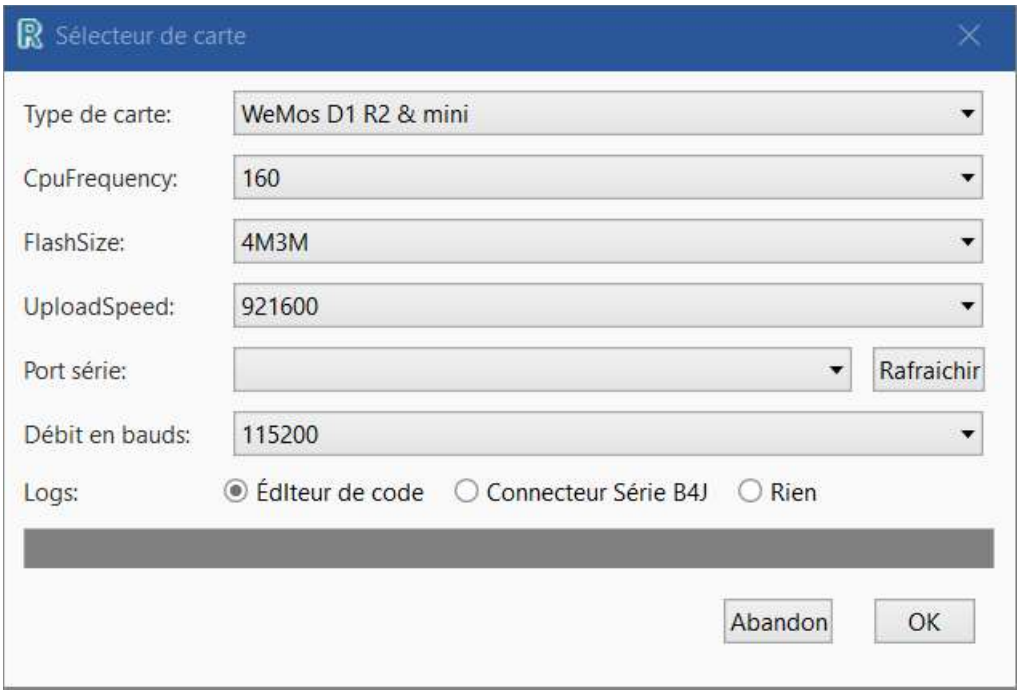


Sélectionnez le type de carte dans la liste déroulante.
Sélectionnez le port série et le débit en bauds.

Si un seul circuit est connecté il sera reconnu automatiquement.

Seul les circuits connectés sont affichés.

Selon le type de circuit d'autres paramètres peuvent être définis.



5.6 Le circuit Arduino UNO

Dans ce chapitre vous trouvez des explications sur quelques fonctions de base de Arduino UNO qui peuvent être utiles aux débutants.
Le circuit Arduino UNO est le circuit de base de la famille Arduino.

Il existe d'autres modules plus avancés.

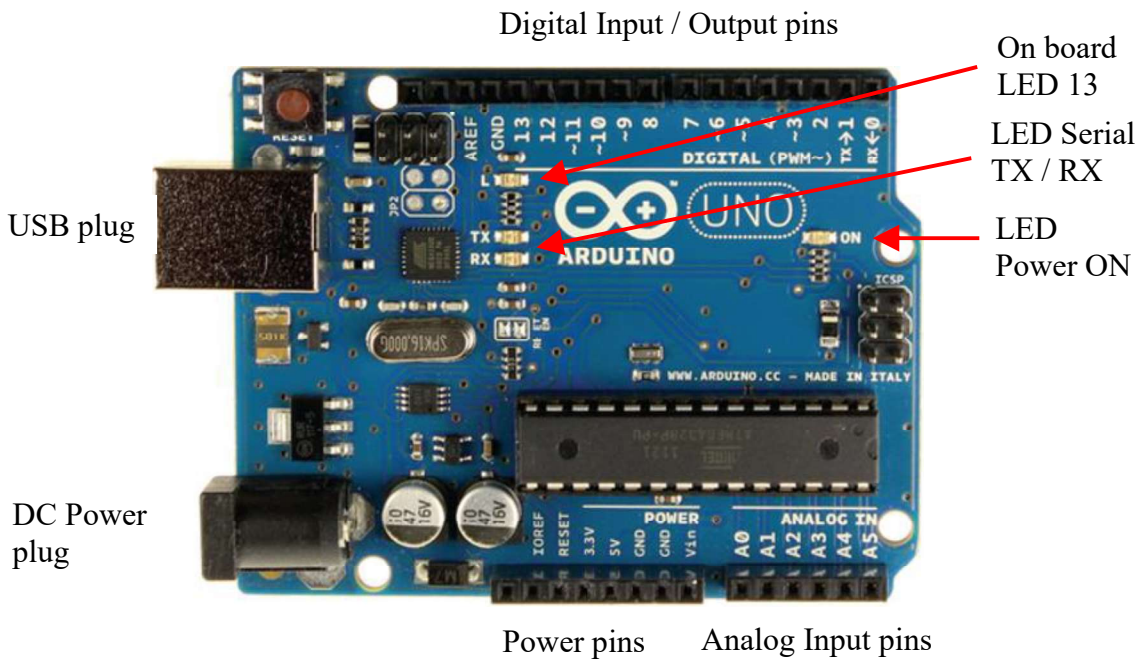
- Arduino DUE
- Arduino MEGA
- Arduino MICRO
- etc. voir Compare board specs.

Des circuits additionnels, appelés 'Shields' peuvent être clipés sur les circuits Arduino.

- Arduino Wi-Fi Shield 101
- Arduino Ethernet Shield
- etc.

L'Arduino UNO :

La source des informations dans ce chapitre est un résumé du site Arduino.



Digital Input / Output pins	Broches entrée / sortie digitales.
USB plug	Connecteur USB.
DC Power plug	Connecteur alimentation DC.
On board LED 13	LED 13 sur le circuit.
Serial TX / RX	Signaux série TX / RX.
LED Power ON	LED alimentation ON.
Power pins	Broches d'alimentation.
Analog Input pins	Broches entrées analogiques.

