

Brouillon

Table des matières

<u>Avant propos</u>	<u>1</u>
<u>Programmation in situ</u>	<u>2</u>
<u>Problématique port ISP</u>	<u>2</u>
<u>Microcontrôleurs AVR Atmel et connecteur ICSP</u>	<u>3</u>
Format connecteur ICSP Atmel	3
Protocole de transfert	4
<u>Microcontrôleurs MicroChip</u>	<u>4</u>
<u>Programmateurs ICSP - AVR</u>	<u>5</u>
<u>USB-Asp</u>	<u>5</u>
Drivers Windows	6
Installation automatique - Zadig	6
Installation manuelle - Génération de fichier .Inf	6
Mise a jour du firmware interne	7
Exemples de modèles commerciaux	8
2012 - Chinois bas de gamme v1	8
2016 - Générique vII _ Alimentation 3v3-5v	8
Cordons et adaptateurs ICSP	9
<u>Arduino as ICSP</u>	<u>10</u>
<u>Interfaces physiques ICSP => Microcontrôleur cible</u>	<u>11</u>
Adaptateur ATtiny simple	11
Shield Arduino - Adaptateur ICSP	11
<u>Logiciels de programmation AVR</u>	<u>13</u>
<u>AvrDude</u>	<u>13</u>
<u>eXtreme Burner AVR</u>	<u>14</u>
Présentation et utilisation	14
Interface GUI	14
Fichiers de configuration	15
Chips.xml	15
Fuselayout.xml	15
Exemples de profils processeurs Arduino	15
<u>Rappels format de fichiers source</u>	<u>18</u>
Fichiers binaires .bin	18

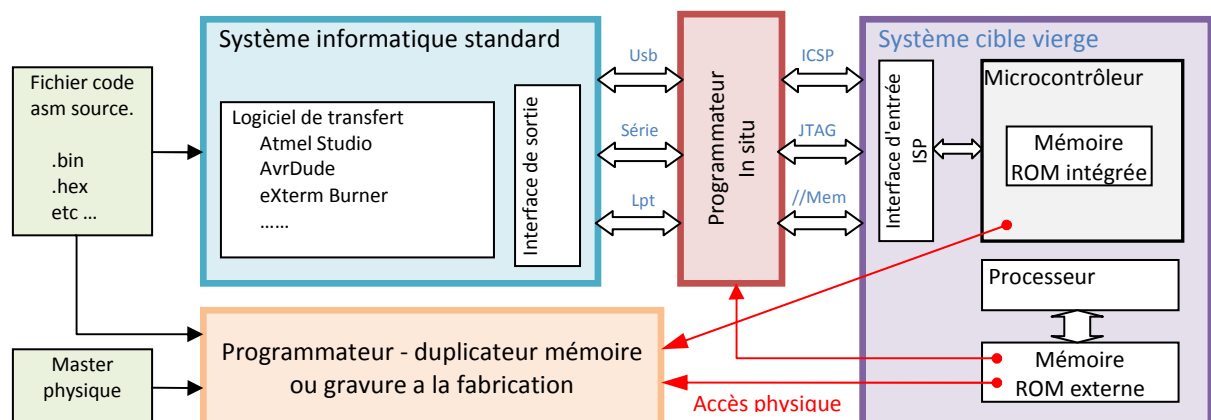
Fichiers Intel Hex	18
<i>Programmation µcontrôleurs AVR</i>	20
<i>Structure mémoire AVR</i>	20
<i>Ecriture mémoire Flash programme</i>	21
Matérielle série (ICSP) - parallèle	21
Logicielle interne - Bootloader	21
<i>Reset et bootloader</i>	22
<i>Fusibles de configuration</i>	23
<i>Signature</i>	23
<i>Registres de configuration</i>	24
Principales fonctions des registres de configuration	25
Horloge	25
Fonctions Processeur	25
Reset et Bootloader	26
Utilisation et calcul des valeurs fusibles	26
Arduino UNO vs ATmega 328p	26
<i>Influence de la configuration horloge CPU</i>	27
<i>Configuration de l'USART des processeurs AVR</i>	27
<i>Exemple pratique en environnement Arduino</i>	28
<i>ICSP en environnement Arduino</i>	29
<i>Particularités Arduino</i>	29
<i>Cartes Arduino et Icsp</i>	29
<i>Chargement programme et bootloader Arduino</i>	29
<i>Fusibles Atmel et cartes Arduino</i>	30
<i>Applications pratique</i>	32
<i>Préparation AT mega328 neuf avec programmeur standard</i>	32
<i>FAQ et résolution de problèmes</i>	34
<i>Liens et révisions document</i>	34
<i>Liens</i>	34
<i>Révision</i>	34

Avant propos

Tout système informatique nécessite pour fonctionner que tout ou partie de son programme lui soit directement disponible à la mise sous tension. Pour les systèmes autonomes complets comme généralement c'est le cas avec les microcontrôleurs ce sera le programme entier, pour des ensembles plus complexes disposant de systèmes externes de mémoire de masse ce sera uniquement la partie permettant le chargement du programme final, un exemple typique de cette configuration étant les ordinateurs classique type PC ou autre munis d'un bios.

Ce logiciel devant être stocké en mémoire permanente le problème de son chargement initial va se poser, le processeur ne disposant d'aucunes instructions à sa disposition cette opération devra obligatoirement faire appel à un processus extérieur adapté au type de mémoire à charger.

Tout un univers sera nécessaire à cet effet, chaque élément de la chaîne devant être compatible avec les autres, le choix de chacun d'entre eux impliquera ou sera alors issu du type de tous les autres.



Le but de ce document est de faire un point sur les méthodes, les matériels qui peuvent être employés et cela dans un environnement principalement dédié aux processeurs et microcontrôleurs Atmel AVR utilisés par les cartes Arduino.

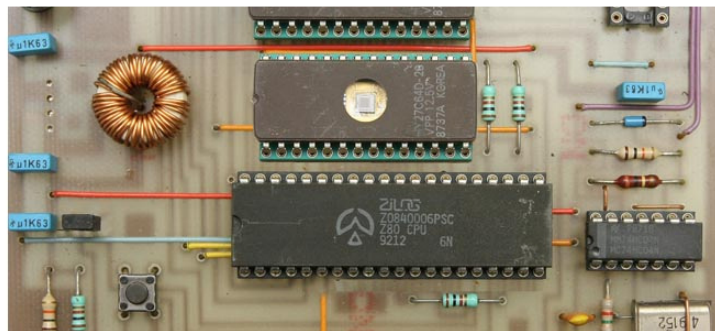
Programmation in situ

Problématique port ISP

Au début de l'utilisation des systèmes utilisant un microprocesseur le logiciel embarqué était mémorisé soit dans une mémoire morte externe a ce processeur (Prom fusible, Uvprom, Eeprom ...) soit directement intégré a celui-ci dans une mémoire UvProm ou a écriture unique OTP gravée a la fabrication. Toute modification du logiciel du système nécessitait alors la dépose physique du composant à reprogrammer. Outre les problèmes de cout, cette méthode est devenue difficilement applicable avec la généralisation des composants de surface soudés.



Microcontrôleur a
UvProm intégrée



Carte prototype utilisant des mémoires
externes sur support DIL (début années 90)

La programmation in situ ISP (In system Programming) d'un microcontrôleur, c'est-à-dire avec la totalité des composants soudés sur leur emplacement définitif a permis de s'affranchir de tous les problèmes engendrés par les méthodes précédentes. Que ce soit lors du développement de la carte prototype, des d'opérations de maintenance et de mise a jour du logiciel, ou directement en sortie de chaine de production la possibilité de charger le logiciel embarqué en utilisant un simple connecteur sans dépose de composants a largement facilité la conception des produits.

Par exemple ce petit module processeur embarqué dans des batteries Li-Ion d'appareils photo utilise un microcontrôleur Atmel Atmega8 dont la programmation est réalisée par l'intermédiaire des points test entourés en rouge en bas a droite de la photo.



Dans des systèmes plus complexes cette facilité de chargement du programme peut permettre en sortie de fabrication le chargement temporaire d'un logiciel de test ou de rodage avant que la version utilisateur finale soit implantée.

Cette méthode de travail n'a bien sur pu être généralisée qu'avec l'apparition de composants flash a bas cout ne nécessitant pas d'une source haute tension externe 12 ou 21v pour être effacée.

Microcontrôleurs AVR Atmel et connecteur ICSP

Afin d'obtenir un connecteur ISP le plus simple possible une liaison serie devra être employée, le transfert des données du programmeur externe à la mémoire interne du microcontrôleur devant aussi être effectué en un minimum de temps la solution la plus adaptée choisie par Atmel est d'utiliser le bus SPI de ses processeurs. Ce protocole permettant une connectivité de plusieurs esclaves au maître permet la programmation de plusieurs processeur a partir du même connecteur ou de réaliser une diffusion a plusieurs cartes simultanément. Cette méthode sera alors appelée ICSP (In Circuit Serial Programming).

Afin de différencier le mode de programmation ICSP de l'utilisation normale du bus SPI l'entrée reset du microcontrôleur sera maintenue à l'état bas tout le temps du transfert. Dans le cas de l'effacement complet de la mémoire programme du processeur une série d'impulsion sur cette entrée reset sera envoyée pour valider le cycle d'effacement.



Lors de la conception du système utilisant le microcontrôleur Il sera donc nécessaire que les deux lignes de données entrante (Mosi), sortante (Miso) et d'horloge (Sck) soient utilisées de façon judicieuse et que leur activation lors du flashage ne pose pas de problème d'impédance ou d'activation intempestive d'un actionneur. Cette notion sera bien sur plus critique dans le cadre de microcontrôleur munis d'un faible nombre de ports E/S comme les ATtiny.

Dans l'exemple ci-dessous, le micro contact pourra court-circuiter le signal Mosi et l'impédance basse de la Led pourra détériore le signal Sck. Pour éviter ces problèmes la solution sera alors de modifier l'attribution des ports E/S ou de rendre le périphérique non perturbant, par ex en utilisant un buffer pour commander la Led transformant cet ensemble en une haute impédance vis-à-vis de Sck, solution utilisée par exemple sur les cartes Arduino pour la Led connectée au port Dig 13 / Sck.

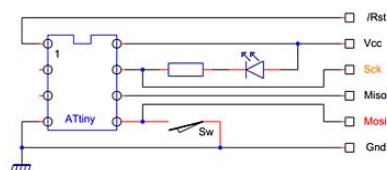


Schéma incorrect

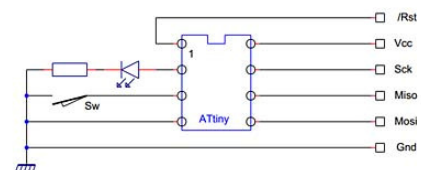


Schéma correct

Format connecteur ICSP Atmel

Le connecteur ICSP devra donc utiliser au minimum cinq points, trois pour le bus SPI, l'entrée reset du processeur et la masse, auxquels généralement on ajoutera une ligne d'alimentation du microcontrôleur.

Atmel préconise l'emploi d'un connecteur standardisé 6 points HE10, ce connecteur se retrouve par exemple sur les cartes Arduino en un ou deux exemplaires, le premier pour le processeur principal, le second pour le processeur assurant la conversion USB-Série.

Connecteur ICSP Atmel					
MISO	1	0	0	2	Vcc5v
SCK	3	0	0	4	MOSI
Reset	5	0	0	6	Gnd

Dans l'absolu le système d'écriture de la mémoire Flash via le bus SPI et le connecteur ICSP est similaire au système utilisé pour transférer un programme dans une carte Arduino via le port série.

Les principales différences tiennent dans le bootloader validé à la mise au niveau bas de l'entrée reset qui est inclus par le constructeur en dur dans le microcode du processeur au lieu d'être en flash, et dans l'accès complet aux ressources mémoire du microcontrôleur (Flash, EEprom, Registres fusibles de configuration).

Le protocole de dialogue entre le programmeur et le microcontrôleur utilise des blocs de 4 octets, un définissant la commande à effectuer, deux d'adressage et le dernier de donnée. Le document Atmel dont le lien suit offre plus de détails sur le fonctionnement de ce système.

http://www.atmel.com/Images/Atmel-0943-In-System-Programming_ApplicationNote_AVR910.pdf

Microcontrôleurs MicroChip

<https://www.elnec.com/sw/30277d.pdf>

Programmateurs ICSP - AVR

De nombreux programmeurs existent sur le marché, soit propriétaires et spécialisées, soit pilotés par un ordinateur standard par l'intermédiaire d'un port série, parallèle, ou plus couramment de nos jours Usb. Seuls quelques modèles courants seront évoqués dans les pages qui suivent.

USB-Asp

Ce programmeur crée par Thomas Fischl dont la documentation est disponible gratuitement en licence open source (rien ne vous empêche de contribuer avec une petite pièce sur son site) se trouve pour quelques euros sous de nombreuses formes sur eBay. Sa large distribution et son emploi facile en a fait un standard de facto.

<http://www.fischl.de/usbasp/>

Les signaux de sortie sont disponibles sur un connecteur HE10 male 2 fois 5 points utilisant un format propriétaire, un adaptateur sera donc nécessaire pour l'utiliser sur un connecteur 6 points standard Atmel. Aux lignes de données ICSP s'ajoutent une liaison série sur les broches 4 et 6, la restant non utilisée.

Connecteur USB ASP

MOSI	1	2	Vcc
	3	4	Tx
RST	5	6	Rx
SCK	7	8	Gnd
MISO	9	10	Gnd

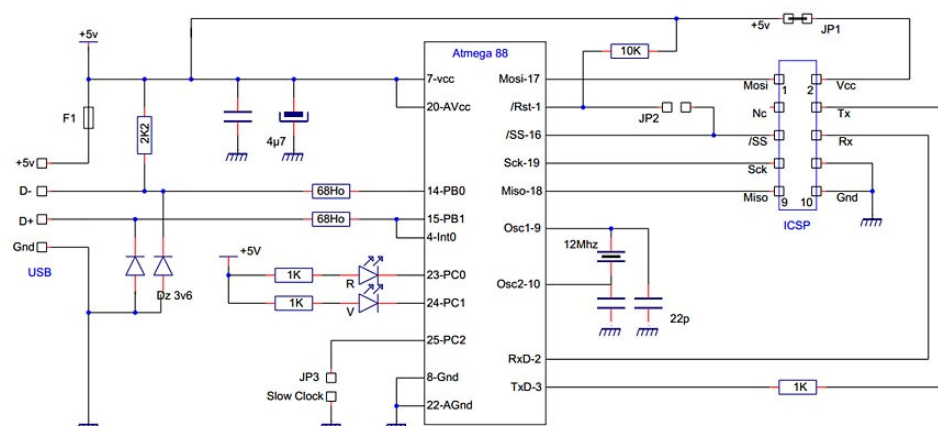
Ce programmeur permet plusieurs options et modes de fonctionnement sélectionnés par les positions de plusieurs ponts ou cavaliers :

- JP1 - Alimentation : Ce cavalier permet de valider l'alimentation du processeur cible par le programmeur, souvent double avec une sélection 3v3 - 5v, il sera à enlever pour une carte cible possédant sa propre alimentation.
- JP2 - Auto-Programmation : Ce cavalier normalement ouvert relie l'entrée reset du processeur programmeur au connecteur UsbAsp ce qui permet une mise à jour du firmware interne.
- JP3 - Vitesse SPI : La fermeture de ce cavalier peut être nécessaire pour les processeurs cible fonctionnant à une vitesse inférieure à 1.5Mhz, la fréquence de l'horloge SCK du bus SPI passe alors de 375kHz à 8kHz.

Deux Leds assurent une visualisation des états de fonctionnement du programmeur :

- Led Verte : Programmeur opérationnel sous tension.
- Led Rouge : Operations de lecture ou écriture sur le port SPI en cours.

Schéma USB-ASP original



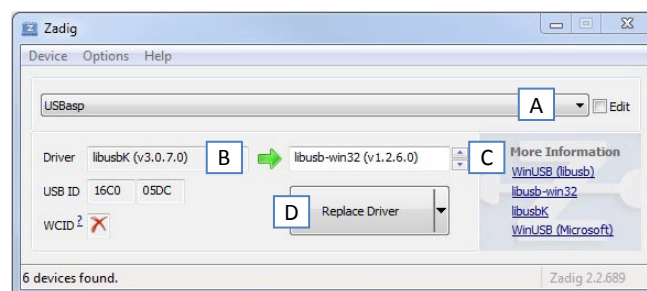
Si les systèmes Linux ne nécessitent pas de driver ce n'est pas le cas de Windows, plusieurs types de drivers sont fonctionnels, en particulier libusb0 et libusbK. Si libusb0 est fourni avec plusieurs logiciels dédiés AVR il ne fonctionne plus avec certaines versions récentes de l'IDE Arduino via Avrdude et quelques clones chinois, il sera nécessaire de choisir libusbk dans ce cas.

Deux solutions existent pour installer les drivers du programmeur sous Windows, une automatique avec le logiciel Zadig et une manuelle standard. Dans les deux cas le programmeur devra être connecté à la machine.

Installation automatique - Zadig

Ce petit logiciel fonctionne en version portable sans installation, il sera préférable de lancer son exécution en tant qu'administrateur; la procédure standard d'utilisation est la suivante :

- Mettre à jour la liste des périphériques Usb (Menu Options/List) et choisir le programmeur UsbAsp dont le driver est à installer dans la liste déroulante principale (A).
- Le type de driver éventuellement installé s'affiche (B), choisir le nouveau type dans liste déroulante(C) et actionner le bouton d'installation (D), une fenêtre de confirmation Windows doit s'ouvrir qui devra bien sur être validée.



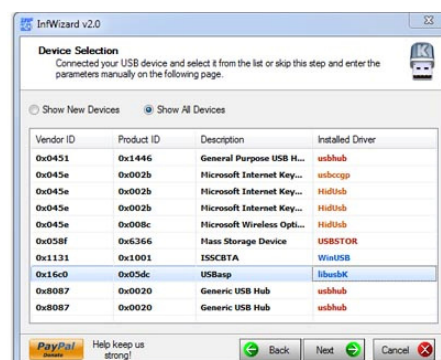
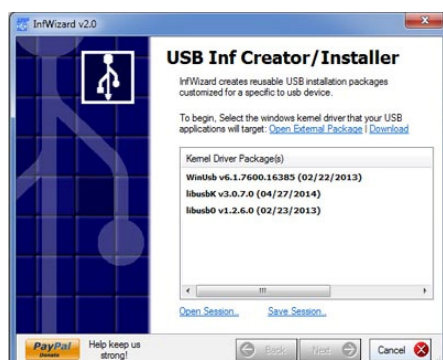
<http://zadig.akeo.ie/>

Installation manuelle - Génération de fichier .Inf

Cette solution si elle est un peu plus complexe à l'avantage de pouvoir générer un package de fichiers d'installation du driver adapté à son programmeur pouvant être utilisé sur plusieurs machines avec un contrôle total sur l'ensemble de la procédure.

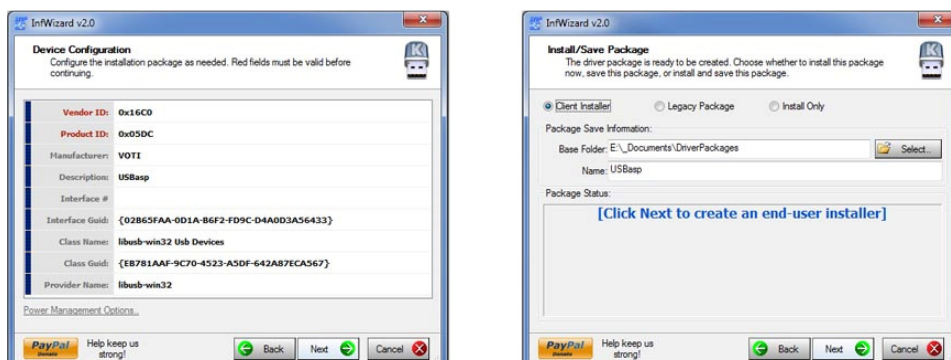
Le fichier libusbK-3-setup.exe devra être chargé et installé, le répertoire "Usbk développement Kit" doit apparaître dans le menu de démarrage. L'assistant de génération du fichier .inf "Driver install creator wizard" se décomposant en quatre étapes peut alors être lancé.

La première fenêtre propose le choix de type de driver à installer, la sélection du périphérique concerné devra ensuite être effectuée.



Une fenêtre récapitulant les paramètres utilisés pour générer le fichier .inf suit, chaque ligne pouvant étant éditable ceux-ci peuvent être modifiés pour s'adapter à un besoin spécifique.

La dernière étape propose le choix de la génération du fichier .inf seul (Legacy), d'un package complet comprenant un exécutable d'installation (Client installer) ou son installation directe (Install Only).