J'ai intentionnellement gardé les dénominations Anglaises sur le schéma.

5.6.1 Alimentation en courant

Le circuit peut être alimenté en courant soit par le connecteur DC power (7 - 12V), le connecteur USB (5V), ou la broche VIN du circuit (7-12V).
Une alimentation par les broches 5V ou 3.3V by-passe le régulateur, et peut endommager votre circuit (voir Broches ci-dessous). A déconseiller.

5.6.2 Broches

L'Arduino UNO a 3 connecteurs :

- Broches d'alimentation.
- Broches d'entrée / sortie digitaux (Digital Input / Output pins).
- Broches d'entrée analogiques (Analog Input pins).

5.6.3 Broches d'alimentation

Les Broches d'alimentations sont :

- **GND** Masse de l'alimentation, 2 broches (GND ground).
- **VIN** Broche d'alimentation (VIN V entrée).
  Tension d'alimentation lorsque le circuit UNO est alimenté par une source extérieure (par opposition à une alimentation de 5 volts depuis le connecteur USB ou une autre source régulée). Vous pouvez alimenter le circuit par cette broche, ou, si le circuit est alimenté par une source externe par le connecteur (DC Power plug), y accéder par cette broche.
  Tension 7 – 12 V.
- **5V** 5 Volt, tension de référence. **N'alimentez pas cette broche !**
  Cette broche fournit une tension régulée de 5V du régulateur du circuit.
- **3.3V** 3.3 Volt, tension de référence. **N'alimentez pas cette broche !**
  Une source de 3.3 volt générée par le régulateur du circuit. Courant maximum 50 mA.
- **RESET**
  Mettez cette broche sur LOW pour reseter microcontrôleur. Typiquement utilisé pour ajouter un bouton reset à des circuits 'shields' qui bloquent celui du circuit.
- **IOREF**
  Cette broche, sur le circuit UNO, fournit la tension de référence avec laquelle le microcontrôleur fonctionne. Un circuit shield, correctement configure, peut lire la tension de la broche IOREF et sélectionner la source appropriée pour travailler avec 5V ou 3.3V.

5.6.3.1 Broches d'entrée / sortie digitales (Digital Input / Output pins)

Chacune des broches digitales de l'UNO peut être utilisée comme entrée ou sortie, en spécifiant son mode *pinMode*, avec *DigitalRead* ou *DigitalWrite*. Elles opèrent avec 5 volts. Chaque broche peut fournir ou recevoir un courant de 20 mA comme conditions de fonctionnement recommandées et comprend une résistance 'pull-up' interne (déconnectée par défaut) de 20-50 kohm. Une valeur de courant maximale de 40mA ne doit pas être dépassée sur aucune des broches pour éviter un endommagement permanent du microcontrôleur.

En complément, quelques broches ont des fonctions spécialisées :

- Serial: 0 (RX) et 1 (TX). Utilisées pour recevoir (RX) et transmettre (TX) TTL des données série TTL. Ces broches sont reliées aux broches correspondantes du circuit ATmega8U2 USB-to-TTL Serial.
- PWM: ~3, ~5, ~6, ~9, ~10, et ~11. Ces numéros ont "~" pour préfixe.
  Elles peuvent fournir des signaux de sortie en PWM (**P**ulse **W**idth **M**odulation) (modulation de largeur d'impulsions) avec la fonction *AnalogWrite* permettant la modulation de la luminosité de LEDs (**L**ight **E**mitting **D**iode) (DELs Diodes Electro-Luminescentes) ou alimenter un moteur CC à différentes vitesses.
  Une valeur de 0 correspond à OFF et 255 correspond à toujours ON.
  Utilisation:
  pinTest3.AnalogWrite(Value As UInt)
  ```
  pinTest3.AnalogWrite(196)
  ```
  Après *AnalogWrite,* la broche va générer un signal rectangulaire constant avec un rapport cyclique spécifié jusqu'à l'appel suivant de *AnalogWrite* (ou un appel à *DigitalRead* ou *DigitalWrite* sur la même broche). La fréquence du signal PWM est approximativement de 490 Hz sur la plupart des broches. Sur l'UNO et les circuits similaires, les broches 5 et 6 ont une fréquence d'approximativement 980 Hz. Les broches 3 et 11 du circuit Leonardo fournissent également un signal à 980 Hz.
- LED 13 : Il y a une LED sur le circuit gérée par la broche 13. Lorsque le signal sur la broche est HAUT, la LED est allumée, si le signal est BAS elle est éteinte.

5.6.3.2 Broches d'entrée analogiques

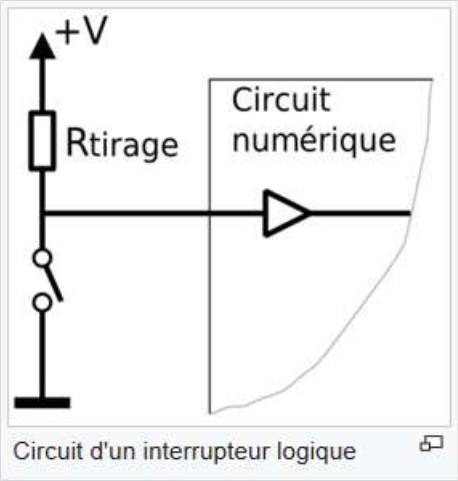
L'Arduino UNO contient 6 broches d'entée analogiques (Analog input pins) A0 à A5 avec un convertisseur analogique vers numérique de 10 bit, donc une résolution de 0 à 1023.
La tension de référence est de 5 Volt donnant une résolution de 4.9 mV par unité.
Alors que la fonction principale des broches analogiques pour la plupart des utilisateurs d'Arduino est de lire des capteurs analogiques, les broches analogiques ont aussi toutes les fonctionnalités des broches numériques 0 - 13.