<https://sourceforge.net/projects/libusb/>
<http://libusbk.sourceforge.net/UsbK3/index.html>

Mise a jour du firmware interne

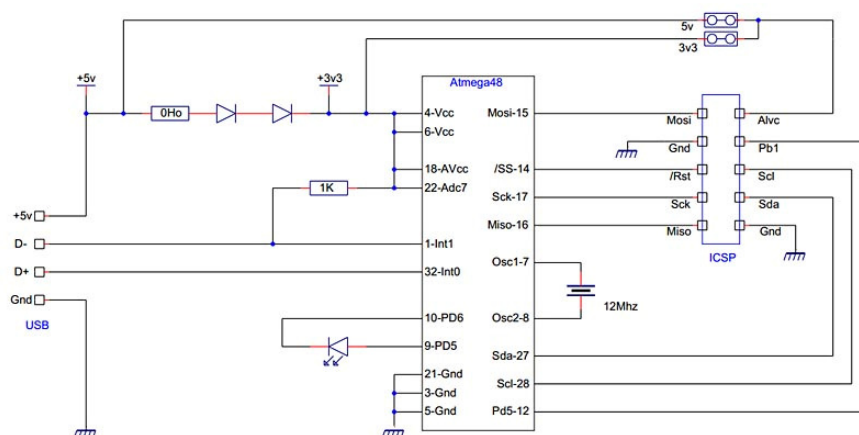
<http://www.rogerclark.net/updating-firmware-on-usbasp-bought-from-ebay/>

2012 - Chinois bas de gamme v1

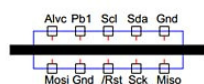
Ce modèle de conception vraiment un peu cheap est fourni avec un cordon fil à fil réalisant directement l'adaptation au format Atmel. Le processeur utilisé est un Atmega48 alimenté en 3v3 par le port Usb, celui-ci n'est muni d'aucune protection contre les ESD ou les courts circuits. Si théoriquement une option permet le choix de la tension de sortie d'alimentation du processeur cible en 3v3 ou 5v Usb la génération du 3v3 par deux diodes mises en série laisse douter de l'efficacité du montage. Le port Com prévu normalement sur le modèle d'origine est remplacé par le bus I2c issu de l'Atmega48.



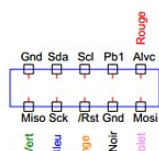
Schéma v1



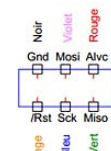
Connecteur et cordon de liaison



Connecteur
Male programmeur



Cordon - Femelle Usb-Asp



Cordon - Femelle ICSP

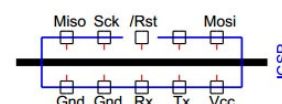
2016 - Générique vII _ Alimentation 3v3-5v

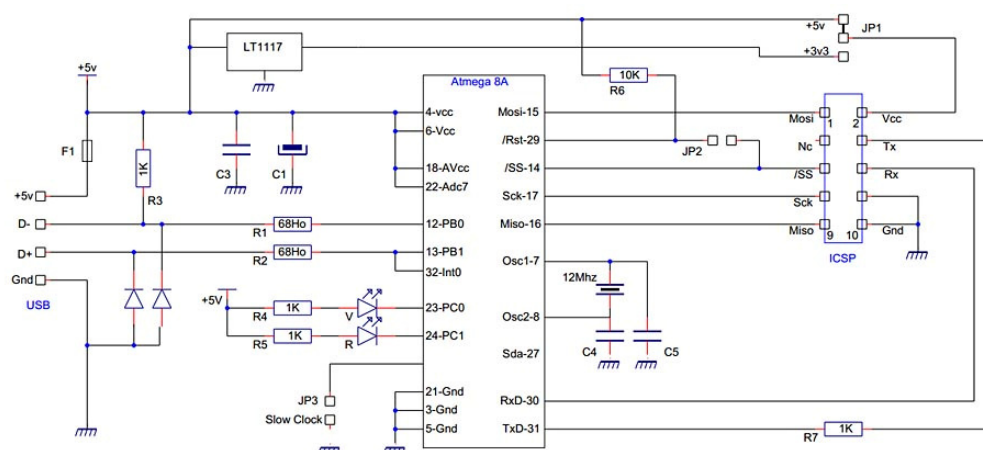


D'une qualité de fabrication supérieure au modèle précédent cette version se trouve actuellement sur le net à des prix variant de 1.5€ à 8€ (pourtant identiques) nu. Ce programmeur peut aussi être trouvé avec un module comprenant un support à force d'insertion nulle DIL 20 points prévu pour recevoir un ATmega328.

Un vrai régulateur LDO est utilisé pour générer la tension d'alimentation de 3.3v ce qui permettra d'alimenter un module cible correctement dans les limites du port USB. Attention, le bus SPI reste en 5V ce qui peut être délicat pour certains processeurs cible.

La prise de sortie constituée d'un connecteur HE10 à la norme UsbAsp d'origine est muni d'un détrompeur limitant les risques d'erreurs de connexion, son sens de montage est inversé par rapport au programmeur v1 précédent.



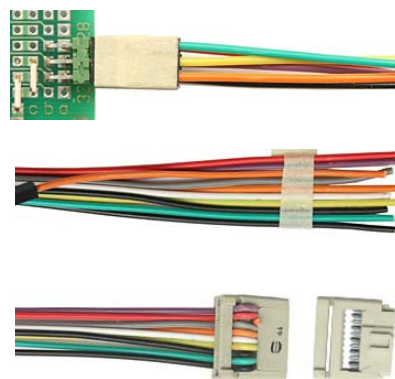


Cordons et adaptateurs ICSP

Un cordon ou un adaptateur est nécessaire pour passer du format 10 points Usb-ASP au format 6 points préconisé par Atmel et utilisé sur les cartes Arduino. Des adaptateurs se branchant sur un câble HE10 femelle-femelle sont disponibles sur le net, mais rajoutent des points de connexion pouvant induire des erreurs de transmission.



Si il ne me semble pas en avoir vu de tout fait chez les fabricants asiatiques il relativement aisé de réaliser un câble dédié, avec des fils d'expérimentation standards Dupont male-femelle et une prise HE10 femelle à sertir.



- Assembler et coller les 6 fils du côté ICSP en s'aidant d'un connecteur male existant pour les positionner.
- Supprimer la prise male des fils Dupont et les coller jointivement dans le bon ordre avec un scotch.
- Positionner la nappe dans la prise HE10, la sertir dans un étau et couper la partie superflue. Eventuellement clipser l'étrier anti-arrachement.

Ordre des fils		
1	Violet	ICSP Mosi
2	Rouge	ICSP Vcc
3	xx	non utilisé
4	Gris	COM Tx
5	Orange	ICSP /Reset
6	Blanc	COM Rx
7	Jaune	ICSP SCK
8	Noir	COM Gnd
9	Vert	ICSP Miso
10	Noir	ICSP Gnd

Les couleurs ne sont qu'indicatives, le fil 3 est obligatoire pour garder l'alignement mais inutilisé sera coupé au raz des deux cotes.

Le cordon ainsi obtenu permet une liaison directe du programmeur a la carte cible, l'interface série présente sur les modèles respectant la norme UsbAsp est aussi disponible.



Port
série

Atmel
6 points

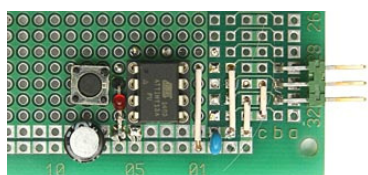
<http://www.instructables.com/id/Arduino-UNO-as-AtMega328P-Programmer/>

Interfaces physiques ICSP => Microcontrôleur cible

Si le protocole de programmation est commun à la gamme AVR, chaque modèle de processeur en fonction de la répartition de ses entrées / sorties et du facteur de forme de son boîtier devra disposer d'un adaptateur ICSP dédié. Selon ses besoins et le nombre de processeurs à charger ces interfaces devront utiliser des matériels plus ou moins sophistiqués, en particulier au niveau de la connexion des circuits intégrés, des supports tulipe n'auront pas une durée de vie ni n'offriront des contraintes mécaniques identiques au montage que des connecteurs ZIF à force d'insertion nulle, ceux-ci seront même un passage obligé dans le cas des boîtiers prévus pour un soudage en surface SOIC ou PLCC.

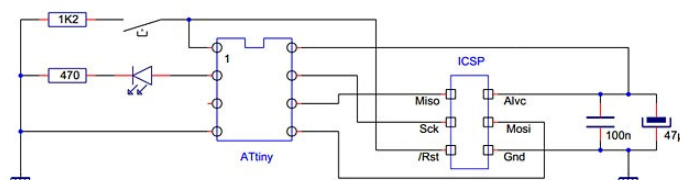


Adaptateur ATtiny simple



Réalisé plus à titre d'exemple que pour une utilisation réelle cet adaptateur permet de programmer les microcontrôleurs de la gamme AT tiny en boîtier DIL8 type 13A, 24, 45 et 85.

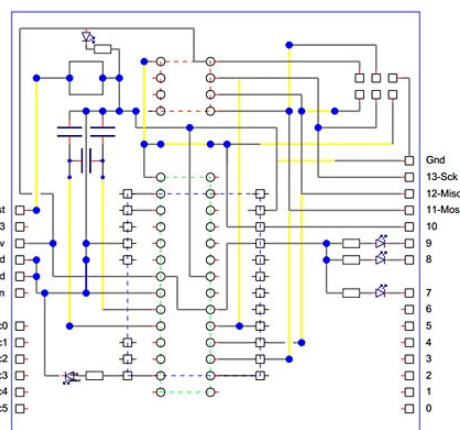
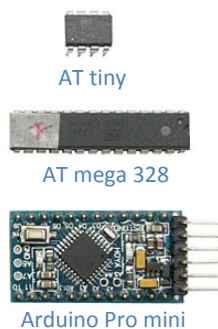
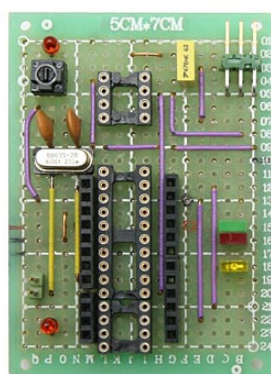
La réalisation sur un morceau de carte d'expérimentation ne présente pas de remarques particulières, la Led et le bouton reset ne sont là que pour réaliser des essais et ne sont pas utiles pour la programmation.



Shield Arduino - Adaptateur ICSP

Cet adaptateur est prévu pour être soit enfiché sur une carte UNO utilisant le logiciel ArduinoISP, soit pour être utilisé comme adaptateur avec tout programmeur ICSP. Trois types de support permettent de recevoir au choix, un ATtiny DIL8, un ATmega 328 ou une carte Arduino Pro mini.

Un quartz externe est prévu pour l'ATmega328, sa valeur importe peu son rôle étant seulement d'assurer un fonctionnement minimal du bus SPI pour tout les cas de configuration des registres fusibles, la platine d'essai utilisée ici est par exemple munie d'un quartz de récupération de 3.57MHz. Outre les 3 Leds d'état pilotées par le logiciel ArduinoISP sont connectées, une Led de présence tension et une seconde facultative de test sont ajoutées. Un bouton reset lui aussi facultatif permet d'initialiser la carte Arduino hôte.

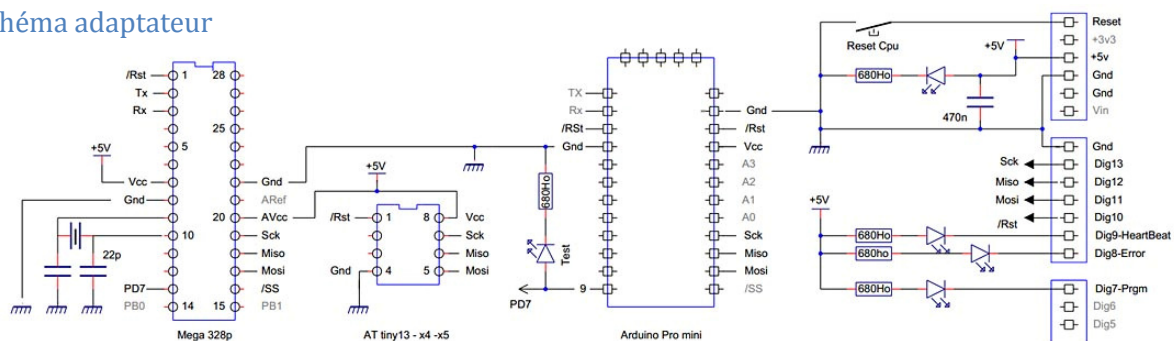


Shield prototype

Plan de câblage pistes

L'ensemble est réalisé sur une carte d'expérimentation a pastilles, le câblage étant effectué par des fils isolés coté composants, nus coté soudure. Certaines broches du connecteur pro mini sont supprimées pour faciliter le câblage et éviter l'injection de tensions indésirables. L'écartement entre les broches de la carte pro mini est insuffisant pour y insérer un support DIL28 à insertion nulle.

Schéma adaptateur



Logiciels de programmation AVR

AvrDude

eXtreme Burner AVR

Si AvrDude est extrêmement puissant et offre un large choix son interface en ligne de commande n'est guère conviviale et sans être manchot on peut la considérer d'un autre âge. De nombreuses versions de logiciels de programmation AVR existent sur le net, celle-ci ne fonctionne qu'avec un programmeur physique UsbAsp mais fait son travail correctement, reste très facile d'emploi, et facilement adaptable aux types de processeurs non prévus par défaut.

Présentation et utilisation

Les sources d'installation Windows ou Linux peuvent être trouvées à cet emplacement, ce logiciel est en shareware et n'est rétribué que par des dons en direct, attention donc aux nombreux sites de téléchargement le proposant avec l'assurance d'obtenir des logiciels publicitaires en sus.

Site source et manuel d'utilisation succinct :

<http://extremeelectronics.co.in/avr-tutorials/gui-software-for-usbasp-based-usb-avr-programmers/>

Lien de chargement direct (adresse mail requise) :

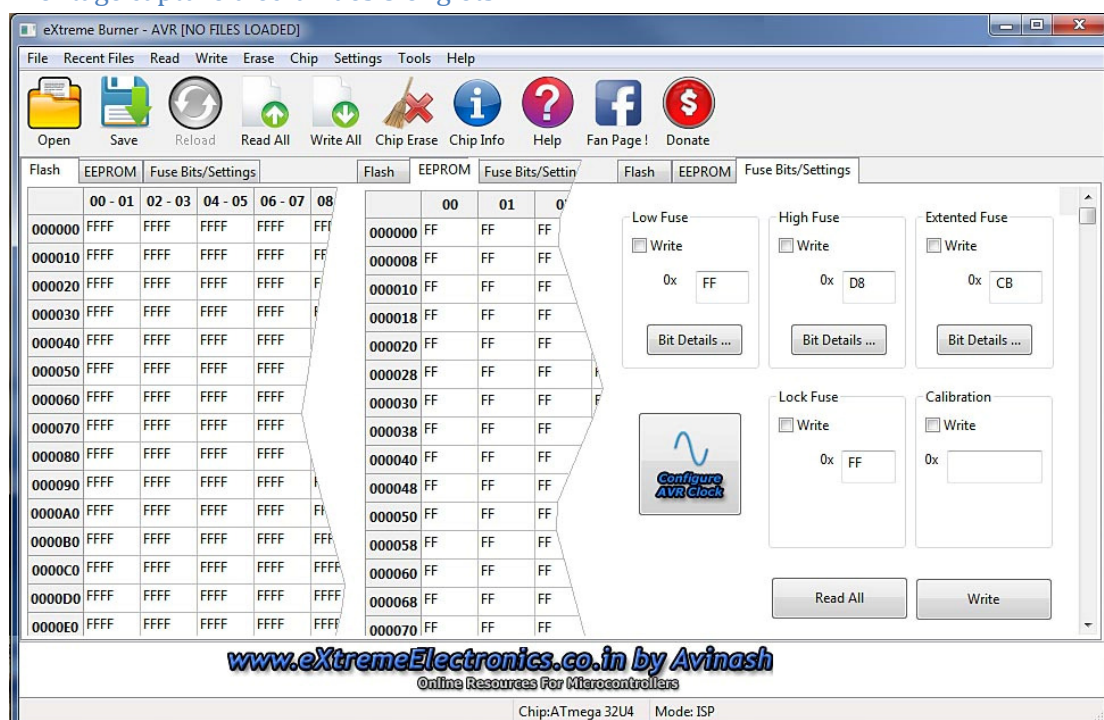
http://digital-wizard.net/files/extreme_burner_avr_v1.4.3_setup.exe

Le setup du logiciel propose et installe par défaut la version libusb0 des drivers utilisés par le programmeur UsbAsp, celle-ci disponible pour processeurs x86 ou ia64 n'est pas la dernière en date et peut poser des problèmes avec l'IDE Arduino 1.6. Il est alors possible comme indiqué dans le chapitre UsbAsp d'utiliser les drivers libusbK sans soucis avec ce logiciel.

Interface GUI

L'utilisation du logiciel ne pose pas de difficultés majeures, la sélection du type de processeur devra être effectuée au préalable via le menu Chip, si celui n'est pas disponible il sera toujours possible d'utiliser un processeur disposant de tailles mémoire et page similaires, et de passer outre l'erreur à la vérification de la signature ... ou de créer le profil dédié.

Montage capture d'écran des 3 onglets.



Les deux commandes principales Lecture - Ecriture permettent un accès simultané ou individuel a la mémoire programme flash, donnée EEprom et aux fusibles. Par protection une case à cocher inactive par défaut valide l'écriture de chaque registre de configuration fusible. Pour rappel l'enregistrement d'une valeur incorrecte peut empêcher tout accès au processeur, la seule méthode de reinitialisation étant d'utiliser le mode de programmation série ou parallèle haute tension.

La fenêtre principale dispose de trois onglets se rapportant a chaque catégorie :

- Flash : Mémoire programme affichée en lecture seule par mots AVR de 16bits, modifiable par lecture d'un fichier .hex.
- EEprom : Dump mémoire data éditable a volonté.
- Fuse bits : Registres de configuration du processeur affichés en valeur hexadécimale, si le profil est défini un affichage détaillé bit par bit est possible.

Fichiers de configuration

La configuration des processeurs utilisables par eXtremBurner est définie par les fichiers chips.xml et fuselayout.xml situés dans le sous répertoire Data du répertoire d'installation du logiciel. Si il est possible de passer outre la vérification de signature et de lire un processeur avec une configuration approchante de nouveaux profils peuvent être facilement écrits a partir des données constructeur situées en section *Memory Programming* de la datasheet du processeur.

Chips.xml

Chaque point de configuration du processeur est inclus dans une section <Chip> disposant d'une structure constante. L'ordre d'affichage des processeurs dans la liste déroulante du menu Chip est celle d'écriture de chaque section dans le fichier.

<Name> : Nom du processeur affiché dans le logiciel.
<Flash> : Taille en octet de la mémoire programme du processeur.
<EEprom> : Taille en octet de la mémoire donnée EEprom du processeur.
<SIG> : Signature du processeur sur 4 octets avec dans l'ordre 00, Sig02 - Type processeur, Sig01 - Taille mémoire, Sig00 - Constructeur (constante 0x1E).
<Page> : Taille page mémoire Flash en octet = Flash / Nb pages. Attention le constructeur donne la taille page en mots de 2 octets, il faudra multiplier par 2 cette valeur.
<xFuse profil>: Validation ou non de l'accès aux fusibles de configuration x (Low, High, eXtend) suivi éventuellement du numéro de profil utilisé. Si aucun profil n'est défini le bouton détail du registre fusible concerné est grisé.
<Lock><Calib> : Validation ou non de l'accès aux fusibles de protection logiciel et de calibration de l'horloge interne du processeur.
<Placement> : Source de l'image de positionnement du processeur dans le support ZIF du programmeur dédié au logiciel. Apparemment non utilisé.