5.6.4 Modes d'entrée INPUT / INPUT_PULLUP

Si vous avez configuré une broche avec le mode INPUT, et que vous lisez l'état d'un interrupteur, l'état de la broche sera "flottant" lorsque l'interrupteur est ouvert, donnant des résultats imprévisibles. Pour assurer une lecture correcte lorsque l'interrupteur est ouvert, une résistance de rappel ('pull-up') doit être ajoutée. La fonction de cette résistance est de mettre la broche dans un état connu lorsque l'interrupteur est ouvert. Une résistance de 10 K ohm est généralement utilisée, c'est une valeur suffisamment basse pour éviter un point flottant et en même temps une valeur suffisamment haute pour ne pas soutirer un courant trop élevé lorsque l'interrupteur est fermé.

Le mode INPUT_PULLUP ajoute une résistance de rappel ('pull up') interne pour éviter de devoir en ajouter une en externe.

Avec une résistance de rappel, la broche renvoie False lorsque l'interrupteur est fermé car elle est mise à 0 Volt.



Circuit d'un interrupteur logique

Source Wikipedia

5.6.5 Fonctions de base des broches

5.6.5.1 Initialize

Initialise une broche.

Pin.Initialize(Pin As Byte, Mode As Byte)

Pin est le numéro de la broche.
- 0, 1, 2, 3 etc. pour les broches digitales
- Pin.A0, Pin.A1 , Pin.A2 etc. pour les roches analogiques.

Mode est un des trois modes de connexion :
- MODE_INPUT
- MODE_INPUT_PULLUP ajoute une résistance 'pull up' interne.
- MODE_OUTPUT

Exemple1 : Initialisation de la broche digitale 3 en entrée.
```
Private pinTest1 As Pin
pinTest1.Initialize(3, pinTest1.MODE_INPUT)
```

Exemple2 : Initialisation de la broche digitale 3 en entrée avec une résistance 'pull up'.
```
Private pinTest2 As Pin
pinTest2.Initialize(3, pinTest2.MODE_INPUT_PULLUP)
```

Exemple3 : Initialisation de la broche digitale 3 en sortie.
```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

Exemple4 : Initialisation de la broche analogique 4 en entrée.
```
Private pinTest4 As Pin
pinTest4.Initialize(pinTest4.A4, pinTest4.MODE_INPUT)
```

Les broches analogiques, sur l'Arduino UNO, peuvent aussi être référencées par des numéros, comme ci-dessous :
```
pinTest4.Initialize(18, pinTest4.MODE_INPUT)
```

Pin.A0 = 14
Pin.A1 = 15
Pin.A2 = 16
Pin.A3 = 17
Pin.A4 = 18
Pin.A5 = 19

L'initialisation d'une broche analogique en sortie fonctionne comme une broche digitale en sortie.

5.6.5.2 DigitalRead

DigitalRead lit la valeur digitale actuelle d'une broche.
Les valeurs renvoyées sont True ou False.

Pin.DigitalRead renvoie une valeur du type Boolean.

Il y a deux modes dépendant du type de signal d'entrée.

- Pin.MODE_INPUT
- Pin. MODE_INPUT_PULLUP ajoute une résistance 'Pulp' à utiliser avec in interrupteur.

Exemple :

```
Private pinTest1 As Pin
pinTest1.Initialize(3, pinTest.MODE_INPUT)

Private Value As Boolean
Value = pinTest1.DigitalRead
```

L'Arduino utilise en interne 0 et 1 pour les valeurs booléennes.
`Log("State: ", Value)`
Va afficher soit 0 pour False ou 1 pour True dans les Logs.
Dans le code vous pouvez utiliser False et True.

5.6.5.3 DigitalWrite

DigitalWrite écrit une valeur booléenne sur la broche spécifiée.
Peut-être utilise sur toutes les broches digitales comme analogiques.

Pin.DigitalWrite (Value As Boolean)

Exemple:

```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)

pinTest3.DigitalWrite(True)  directement avec une valeur.
pinTest3.DigitalWrite(Value)  avec une variable.
```

5.6.5.4 AnalogRead

AnalogRead lit la valeur actuelle sur une broche analogique.
La valeur renvoyée est du type UInt avec des valeurs entre 0 et 1023 (10 bits).
La tension de référence est 5V.

Exemple :

```
Private pinPot As Pin
pinPot.Initialize(pinPot.A4, pinPot.MODE_INPUT)

Private Value As UInt
Value = pinPot.AnalogRead
```

5.6.5.5 AnalogWrite

AnalogWrite attribue un octet (Byte) à la broche définie pour une modulation de largeur d'impulsions.
AnalogWrite n'a rien faire avec les broches analogiques ni avec AnalogRead.

AnalogWrite peut être utilise seulement avec les broches numériques ~3, ~5, ~6, ~9, ~10, et ~11 sur le circuit Arduino UNO, les broches avec le préfixe ~.

Pin.AnalogWrite (Value As UInt)

Exemple : nous utilisons la broche ~3 qui permet la modulation PWM.

```
Private pinTest3 As Pin
pinTest3.Initialize(3, pinTest3.MODE_OUTPUT)
```

`pinTest3.AnalogWrite(145)` directement avec une valeur.
`pinTest3.AnalogWrite(Value)` avec une variable, `Value` doit être une variable du type UInt.

5.7 Premiers programmes

Tous les projets ont été réalisés avec le Arduino Starter Kit.

Les diagrammes de montage ont été réalisés avec le programme Fritzing.

Lorsque nous exécutons B4R nous obtenons le code par défaut ci-dessous.

La `#Region Project Attributes` est normalement réduite; ces attributs sont expliqués dans le chapitre *Entêtes de code Project Attributes / Activity Attributes* dans le livret *B4x EDI*.

```
#Region Project Attributes
   #AutoFlushLogs: True
   #CheckArrayBounds: True
   #StackBufferSize: 300
#End Region

Sub Process_Globals
   'These global variables will be declared once when the application starts.
   'Public variables can be accessed from all modules.
   Public Serial1 As Serial
End Sub

Private Sub AppStart
   Serial1.Initialize(115200)
   Log("AppStart")
End Sub
```

`#AutoFlushLogs:`	AutoFlushLogs: True assure que les Logs sont renvoyés sans problèmes. AutoFlushLogs: False peut provoquer des problèmes.
`#CheckArrayBounds:`	Vérifie les limites des tableaux (arrays) ou pas.
`#StackBufferSize:`	Définit la dimension par défaut du tampon de la pile (Stack buffer size).

`Public Serial1 As Serial`	Définit l'interface sérielle avec l'ordinateur.
`Serial1.Initialize(115200)`	Initialise le port série avec un débit en baud de 115200 Hertz.
`Log("AppStart")`	Affiche `AppStart` dans l'onglet Logs au démarrage du programme. Les Logs sont expiliqués dans le chapitre *Logs* dans le livret *B4x EDI*.

5.7.1 Bouton.b4r

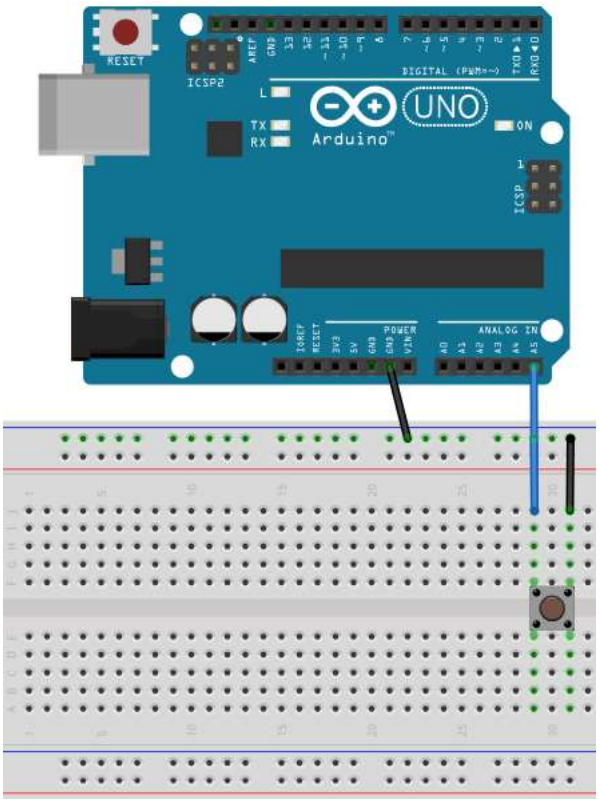
Écrivons le premier programme.

Il est similaire à l'exemple Button d'Erel dans le forum. Nous utilisons un commutateur à bouton poussoir et la LED 13 se trouvant sur la carte de l'Arduino UNO.

Le projet Bouton.b4r est disponible dans le dossier CodesSource\B4R\Bouton.

- Lancez B4R.
- Sauvez le projet sous Bouton dans un dossier avec le nom Bouton.
- Réalisez le montage du circuit avec le bouton et le câblage.
- Connectez l'Arduino au PC avec un câble USB.
- Écrivez le code.
- Exécutez le programme.

5.7.1.1 Schéma



Matériel :
- 1 commutateur à bouton poussoir

Reliez une des broches **GND** (masse / ground) de l'Arduino à une des lignes **GND** du breadboard.
Puis, reliez une des broches du commutateur à la ligne **GND** du breadboard.
Et, reliez l'autre broche du commutateur à la broche analogique **A5** de l'Arduino.

Nous aurions pu relier la première broche du commutateur directement à une des broches **GND** de l'Arduino, mais le montage est déjà prêt pour les exemples suivants.

Nous aurions aussi pu utiliser une des broches digitales au lieu d'une broche analogique.

5.7.1.2 Code

```
Sub Process_Globals
   Public Serial1 As Serial
   Private pinBouton As Pin      'broche (pin) pour le bouton
   Private pinLED13 As Pin       'broche (pin) pour la LED 13 de l'Arduino
End Sub
```

Nous déclarons la broche du commutateur et la LED 13 de l'Arduino.

```
Private Sub AppStart
   Serial1.Initialize(115200)
   Log("AppStart")

   pinBouton.Initialize(pinBouton.A5, pinBouton.MODE_INPUT_PULLUP)
   'Utilisation d'une résistance pull up pour éviter que la broche soit flottante.
   pinBouton.AddListener("pinBouton_StateChanged")

   pinLED13.Initialize(13, pinLED13.MODE_OUTPUT)
End Sub
```

Nous initialisons `pinBouton`, comme broche analogique **A5**, avec `pinBouton.A5` et définissons le mode d'entrée à `pinBouton.MODE_INPUT_PULLUP`. Nous avons besoin d'une résistance 'pull up' pour éviter que la broche ne soit flottante, `MODE_INPUT_PULLUP` connecte une résistance 'pull up' interne.

Nous ajoutons `pinBouton.AddListener("pinBouton_StateChanged")`, pour générer un événement StateChanged lorsque l'état de la broche `pinBouton` change, ce qui veut dire lorsque le bouton est pressé ou relâché.

Nous initialisons `pinLED13`, comme broche digitale 13, la LED 13 interne de l'Arduino, et définissons son mode de sortie `pinLED13.MODE_OUTPUT`.