Fuselayout.xml

Exemples de profils processeurs Arduino

ATtiny85 / 167 : Hormis le package et la distribution des entrées sorties ces processeurs sont identiques a l'ATtiny84 déjà implémenté dans le fichier *chips.xml*. Seule une copie de la section ATtiny84 sera à effectuer avec les nouvelles valeurs de titre et de signature

```
<CHIP>
  <NAME>ATtiny85</NAME>
  <FLASH>8192</FLASH>
  <EEPROM>512</EEPROM>
  <SIG>0x000B931E</SIG>
  <PAGE>64</PAGE>
  <LFUSE layout="2">YES</LFUSE>
  <HFUSE layout="3">YES</HFUSE>
  <EFUSE layout="1">YES</EFUSE>
  <LOCK>YES</LOCK>
  <CALIB>YES</CALIB>
  <PLACEMENT>.\Images\Placements\ZIF_DIP_
40.bmp</PLACEMENT>
</CHIP>
```

```
<CHIP>
  <NAME>AT tiny 167</NAME>
  <FLASH>16384</FLASH>
  <EEPROM>512</EEPROM>
  <SIG>0x0087941E</SIG>
  <PAGE>128</PAGE>
  <LFUSE layout="2">YES</LFUSE>
  <HFUSE layout="3">YES</HFUSE>
  <EFUSE layout="1">YES</EFUSE>
  <LOCK>YES</LOCK>
  <CALIB>YES</CALIB>
  <PLACEMENT>.\Images\Placements\ZIF_DIP_
40.bmp</PLACEMENT>
</CHIP>
```

ATmega 8/16U2 : Ces processeurs sont utilisés comme convertisseur USB-Serial sur les cartes Arduino Uno et Mega. La structure des bits fusible est identique pour ces deux modèles. Si le registre fusible Low est similaire à celui des ATmega 328 déjà présent les valeurs par défaut en diffèrent, en revanche les registres fusibles High et Ext étant différents il sera nécessaire de créer 3 nouveaux profils dans le fichier *Fuselayout.xml* en plus des deux sections du fichier *Chips.xml*.

```
<CHIP>
  <NAME>ATmega 8U2</NAME>
  <FLASH>8192</FLASH>
  <EEPROM>512</EEPROM>
  <SIG>0x0089931E</SIG>
  <PAGE>64</PAGE>
  <LFUSE layout="4">YES</LFUSE>
  <HFUSE layout="6">YES</HFUSE>
  <EFUSE layout="6">YES</EFUSE>
  <LOCK>YES</LOCK>
  <CALIB>YES</CALIB>
  <PLACEMENT>.\Images\Placements\ZIF_DIP_
40.bmp</PLACEMENT>
</CHIP>
```

```
<CHIP>
  <NAME>ATmega 16U2</NAME>
  <FLASH>16384</FLASH>
  <EEPROM>512</EEPROM>
  <SIG>0x0089941E</SIG>
  <PAGE>128</PAGE>
  <LFUSE layout="4">YES</LFUSE>
  <HFUSE layout="6">YES</HFUSE>
  <EFUSE layout="6">YES</EFUSE>
  <LOCK>YES</LOCK>
  <CALIB>YES</CALIB>
  <PLACEMENT>.\Images\Placements\ZIF_DIP_
40.bmp</PLACEMENT>
</CHIP>
```

```
<LOWFUSE>
.....
  <LAYOUT id="4" comment="For ATmega 8/16/32 U2">
    <BIT position="0" name="CKSEL0" details="Select Clock source" default="0" />
    <BIT position="1" name="CKSEL1" details="Select Clock source" default="1" />
    <BIT position="2" name="CKSEL2" details="Select Clock source" default="1" />
    <BIT position="3" name="CKSEL3" details="Select Clock source" default="1" />
    <BIT position="4" name="SUT0" details="Select start-up time" default="1" />
    <BIT position="5" name="SUT1" details="Select start-up time" default="0" />
    <BIT position="6" name="CKOUT" details="Clock output" default="1" />
    <BIT position="7" name="CKDIV8" details="Divide clock by 8" default="0" />
  </LAYOUT>
.....

<HIGHFUSE>
  <LAYOUT id="6" comment="For ATmega 8/16/32 U2">
    <BIT position="0" name="BOOTRST" details="Select Reset Vector" default="1" />
    <BIT position="1" name="BOOTSZ0" details="Select Boot Size" default="0" />
    <BIT position="2" name="BOOTSZ1" details="Select Boot Size" default="0" />
    <BIT position="3" name="EESAVE" details="EEPROM memory is preserved through the
Chip Erase" default="1" />
    <BIT position="4" name="WDTON" details="Watchdog Timer Always On" default="1" />
    <BIT position="5" name="SPIEN" details="Enable Serial Program and Data
Downloading" default="0" />
    <BIT position="6" name="RSTDISBL" details="External Reset Disable" default="1" />
    <BIT position="7" name="DWEN" details="debugWIRE Enable" default="1" />
  </LAYOUT>
.....
```

```

<EXTFUSE>
  <LAYOUT id="6" comment="For ATmega U2">
    <BIT position="0" name="BODLEVEL0" details="Brown-out Detector trigger level"
default="0" />
    <BIT position="1" name="BODLEVEL1" details="Brown-out Detector trigger level"
default="0" />
    <BIT position="2" name="BODLEVEL2" details="Brown-out Detector trigger level"
default="1" />
    <BIT position="3" name="HWBE" details="Hardware Boot Enable" default="0" />
    <BIT position="4" name="UNIMPLEMENTED" details="----" default="1" />
    <BIT position="5" name="UNIMPLEMENTED" details="----" default="1" />
    <BIT position="6" name="UNIMPLEMENTED" details="----" default="1" />
    <BIT position="7" name="UNIMPLEMENTED" details="----" default="1" />
  </LAYOUT>
</EXTFUSE>

```

ATmega 32/U4 : Ce processeur utilisé sur les carte Leonardo a une configuration de fusibles identiques a celle des AT90USB hormis pour les valeurs utilisée par défaut du registre Low, un nouveau profil sera la aussi nécessaire.

```

<CHIP>
  <NAME>ATmega 32U4</NAME>
  <FLASH>32768</FLASH>
  <EEPROM>1024</EEPROM>
  <SIG>0x0087951E</SIG>
  <PAGE>128</PAGE>
  <LFUSE layout="5">YES</LFUSE>
  <HFUSE layout="4">YES</HFUSE>
  <EFUSE layout="5">YES</EFUSE>
  <LOCK>YES</LOCK>
  <CALIB>YES</CALIB>
  <PLACEMENT>.\Images\Placements\ZIF_DIP_
40.bmp</PLACEMENT>
</CHIP>

```

```

<LOWFUSE>
.....
  <LAYOUT id="5" comment="For ATmega 16/32 U4">
    <BIT position="0" name="CKSEL0" details="Select Clock source" default="0" />
    <BIT position="1" name="CKSEL1" details="Select Clock source" default="1" />
    <BIT position="2" name="CKSEL2" details="Select Clock source" default="0" />
    <BIT position="3" name="CKSEL3" details="Select Clock source" default="0" />
    <BIT position="4" name="SUT0" details="Select start-up time" default="1" />
    <BIT position="5" name="SUT1" details="Select start-up time" default="0" />
    <BIT position="6" name="CKOUT" details="Clock output" default="1" />
    <BIT position="7" name="CKDIV8" details="Divide clock by 8" default="0" />
  </LAYOUT>
</LOWFUSE>

```

Rappels format de fichiers source

Fichiers binaires .bin

Les fichiers de type binaires sont la forme la plus simple existante pour exprimer le contenu d'une mémoire. Les données sont directement exprimées en hexadécimal octet par octet, sous forme linéaire à partir d'une adresse relative commençant à zéro.

Le rôle du programmeur se bornera à transférer ces données une à une, éventuellement en décalant l'espace d'adressage avec une valeur initiale différente de zéro.

Contenu fichier	0xAA	0x01	0x10	0x11	0x20	0x40	0xCA	0xAB	0xDD		
Mémoire sans décalage	Add	0x000	0x001	0x002	0x003	0x004	0x005	0x006	...		0x100
	Data	0xAA	0x01	0x10	0x11	0x20	0x40	0xCA	...		vierge
Mémoire avec offset 0x100	Add	0x100	0x101	0x102	0x103	0x104	0x105	0x106	0x106	0x108	0x109
	Data	0xAA	0x01	0x10	0x11	0x20	0x40	0xCA	0xAB	0xDD	Vierge

Fichiers Intel Hex

Le format Intel Hex est extrêmement ancien et apparu avec les premières EEprom, plus évolué qu'un fichier binaire ce type de fichier ne contient plus directement les données mais leur valeur au format texte, il possède en outre des fonctions d'adressage et de contrôle d'erreurs.

Les valeurs sont toujours exprimées sous forme de deux caractères texte représentant un nombre en hexadécimal, les valeurs supérieures à 8 bits sont inscrites en notation big endian octet de poids fort en premier. Chaque ligne de texte constitue un bloc d'enregistrement utilisant la structure suivante `":LLAAATTbbb...CC"` avec

- `:` Délimiteur de ligne annonçant le format intel Hex.
- `LL` Longueur du bloc de données en hexa (10 = 16 octets, maximum 255).
- `AAAA` Adresse d'offset.
- `TT` Type de ligne de données.
- `bbb...` Bloc de données de la ligne en hexa (deux caractères = 1 octet) de longueur variable.
- `CC` Checksum en complément à 2 calculé sur la longueur totale de la ligne avec la formule suivante $CC = 0x100 - ((LL + AA_H + AA_L + dd + dd + dd...) \& 0xFF)$

En fonction du type de ligne le contenu de AAAA et la signification du bloc de donnée bb peut varier.

- **Type 00** : Contenu de la mémoire, type d'enregistrement le plus courant du fichier.
 - `LL` : Nb octets du bloc de données, variable, généralement multiple de 8.
 - `AA_HAA_L` : Adresse 16 bits de début d'enregistrement du bloc de données dans la mémoire.
 - `TT=00` : Type 00
 - `bbbb` : Données de la mémoire exprimés linéairement octets par octets.
- **Type 01** : Fin de fichier : bloc de longueur nulle utilisé qu'une seule fois et constitue la dernière ligne du fichier.
 - `LL= 0` : Longueur de bloc nul
 - `AA_HAA_L=0000` : Non utilisé.
 - `TT=01` : EOF
 - `CC = FF` : Checksum = $0x100 - 0x01$

- **Type 02** : Adresse mémoire étendue : Pour systèmes d'adressage 32bits HHHHAAAA.
LL=04 : Longueur fixe.
AA_HAA_L=0000 : Non utilisé.
TT=03 : Type 03
HHHH : Valeur des 16bits haut de l'adresse 32bit utilisée pour les lignes de type 0 qui suivent, les 16 bits bas y étant déjà indiqués.
- **Type 03** : Adresse de début de segment.
LL=04 : Longueur fixe.
AA_HAA_L=0000 : Non utilisé.
TT=03 : Longueur fixe.
CS_HCS_L-IP_HIP_L : Contenu des registres 16bits d'adressage Intel CS et IP.

Exemple d'écriture de fichier Hex en mémoire.