```
Sub pinBouton_StateChanged (State As Boolean)
   Log("State: ", State)
   'State sera False lorsque le bouton est pressé à cause du mode PULLUP
   pinLED13.DigitalWrite(Not(State))
End Sub
```

Nous ajoutons un Log, `Log("State: ", State)`, pour afficher l'état.

Nous attribuons la valeur de State à la broche digitale LED 13, `pinLED13.DigitalWrite(Not(State))`.

Nous écrivons `Not(State)` car State sera False lorsque le bouton poussoir est pressé à cause du mode PULLUP.

Cliquez sur ▶ ou pressez F5 pour exécuter le code.

Lorsque nous pressons le bouton poussoir, la LED 13 de l'Arduino UNO s'allume et lorsque nous relâchons le bouton poussoir la LED s'éteint.

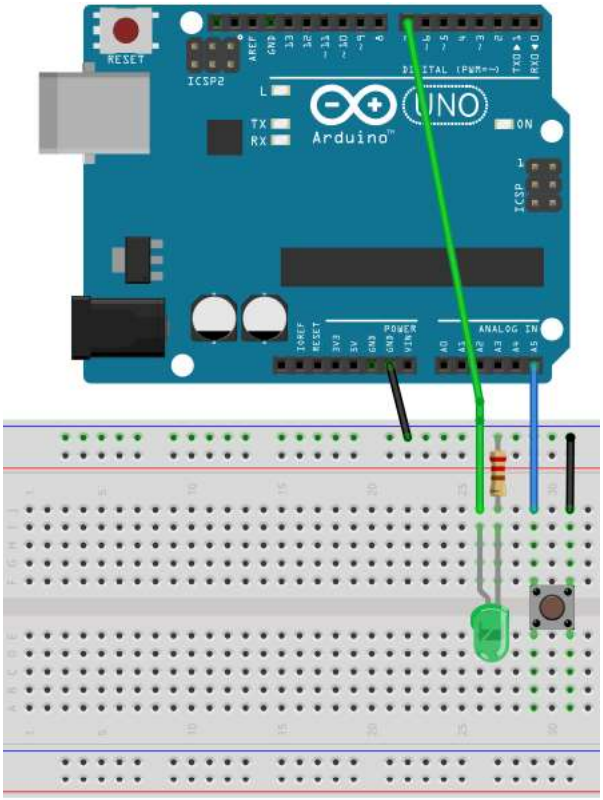
5.7.2 LedVerte.b4r

Pour ce projet nous utilisons une copie du projet Bouton.
Créez un nouveau dossier LedVerte, copiez-y tous les fichiers du projet Bouton et renommez les fichiers Bouton.xxx en LedVerte.xxx.

Nous ajoutons une LED verte à notre circuit qui peut être allumée ou éteinte avec le commutateur à bouton poussoir du premier projet.

Le projet LedVerte.b4r est disponible dans le dossier CodesSource\B4R\LedVerte.

5.7.2.1 Schéma



Matériel :
- 1 commutateur à bouton poussoir
- 1 LED verte
- 1 résistance 220 Ω

Nous gardons le montage du commutateur de premier exemple.
Une broche sur la ligne **GND** du breadboard.
L'autre sur la broche analogique **A5** de l'Arduino.

Et, nous
- Ajoutons une LED verte sur le breadboard.
- Relions la cathode (-) via une résistance de 220 Ω à la ligne **GND** du breadboard.
- Relions l'anode (+) à la broche digitale **7**.

5.7.2.2 Code

```
Sub Process_Globals
   Public Serial1 As Serial
   Private pinBoutton As Pin           'broche (pin) pour le bouton
   Private pinLEDVerte As Pin          'broche (pin) pour la Led verte
   Private Allumé = False As Boolean
End Sub
```

Nous gardons la définition de `pinBouton`.

Nous changeons la définition
`Private pinLED13 As Pin`
en
`Private pinLEDVerte As Pin`
Pour la LED verte.

Nous ajoutons une variable booléenne globale `Allumé` qui est True quand la LED est allumée.

```
Private Sub AppStart
   Serial1.Initialize(115200)

   pinBouton.Initialize(pinButton.A5, pinBouton.MODE_INPUT_PULLUP)
   'Utilisation d'une résistance pull up pour éviter que la broche soit flottante.
   pinBouton.AddListener("pinBouton_StateChanged")

   pinLEDVerte.Initialize(7, pinLEDVerte.MODE_OUTPUT)
End Sub
```

Nous gardons le code pour `pinBouton`.
Nous initialisons `pinLEDVerte` comme broche digitale **7** et définissons le mode de sortie à `pinLEDVerte.MODE_OUTPUT`.