Contenu fichier	:08 0000 00 AABCC0102030405 B8 :04 0100 00 11223344 51 :00 0000 01 FF										
Mémoire ligne 1	Add	0x000	0x001	0x002	0x003	0x004	0x005	0x006	0x007	0x008
	Data	0xAA	0xBB	0xCC	0x01	0x02	0x03	0x04	0x05	Vierge
Mémoire ligne 2	Add	0x100	0x101	0x102	0x103	0x104	0x105	0x106			
	Data	0x11	0x22	0x33	0x44	Vierge				

<http://www.hse-electronic.de/English/eDownloads/edownloads.html>

Si en environnement Arduino le chargement du programme dans le microcontrôleur et sa structure mémoire et ne sont pas des sujets importants il peut en aller tout autrement lors de l'utilisation de matériels sortant du standard habituel, que ce soit un processeur non prévu, utilisé seul ou avec une configuration d'horloge différente. Il est alors nécessaire de connaître les quelques concepts de base suivants.

Les microprocesseurs AVR disposent de trois espaces mémoire séparés en fonction du type de mémoire utilisé, la taille de chaque espace dépend du modèle de processeur employée. Si cette gamme de produit comprend des processeurs 8bits (ALU) la totalité des codes d'instruction (Opcode) sont en 16 ou 32bits, la mémoire programme flash est donc organisée sur une largeur de bande de 16bits, une adresse occupant alors 2 octets de mémoire.

Sauf pour les microcontrôleurs anciens ou de faible puissance la mémoire programme flash est elle aussi séparée avec en partie basse la zone RWW recevant le programme utilisateur et en partie haute une zone protégée NRW (No Read While Write) destinée à recevoir un bootloader ou un logiciel de flashage, une page de l'espace NRW ne pouvant être écrite que CPU à l'arrêt. Ces deux zones disposent aussi de niveaux de protection séparés en lecture et écriture via le registre fusible Lock.

The diagram illustrates the memory layout for ATtiny85 and ATmega328P. It is divided into three main sections:

- Flash programme:**
 - Starts at 0x0000 (Début).
 - Includes a Reset vector at 0x0000.
 - Contains a boot loader area.
 - Ends at 0x0027 (Fin).
- RAM interne:**
 - Starts at 0x0020 (Début).
 - Contains 32x Registers généraux (0x0020 to 0x003F).
 - Contains 64x Registers I/O (0x0040 to 0x005F).
 - Contains 160x Ext Registers I/O (0x0060 to 0x00FF).
 - Ends at 0x00FF (Fin).
- EEprom:**
 - Starts at 0x0100 (Début).
 - Contains ATtiny85 memory (0x0100 to 0x01FF).
 - Contains ATmega328P memory (0x0100 to 0x03FF).
 - Ends at 0x03FF (Fin).

Ecriture mémoire Flash programme

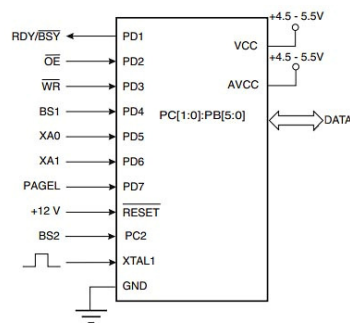
Matérielle série (ICSP) - parallèle

Selon le modèle de processeur et surtout le nombre de ports E/S qu'il propose deux méthodes permettent l'accès direct aux ressources mémoire de celui-ci. L'écriture de la mémoire est alors assurée en partie par le micro code interne et de la logique câblée. Ce sont les seules méthodes permettant l'accès en écriture aux registres fusible de configuration.

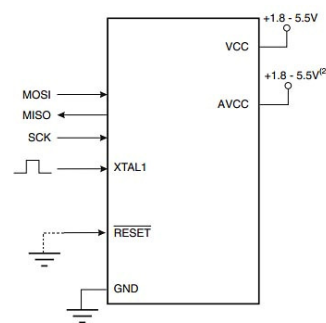
La méthode parallèle est la plus rapide mais utilisant un grand nombre de ports d'entrée est interdite a certains modèles de microcontrôleurs comme les ATtiny en boîtier 6 ou 8 broches. Elle permet un accès à la totalité des ressources, ce sera la seule permettant par ex de désactiver les fonctions SPI du processeur.

La méthode série utilisant la bus SPI ne nécessite qu'un nombre réduit de connexion et est la base des ports ISCP faisant l'objet de ce document. Les données et ordres de commandes sont transférés logiquement sous forme sérielle, pour différencier ce mode d'écriture mémoire de l'utilisation normale du port SPI l'entrée Reset est maintenue à l'état bas.

Attention : Ce mode nécessite une configuration d'horloge valide, un quartz externe sera donc obligatoire pour toute configuration de CKSEL autre que la sélection de l'horloge interne, la fréquence du bus SPI programmeur devra aussi être au maximum égale a $F_{cpu} / 6$.



Méthode parallèle



Méthode série ICSP

Logicielle interne - Bootloader

La mémoire flash programme est écrite via un code interne hébergé sur elle même, les données peuvent être acquises a partir de n'importe quelle source externe (Ports I/O, bus de communication SPI, Serie, I²C ...). Le microcode du processeur s'occupant de la gestion de base de la mémoire flash le logiciel de chargement (Bootloader) ne gèrera les données que par pages avec des ordres d'effacement, de lecture ou d'écriture via un buffer temporaire.

Si la totalité de la mémoire Flash peut être écrite de façon logicielle les processeurs suffisamment puissants réservent une zone celle ci pour héberger ce logiciel de chargement. Le programme utilisateur est alors écrit en zone basse RWW et le bootloader en zone haute NRWW.

Outre le fait que ces deux zones disposent d'une gestion séparée par le registre de protection mémoire Look et le fait que les adresses du vecteur Reset pointent en zone NRWW, la principale différence tient dans le comportement de la CPU lors de l'écriture ou l'effacement d'une page par le microcode du processeur.

- Ecriture RWW : La lecture du code en zone en NRWW est possible, la CPU est en fonction.

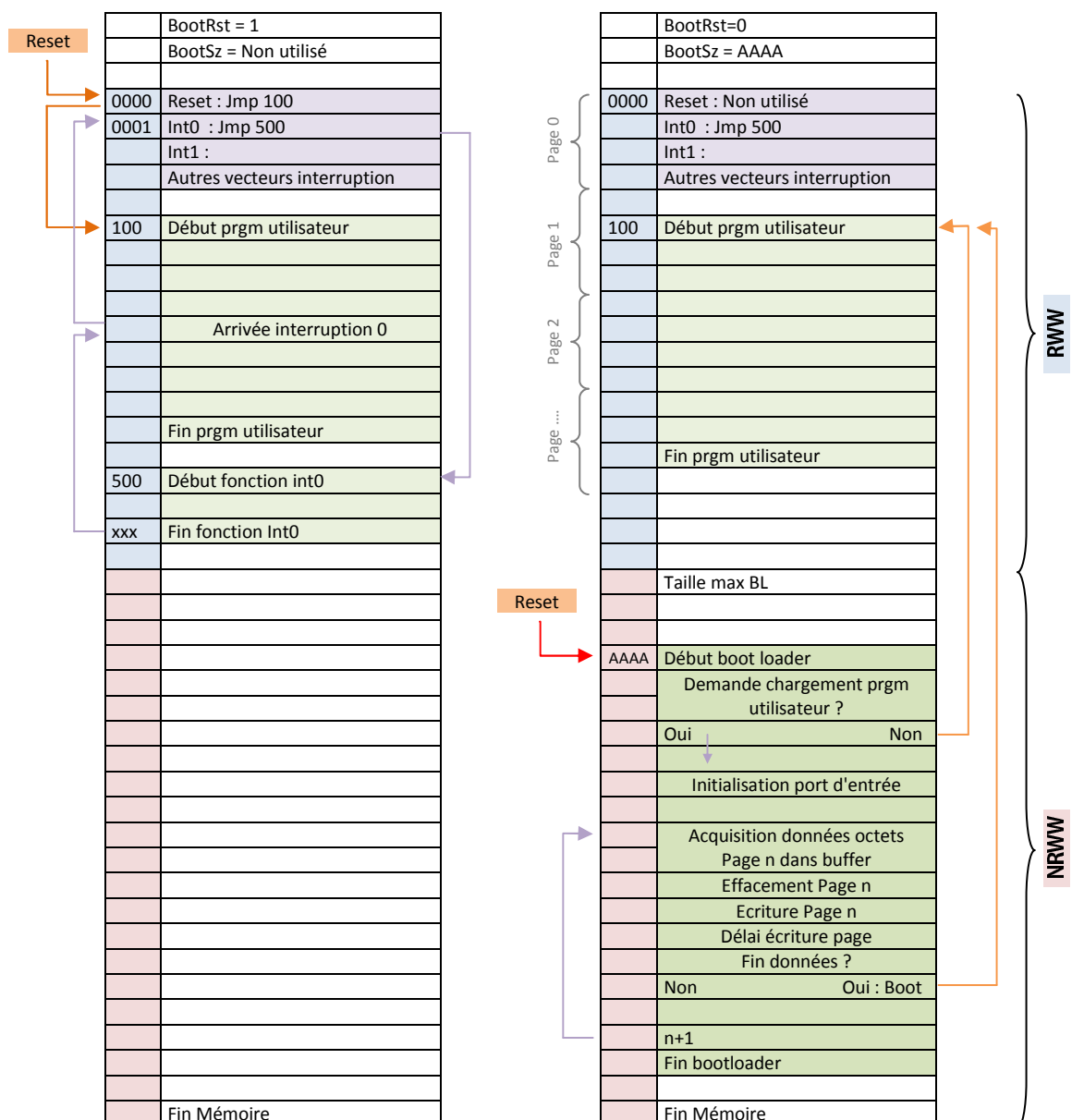
- Ecriture NRWW : La lecture du code en zone NRWW est impossible, la CPU est à l'arrêt.
- Dans tout les cas : La lecture de la zone RWW est interdite, le logiciel de chargement ne doit faire appel à aucune adresse dans cette zone, y compris les vecteurs d'interruption.

Dans la plupart des cas la zone NRWW est de taille fixe, seule l'adresse du vecteur Reset est affecté par les bits fusibles de configuration BOOTSZ.

Reset et bootloader

Traditionnellement a la mise sous tension d'un processeur ou après une action Reset le compteur program est pointé sur l'adresse 0000, le Cpu exécutant les instructions contenues a cet emplacement (généralement un jmp adresse). Les microcontrôleurs Atmel étant étudiés pour intégrer la présence d'un bootloader disposent d'options permettant de modifier cette adresse, les bits fusibles BootRst et BootSZ en définissant le mode de fonctionnement.

L'initialisation du processeur dans les deux cas pourrait ressembler aux organigrammes suivants.