```
Private Sub pinBouton_StateChanged (State As Boolean)
   Log("State: ", State)
   'State sera False lorsque le bouton est pressé à cause du mode PULLUP.
   If State = False Then
      Allumé = Not(Allumé)
      pinLEDVerte.DigitalWrite(Allumé)
   End If
End Sub
```

Chaque fois que la variable State est False, bouton poussoir pressé, nous changeons la variable `Allumé` et l'attribuons à la broche `pinLEDVerte`.

5.7.3 LedVerteSansRebond.b4r

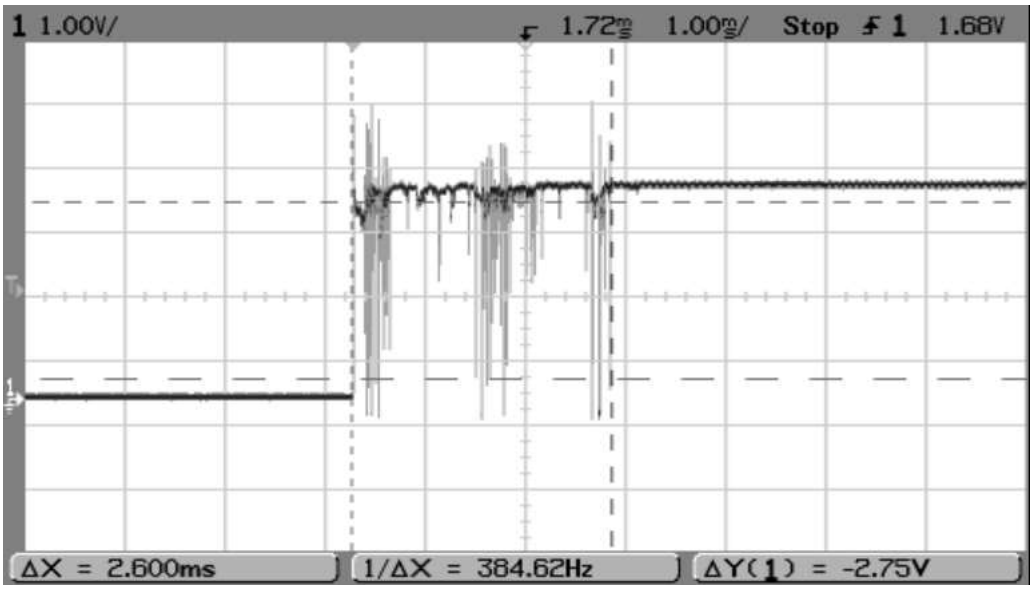
Pour ce projet nous utilisons exactement le même circuit que pour le projet LedVerte.b4r.

La seule différence se trouve dans le code.

Le projet LedVerteSansRebond.b4r est disponible dans le dossier CodesSource\B4R\ LedVerteSansRebond.

Le commutateur à bouton poussoir a un problème appelé rebond.
Le signal d'un commutateur mécanique n'est pas propre, le contact rebondit plusieurs fois ce qui est interprété comme plusieurs changements d'état. Si nous avons un nombre pair de changements d'état c'est comme si nous n'avions rien fait.
Mais nous ne voulons qu'un seul changement d'état par appui sur le bouton.

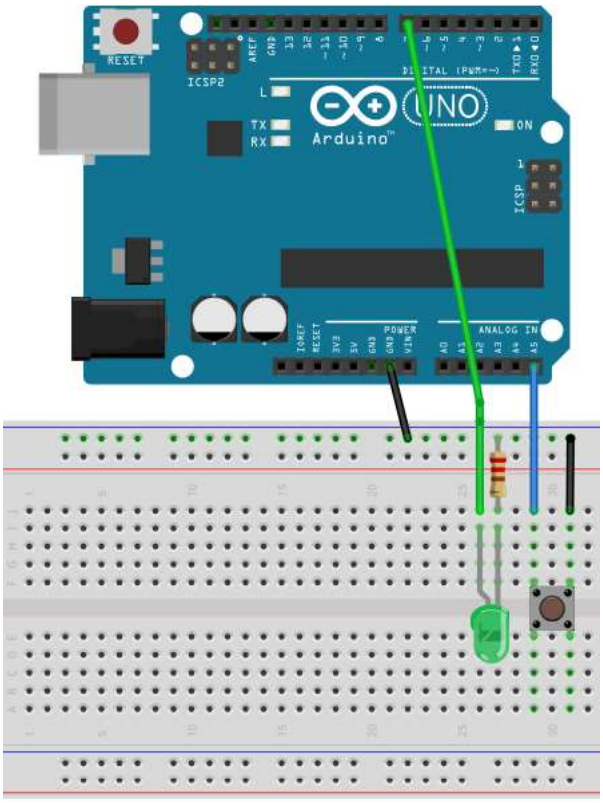
Image de rebonds d'un commutateur (source Wikipedia) :



Le commutateur rebondit plusieurs fois générant plusieurs changements d'état avant de rester dans un état stable.

Pour résoudre ce problème, nous ne réagissons pas, dans la routine `pinButton_StateChanged`, à des changements d'état pendant un temps donné (10 millisecondes).

5.7.3.1 Schéma



Matériel:

- 1 commutateur à bouton poussoir
- 1 LED verte
- 1 résistance 220 Ω

Nous gardons le montage du bouton poussoir du premier exemple.
Une broche liée à la ligne **GND** du circuit.
L'autre à la broche digitale **7** de l'Arduino.

Et nous
- ajoutons une LED verte sur le circuit.
- relions la cathode (-) à la ligne **GND** du circuit via une résistance de 220 Ω.
- relions l'anode (+) to digital pin **7**.

5.7.3.2 Code

Le code est pratiquement le même que LedGreen.b4r.

```
Sub Process_Globals
   Public Serial1 As Serial
   Private pinBouton As Pin            'broche (pin) pour le bouton
   Private pinLEDVerte As Pin          'broche (pin) pour la Led verte
   Private Allumé = False As Boolean
   Private RebondTemps As ULong
   Private RebondRetard = 10 As ULong
End Sub
```

Nous ajoutons deux nouvelles variables : `RebondTemps` et `RebondRetard.`

```
Private Sub AppStart
   Serial1.Initialize(115200)

   pinBouton.Initialize(pinBouton.A5, pinBouton.MODE_INPUT_PULLUP)
   'Utilisation d'une résistance pull up pour éviter que la broche soit flottante.
   pinBouton.AddListener("pinBouton_StateChanged")

   pinLEDVerte.Initialize(7, pinLEDVerte.MODE_OUTPUT)
End Sub
```

Même que LedVerte.b4r

Nouvelle routine `pinButton_StateChanged` :

```
Private Sub pinBouton_StateChanged (State As Boolean)
   Log("State: ", State)
   'State sera False lorsque le bouton est pressé à cause du mode PULLUP.
   If State = False Then
      If Millis - RebondTemps < RebondRetard Then
         Return            'Retour, dans le rebond !
      Else
         Allumé = Not(Allumé)
         pinLEDVerte.DigitalWrite(Allumé)
         RebondTemps = Millis  'remettre RebondTemps au temps actuel
      End If
   End If
End Sub
```

Chaque fois que State est False, bouton poussoir pressé :

- Nous comparons le temps entre le changement d'état actuel par rapport au premier changement d'état.
  Si le temps est plus court que `RebondRetard`, c'est un rebond, nous ne faisons rien.
  Si le temps est plus long que `RebondRetard`, c'est un changement d'état réel, nous exécutons le code.
- Nous modifions la variable `Allumé` et attribuons sa valeur à `pinLEDVerte`.
- Définissons un nouveau `RebondTemps`.

5.7.4 FeuxSignalisation.b4r

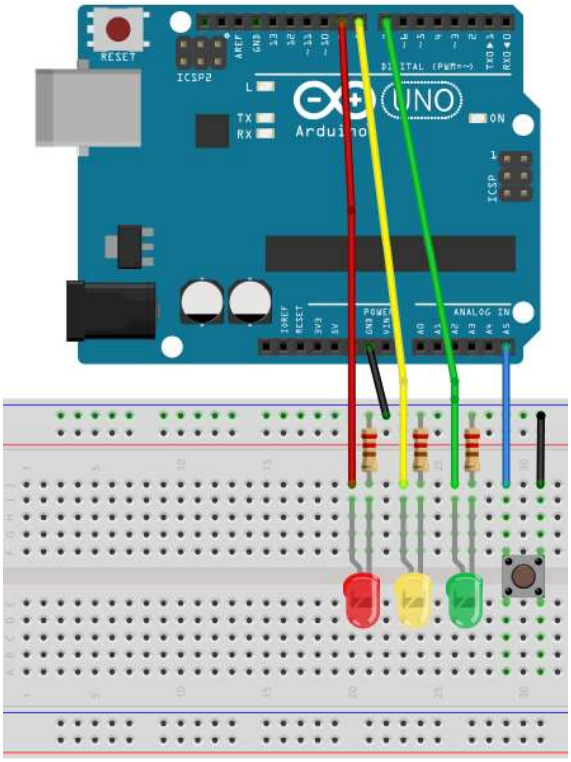
Ce projet est une evolution du projet LedVerteSansRebond et simule des feux de signalisation.

Nous utilisons une copie du projet LedVerteSansRebond.
Créez un nouveau dossier FeuxSignalisation, copiez-y tous les fichiers du projet LedVerteSansRebond et renommez les fichiers *LedVerteSansRebond .xxx* en *FeuxSignalisation.xxx*.

Les feux peuvent être actives ou désactivés avec le commutateur à bouton poussoir comme dans les projets précédents.
Le cycle rouge – vert et vert – rouge est géré par un Timer (minuteur) et le cycle jaune est régi par une routine appellee avec un retard correspondant à la duré de cycle jaune.

Le projet FeuxSignalisation.b4r est disponible dans le dossier CodesSource\B4R\ FeuxSignalisation.

5.7.4.1 Sketch



Matériel:

- 1 commutateur à bouton poussoir
- 1 LED verte
- 1 LED jaune
- 1 LED rouge
- 3 résistances de 220 Ω

Nous
- ajoutons une LED jaune et une rouge similaire à la verte.
- relions le (+) de la LED jaune à la broche digitale **8**.
- relions le (+) de la LED rouge à la broche digitale **9**.

5.7.4.2 Code