Fusibles de configuration

Les fusibles de configuration sont des registres spéciaux définissant la configuration de fonctionnement du microcontrôleur, le mot fusible vient d'une époque où ces registres n'étaient pas stockés dans des cellules de type EEprom et réinscriptibles mais utilisaient des composants internes détruits lors de l'écriture, la configuration n'étant possible qu'une seule fois.

Plusieurs registres de 8bits sont utilisés dont notamment:

Nom	Accès	Fonction	
Low	R/W	Clock conf.	Gestion de différents points de configuration du microcontrôleur (Horloge, Bootloader, Autorisation fonctions ...).
High	R/W	Core conf.	
Ext	R/W	Extension	
Lock	R/W	Protection	Gestion de la sécurité (Copyright) et pouvant empêcher toute nouvelle écriture ou lecture du programme en Flash et des valeurs stockées en EEprom.
Cal	R/W	Calibration	Valeur de calibration de l'horloge interne stockée dans le registre OSCCAL à la mise sous tension.
Sig00	R only	Fondeur	Signature : Valeur sur 3 octets définissant le modèle exact du microcontrôleur.
Sig01	R only	Mem	
Sig02	R only	Modèle	

Signature

Les 3 octets de signature définissent le modèle exact de microcontrôleur, ils sont accessibles en lecture seule par le programme utilisateur ou en mode de programmation série ICSP ou parallèle.

Ils peuvent par exemple être utilisés par le logiciel utilisateur pour connaître la taille de l'espace mémoire en EEprom ou en Ram disponible, et sont généralement systématiquement lus par les logiciels de programmation pour la vérification de la bonne configuration des fusibles.

Le premier octet Sig00 définit le constructeur avec pour les microcontrôleurs Atmel AVR, une valeur constante égale à 0x1E.

Le second octet Sig01 indique la capacité mémoire disponible avec le code suivant :

Sig01 (Hex)	90	91	92	93	94	95	96	97	98	99
Taille Flash (ko)	1	2	4	8	16	32	64	128	256	512 ?

Le dernier octet Sig02 indique le modèle dans la gamme de taille mémoire indiquée par Sig01.

Processeur vs Sig01-Sig02	Taille mémoire flash en ko						
Gamme AT Tinny	1	2	4	8	16		
AT tiny 13	90-07						
AT tiny 13A	90-07						
AT tiny 20		91-0F					
AT tiny x4 (24, 44, 84)		91-0B	92-07	93-0C			
AT tiny x41 (441, 841)			92-15	93-15			
AT tiny x5 (25, 48, 85)		91-08	92-06	93-0B			
AT tiny x7 (87, 167)				93-87	94-87		
AT tiny x8 (48, 88)			92-09	93-11			
AT tiny x8 (48, 88, 168) - TSop			92-05	93-0A	94-06		
Gamme AT Mega	4	8	16	32	64	128	256
AT Mega 8-8L		93-07					
AT Mega 48-88 -168-328 (A/V)	92-05	93-0A	94-06	95-14			
AT Mega 48 - 168 - 328 PA	92-0A	93-0F	94-0B	95-0F			
AT Mega 640-1280-2560					96-08	97-03	98-01
AT Mega 1281-2561						97-04	98-02
AT Mega 8 - 16 - 32 U2		93-89	94-89	95-8A			
AT Mega 16 - 32 U4			94-88	95-87			

Registres de configuration

Les registres fusible de configuration sont au nombre de trois, la fonction de chaque de ces fusibles varie selon la gamme ou le modèle de processeur, les tableaux suivants indiquent pour les principaux processeurs de la gamme AVR 8bits cette correspondance et la valeur par défaut utilisée par le constructeur, les bits grisés ne sont pas utilisés.

AT Tiny ==>	20	13-13A	x4 - x5 - x7		x41	x8	x8 tsop	
	Registre Low							
Bit 7		SPIEN	CKDIV8					
Bit 6		EESAVE	CKOUT					
Bit 5	BODLEVEL2	WDTON	SUT1		SUT1	SUT1		
Bit 4	BODLEVEL1	CKDIV8	SUT0					
Bit 3	BODLEVEL0	SUT1	CKSEL3				CKSEL3	
Bit 2	CKOUT	SUT0	CKSEL2				CKSEL2	
Bit 1	WDTON	CKSEL1	CKSEL1			CKSEL1	CKSEL1	
Bit 0	RSTDISBL	CKSEL2	CKSEL0			CKSEL0	CKSEL0	
	Registre High							
Bit 7			RSTDISBL					
Bit 6			DWEN					
Bit 5			SPIEN					
Bit 4		SELFPRGEN	WDTON					
Bit 3		DWEN	EESAVE					
Bit 2		BODLEVEL1	BODLEVEL2					
Bit 1		BODLEVEL0	BODLEVEL1					
Bit 0		RSTDISBL	BODLEVEL0					
	Registre Extension							
Bit 7				ULPOSCSEL2				
Bit 6				ULPOSCSEL1				
Bit 5				ULPOSCSEL0				
Bit 4				BODDPD1				
Bit 3				BODDPD0				
Bit 2				BODACT1			BootSZ1	
Bit 1				BODACT0			BootSZ0	
Bit 0			SELFPRGEN	SELFPRGEN	SELFPRGEN		BootTRST	

AT Mega =>	8	48	88A-88PA 168-168A	328-328P	640 à 2560 1281-2561	8-16-32 U2	16-32 U4
	Registre Low						
Bit 7	BODLEVEL	CKDIV8					
Bit 6	BODEN	CKOUT					
Bit 5	SUT1	SUT1					
Bit 4	SUT0	SUT0					
Bit 3	CKSEL3	CKSEL3					
Bit 2	CKSEL2	CKSEL2					
Bit 1	CKSEL1	CKSEL1					
Bit 0	CKSEL0	CKSEL0					
	Registre High						
Bit 7	RSTDISBL				OCDEN	DWEN	OCDEN
Bit 6	DWEN				JTAGEN	RSTDSDL	JTAGEN
Bit 5	SPIEN						
Bit 4	WDTON						
Bit 3	EESAVE						
Bit 2	BOOTSZ1	BODLEVEL2		BOOTSZ1			
Bit 1	BOOTSZ0	BODLEVEL1		BOOTSZ0			
Bit 0	BOOTRST	BODLEVEL0		BOOTRST			
	Registre Extension						
Bit 7							
Bit 6							
Bit 5							
Bit 4							
Bit 3						HWBE	HWBE
Bit 2			BOOTSZ1	BODLEVEL2			
Bit 1			BOOTSZ0	BODLEVEL1			
Bit 0		SELFPRGEN	BOOTRST	BODLEVEL0			
	Valeur initiales par défaut						
Low	E1	62	62	62	62	5E	5E
High	D9	DF	DF	D9	D9	D9	99
Ext.	-	FF	F9	FF	FF	F4	F3

Principales fonctions des registres de configuration

Il est à noter que les fonctions utilisant une valeur booléenne sont en logique inversée, actives bit à zéro.

Horloge

CKSELxx : Sélection de la source générant l'horloge du processeur avec les valeurs suivantes

bin	hex	Source	0011	3	Oscillateur interne 128kHz
1xxx	F > 8	Quartz basse puissance avec 2Cx	0010	2	Oscillateur RC interne 8Mhz
011x	7 > 6	Résonateur ou quartz pleine onde	0001	1	Réservé
010x	5 > 4	Quartz basse fréquence	0000	0	Horloge externe

CKDIV8 : Diviseur fréquence d'horloge processeur.

0	Fcpu = F Horloge source / 8	1	Fcpu = F Horloge source
----------	-----------------------------	----------	-------------------------

CKOUT : Sortie horloge processeur sur la broche dédiée CLKO (dépend du processeur).

0	CLKO = Fprocesseur	1	CLKO = Port I/O normal
----------	--------------------	----------	------------------------

SUTx : Délai au démarrage du processeur lors d'un reset.

00	14 clock (Si BOD activé)	10	14 clock + 65ms (Alimentations standard)
01	14 clock + 4.1ms (Alims a montée rapide)	11	Reservé

Registres CAL / OSCAL : L'oscillateur interne 8Mhz est ajusté a cette fréquence par la valeur du registre OSCAL. Celui-ci est chargé au démarrage avec la valeur contenue dans le registre fusible CAL laquelle est initialisée en usine à la fabrication du processeur.

Important : Quelle que soit les valeurs utilisées dans CKSEL la logique d'écriture des mémoires flash et EEprom utilise l'oscillateur interne, une valeur de correction incorrecte provoquant une dérive trop importante de la fréquence peut engendrer des problèmes d'écriture.

Fonctions Processeur

SPIEn : Activation port SPI, ce bit n'est pas accessible en mode programmation série, le bus SPI étant alors nécessaire a son fonctionnement.

0	Bus SPI Actif	1	Bus SPI désactivé
----------	---------------	----------	-------------------

WDTON : Activation temporisation chien de garde en relation avec le registre WDTCSR.

0	Cdc inactif	1	Cdc configurable
----------	-------------	----------	------------------

JTAGEn : Autorisation protocole JTAG.

0	JTAG actif	1	JTAG inactif
----------	------------	----------	--------------

EESAVE : Protection en écriture de la mémoire EEprom.

0	Effacement interdit	1	Mode normal
----------	---------------------	----------	-------------

DWEn : Activation du mode de débogage debug Wire.

0	Debug wire activé (Voir doc Atmel)	1	Debug wire inactif.
----------	------------------------------------	----------	---------------------

OCDEn : Activation du mode de débogage OnChip Debug System.

0	Debug activ	1	Debug inactif
----------	-------------	----------	---------------

RstDISBL : Affectation de broche d'entrée Reset du processeur, le numéro de port I/O dépend du modèle de processeur.

0	Pin Reset = Port I/O	1	Pin Reset = Entrée Reset
---	----------------------	---	--------------------------

BOOTRST : Vecteur entrée Reset. Adresse de destination lors d'un reset matériel ou logiciel.

0	Adresse bootloader défini par BootSZ	1	Adresse standard 0x0000
---	--------------------------------------	---	-------------------------

BOOTSZxx : Taille et adresse de départ de la zone protégée NRWW réservée à héberger le code du bootloader. Les valeurs d'adresse dépendent du modèle et de la taille de la zone mémoire du processeur.