```
Sub Process_Globals
   Public Serial1 As Serial

   Public pinButton As Pin                              'broche pour le bouton
   Public pinLEDVert, pinLEDJaune, pinLEDRouge As Pin 'broches pour les LEDs
   Public TimerVertRouge As Timer
   Public FeuxON = False As Boolean
   Public LEDVerte = False As Boolean
   Public RebondTemps  As ULong
   Public RebondRetard  = 10 As ULong
End Sub
```

Nous déclarons le commutateur, les trois broches pour les LEDs, le Timer et qutre variables globales LightOn, LightGreen, RebondTemps et RebondRetard.

FeuxOn = False > Feux OFF
LEDVerte = True > LED verte ON

```
Private Sub AppStart
   Serial1.Initialize(115200)

   TimerVertRouge.Initialize("TimerVertRouge_Tick", 2000)

   'Utilisation de la résistance interne.
   pinButton.Initialize(pinButton.A5, pinButton.MODE_INPUT_PULLUP)
   pinButton.AddListener("pinButton_StateChanged")

   pinLEDVert.Initialize(7, pinLEDVert.MODE_OUTPUT)
   pinLEDJaune.Initialize(8, pinLEDJaune.MODE_OUTPUT)
   pinLEDRouge.Initialize(9, pinLEDRouge.MODE_OUTPUT)
End Sub
```

Nous initialisons TimerVertRouge avec le nom d'événement "TimerVertRouge" et un intervalley de 2000 ce qui signifie que l'événement Tick sera généré toutes les 2 secondes (2000 milli-secondes).
Nous gardons le code pour le bouton et la LED verte, et attribuons la broche digitale 8 en sortie à pinLEDJaune et attribuons la broche digitale 9 en sortie à pinLEDRouge.

Le code est, j'espère, auto-explicatif.

```
Private Sub pinButton_StateChanged (State As Boolean)
   Log("État: ", State)                          'Log la valeur de State (état)

   If State = False Then                         'si State = False
      If Millis - RebondTemps  < RebondRetard  Then
         Return
      Else
         pinLEDRouge.DigitalWrite(True)          'allume la LED rouge
         FeuxON = Not(FeuxON)                    'change la valeur de FeuxON
         RebondTemps  = Millis
         Log("Feux: ", FeuxON)                   'Log la valeur de FeuxON

         TimerVertRouge.Enabled = FeuxON         'active le Timer TimerVertRouge

         If FeuxON = False Then                  'si FeuxON = False
            pinLEDVert.DigitalWrite(False)       'éteint la LED verte
            pinLEDJaune.DigitalWrite(False)      'allume la LED jaune
            pinLEDRouge.DigitalWrite(False)      'éteint la LED rouge
         End If
      End If
   End If
End Sub

Private Sub TimerVertRouge_Tick
   If LEDVerte = True Then                       'si LEDVerte = True
      Log("TimerVertRouge_Tick LEDJaune ON")     'écrit le Log
      CallSubPlus("FinJaune", 500, 0)
      pinLEDVert.DigitalWrite(False)             'switch OFF LED Green
      pinLEDJaune.DigitalWrite(True)             'allume la LED jaune
      LEDVerte = False                           'met LEDVerte à False
   Else
      Log("TimerVertRouge_Tick LEDVerte ON")     'écrit le Log
      pinLEDRouge.DigitalWrite(False)            'éteint la LED rouge
      pinLEDVert.DigitalWrite(True)              'allume la LED verte
      LEDVerte = True                            'met LEDVerte à True
   End If
End Sub

Private Sub FinJaune(Tag As Byte)
   Log("LEDRouge ON")                            'écrit le Log
   Log(" ")
   pinLEDJaune.DigitalWrite(False)               'éteint la LED jaune
   pinLEDRouge.DigitalWrite(True)                'allume la LED rouge
End Sub
```

Nous utilisons la routine FinJaune pour passer du feu jaune au feu rouge.
Cette routine est appellee avec le mot clé CallSubPlus qui permet d'appeler la routine définie avec un retard donné.

```
CallSubPlus("FinJaune", 500, 0)
```

`FinJaune` = Nom de le routine
`500` = retard, 0.5 seconde (500 milli-secondes)
`0` = Tag, pas utilize dans notre cas.

Nous pouvons activer ou désactiver les feux avec le bouton commutateur.

5.8 Glossaire

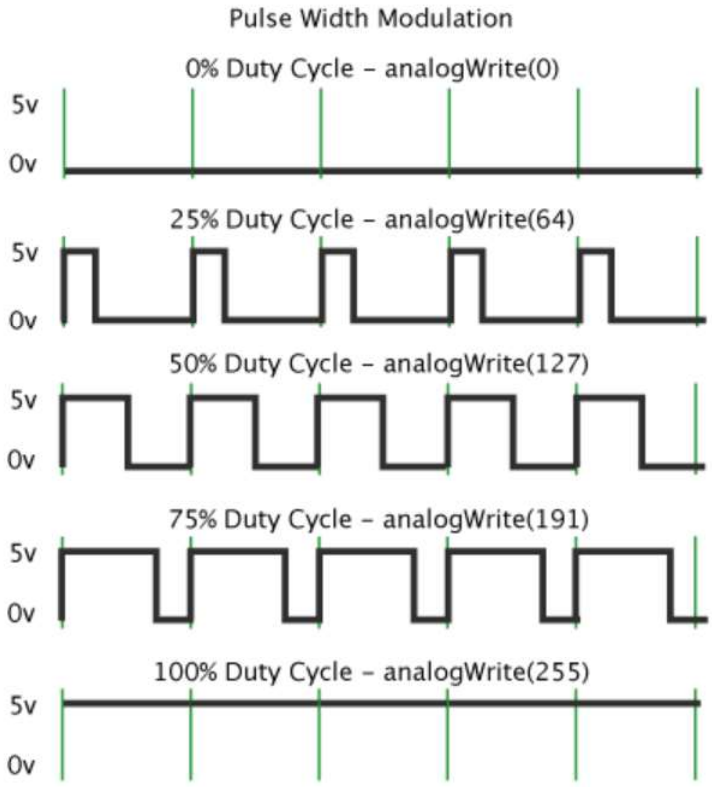
5.8.1 Bases d'électricité

Bases d'électricité (en anglais) : Electricity Basics.

5.8.2 PWM Pulse Width Modulation

Pulse Width Modulation, ou PWM (modulation de largeur d'impulsions), est une technique pour obtenir des résultats analogiques avec des moyens numériques. Un contrôle numérique est utilisé pour créer un signal rectangulaire, un signal qui est commute entre ON et OFF. Ce genre de commutations peut simuler des tensions entre 5 Volt (toujours ON) et 0 Volt (toujours OFF) en modifiant la durée pendant laquelle la tension est à 5 V par rapport à la durée pendant laquelle la tension est à 0 V, appelé le 'rapport cyclique'. La durée pendant laquelle le signal est à 5 V est appelée largeur d'impulsion (pulse width). Pour obtenir des valeurs analogiques variables, on modifie ou module cette largeur d'impulsion. Si on répète ce schéma de commutations suffisamment rapidement, avec une LED par exemple, le résultat est comme si on avait une tension stable entre 0 et 5V modulant la luminosité de la LED.

Dans le graphique ci-dessous, les lignes vertes représentent une périodicité régulière dans le temps. Cette durée ou période est l'inverse de la fréquence de modulation. En d'autres termes, avec une fréquence de modulation d'environ 500Hz, (celle des cartes Arduino), les lignes vertes auraient un espacement de 2 millisecondes. La fonction analogWrite() est basée sur 0 - 255, donc analogWrite(255) fournit un rapport cyclique de 100% (toujours ON), analogWrite(127) est à 50% de rapport cyclique (moitié du temps) et analogWrite(0) 0% donc toujours OFF.



0% de rapport cyclique

25% de rapport cyclique

50% de rapport cyclique

75% de rapport cyclique

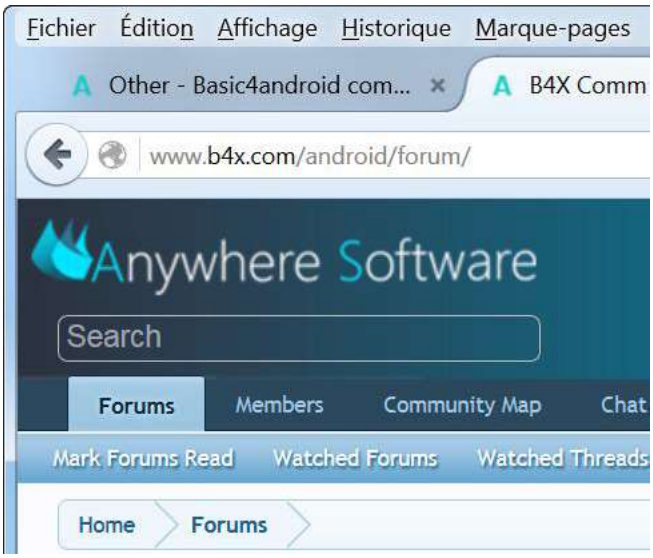
100% de rapport cyclique

Source Android Site Tutorials.

6 Outils d'aide

Les outils suivants sont utiles pour trouver des réponses à beaucoup de vos questions.

6.1 Fonction recherche dans le forum / Search

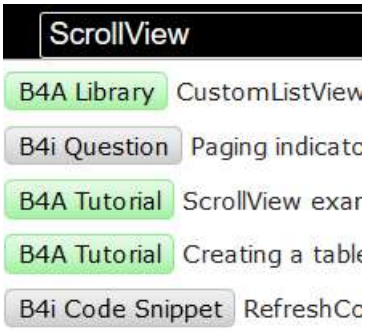


Dans le coin supérieur gauche vous trouvez le champ de recherche 'Search' pour le forum. Selon la largeur de la fenêtre, le champ 'Search' peut se trouver dans le coin supérieur droit.

Entrez une question ou un mot clé puis pressez 'Entrée'.

La fonction montre les posts qui correspondent à votre requête.

Exemple : Entrez le mot clé ScrollView :



Une liste de résultats est affichée juste en dessous du champ de recherche.

Cliquez sur un élément de la liste pour afficher le 'post' complet.

6.4.4 B4R

Page d'accueil Arduino :

https://www.arduino.cc/en/Guide/HomePage