	AT Mega 8U2	AT mega 328	AT mega 2560
SZ1-SZ0	Taille max BL (ko) - Add départ	Taille max BL (ko) - Add départ	Taille max BL (ko) - Add départ
00	2k - 0x800	2k - 0x3800	4k - 0x1F000
01	1k - 0xC00	1k - 0x3C00	2k - 0x1F800
02	512 - 0xE00	512 - 0x3E00	1k - 0x1FC00
03	256 - 0xF00	256 - 0x3F00	512 - 0x1FE00

BODLEVELxx : Réglage du seuil de tension d'alimentation minimale avant de générer un reset, cette fonction peut être utilisée pour éviter des déclenchements de périphériques intempestifs suite à une baisse inattendue de Vcc. BODLevel inactif le reset ne dépend pas d'une valeur précise, des erreurs processeur pouvant alors se produire avant l'arrêt complet du système.

BODLevel2 - 1 - 0	111	110	101	100	011 > 000
Seuil déclenchent reset	Inactif	1.7 à 2v	2.5 à 2.9v	4.1 à 4.5v	Réservé

Utilisation et calcul des valeurs fusibles

Si le choix de la valeur de chaque bit du registre ne pose pas de soucis, celui de la valeur globale du registre est plus délicat les emplacements de chaque fonction pouvant changer le modèle de microcontrôleur. Il sera donc nécessaire de se reporter aux tableaux précédents ou aux datasheet du constructeur.

Plus simplement de nombreux sites proposent des calculateurs en ligne proposent un calcul ou un décodage de ces valeurs.

<http://www.engbedded.com/fusecalc/>

Arduino UNO vs ATmega 328p

A titre d'exemple le tableau suivant montre la différence entre les fusibles de configuration d'un microcontrôleur ATmega328P vierge, et un autre utilisé sur une carte Arduino UNO. Celles-ci expliquent les problèmes de compatibilité que peuvent rencontrer des montages processeur seul (Quartz obligatoire pour l'un), de temps de fonctionnement (Horloges Cpu de 16 et 1MHz) ou de chargement impossible d'un processeur neuf sur la carte UNO (Reset vs Bootloader).

Fuse Low	CKDIV8	CKOUT	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0
328 Vierge (0x62)	Fhorl / 8	PBO=I/O	tReset = 14ck + 65ms					Oscillateur Rc interne 8Mhz
Arduino (0xFF)	Fhorl / 1	PBO=I/O	tReset = 14ck + 4ms					Quartz basse puissance
Fuse High	RSTDISBL	DWEN	SPIEN	WDTON	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST
328 Vierge (0xD9)	Reset	debug off	SPI On	CDC normal	EEprom WR	Add bootloader=3800h		0000
Arduino (0xDE)	Reset	debug off	SPI On	CDC normal	EEprom WR	Add bootloader=3F00h		Bootloader
Fuse Ext.						BODLEVEL2	BODLEVEL1	BODLEVEL0
328 Vierge (0xFF)								Inactif
Arduino (0xFD)								Actif seuil 1.8v

Influence de la configuration horloge CPU

Hormis pour des processus pilotés et synchronisés par une horloge externe la plupart des fonctions hardware ou obtenues logiciellement vont dépendre de la fréquence d'horloge de la Cpu déterminée par la valeur des registres fusibles et éventuellement la fréquence du quartz externe utilisé. Seules les fonctions internes d'écriture en mémoire flash ou EEprom demandant un timing précis utilisent l'horloge interne 8MHz s'affranchissent de tout problème de configuration.

Ces facteurs seront à prendre en compte dans la majorité des cas lors du développement des logiciels. Un changement de la configuration horloge sans modification du programme pourra provoquer des dysfonctionnements de celui-ci, notamment au niveau des ports de communication asynchrones et bien sur toutes les temporisations internes ou réalisées par boucle logicielle. Les ports de communication synchrones I²C ou SPI ne seront impactés que si leur signal d'horloge maître voit sa fréquence contrôlée ou utilisent des filtres limitant la bande passante.

Point Important : Ces propos ne prennent en compte que l'aspect fréquence et non pas la source, la sélection d'une source incorrecte principalement celle d'un quartz externe sans que celui-ci soit présent empêchera le démarrage du microcontrôleur et toute tentative de reprogrammation des fusibles ultérieure.

Configuration de l'USART des processeurs AVR

De nombreux points de configuration concernent la gestion des ports de communication intégrés aux processeurs AVR, dans le contexte de ce document seul le mode de génération de la vitesse de transmission en mode asynchrone nous intéresse. L'horloge de synchronisation interne de l'Usart est générée à partir de la fréquence CPU et d'un diviseur programmable initialisé par la valeur des 12 bits bas du registre 16 bits UBRR.

$$nBauds = \frac{F_{cpu}}{16 \times (UBRR + 1)} \qquad UBRR = \frac{F_{cpu}}{16 \times Bauds} - 1$$

Dans un environnement informatique les calculs étant effectués sur des valeurs entières et l'arrondi réalisé à l'unité inférieure il sera préférable d'utiliser la méthode suivante permettant d'obtenir un résultat à 0.5 près.

$$UBRR = \frac{F_{cpu}}{16 \times Bauds} - 0.5 = \left(\frac{F_{cpu}}{8 \times Bauds} - 1 \right) / 2$$

Un mode double vitesse est sélectionnable via le registre de configuration UCSR en plaçant son bit1 U2X à l'état un, dans ce mode la vitesse de transmission en baud et la valeur de UBRR calculés avec les formules précédentes seront à multiplier par deux.

La valeur UBRR étant forcément arrondi à l'unité près la vitesse de transmission obtenue ne sera pas forcément égale aux valeurs standards, certaines combinaisons pourront provoquer un taux d'erreur supérieur à celui permis en mode asynchrone ou ne pourront pas être possibles du tout avec une fréquence Fcpu trop basse. Le tableau ci-dessous récapitule les valeurs obtenues pour les différentes sources d'horloge utilisées lors des essais, les combinaisons en rouge ne seront pas utilisables le taux d'erreur entre le débit obtenu et la valeur standard désirée étant trop important.

Mode asynchrone simple vitesse

	Valeur de UBRR calculée			Vitesse Baud réelle obtenue			Taux d'erreur (%)		
	9600	57 600	115 200	9600	57 600	115 200	9600	57 600	115 200
Qz 16Mhz	103	16	8	9615	58 823	111 111	0.16	2.2	-3.5
Interne 8Mhz	51	8	3	9615	55 555	125 000	0.16	-3.5	8.5
Qz 3.579 Mhz	22	3	1	9726	55 929	111 859	1.3	-2.9	-2.9
Interne / 8 = 1Mhz	6	0	0	8928	62 500	62 500	-7	8.5	-45

Mode asynchrone double vitesse

	Valeur de UBRR calculée			Vitesse Baud réelle obtenue			Taux d'erreur (%)		
	9600	57 600	115 200	9600	57 600	115 200	9600	57 600	115 200
Qz 16Mhz	207	34	16	9615	57 142	117 647	0.16	-0.8	2.12
Interne 8Mhz	103	16	8	9615	58 823	111 111	0.16	2.12	-3.5
Qz 3.579 Mhz	46	7	3	9520	55 929	111 859	-0.8	2.9	2.9
Interne / 8 = 1Mhz	12	1	0	9615	62 500	125 000	0.16	8.5	8.5

Exemple pratique en environnement Arduino

Le petit programme suivant va être chargé à partir d'une carte UNO sur un ATmega328, lequel sera déplacé sur un adaptateur ICSP muni de quartz interchangeable. Les temps de clignotement de la sortie 7 ou 9 et celui de transmission d'un bit sur le port série sont alors mesurés en faisant varier la configuration des registres fusible et la valeur du quartz externe.

Les valeurs suivantes sont alors mesurées, les résultats sont bien sur ceux attendus et montrent que la seule la configuration d'horloge prévue a la compilation du code source rend le système fonctionnel.

Source Cpu clock	CKSEL	CKDIV8	Low fuse	t Out7 (s)	t bit série (µs)	Bauds
Quartz 16Mhz	1111	1	0xFF	0.500143	104	9615
Quartz 3.5795Mhz	1111	1	0xFF	2.23	465	2150
Horloge interne 8Mhz	0010	1	0xF2	1.019	209.8	4766
Horloge interne 1Mhz	0010	0	0x72	8.13	1689	592

Il est aussi possible de tirer parti de ce test pour calculer la fréquence réelle de l'oscillateur interne laquelle dépendant d'une cellule RC est naturellement bien moins précise qu'un quartz, dans le cas de ce processeur elle est d'environ 7.9Mhz.

Listing Blink-Serial

```

bool BitOut=LOW;

void setup() {
  Serial.begin(9600);
  pinMode(9, OUTPUT);
  pinMode(7, OUTPUT);
}

void loop() {
  delay(500);
  digitalWrite(7, BitOut);           // Basculement sorties Dig7 - Pin 13
  digitalWrite(9, BitOut);           // Basculement sorties Dig9 - Pin 23
  Serial.write("AAAAA");             // Emission 9600 bauds - Pin 3
  BitOut = !BitOut;
}

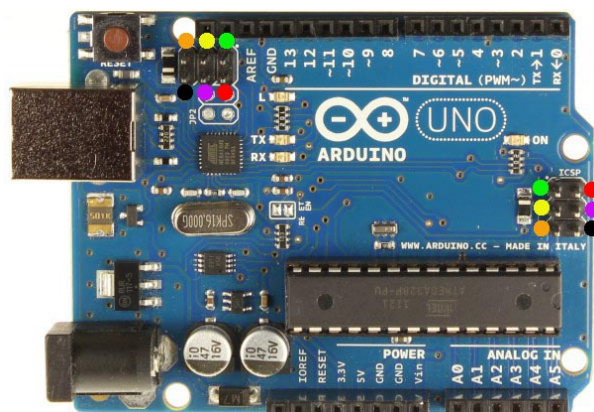
```

ICSP en environnement Arduino

Particularités Arduino




Cartes Arduino et Icspp

Selon les modèles les cartes Arduino disposent de un ou deux connecteurs ICSP dans un format 6 points respectant les préconisations Atmel. Ces connecteurs permettent une programmation directe de la CPU principale et éventuellement du microcontrôleur assurant la conversion Usb-Série utilisée pour charger le programme via l'IDE et le bootloader.



Quelques cartes de petit format ne proposent pas cette solution, la connexion au bus SPI devra se faire directement par les broches de sortie utilisateur standards.

Connecteur ICSP Atmel

MISO	1		2	Vcc5v
SCK	3		4	MOSI
Reset	5		6	Gnd

Le tableau suivant récapitule les connexions proposées et la référence du firmware utilisé pour les principales cartes Arduino.

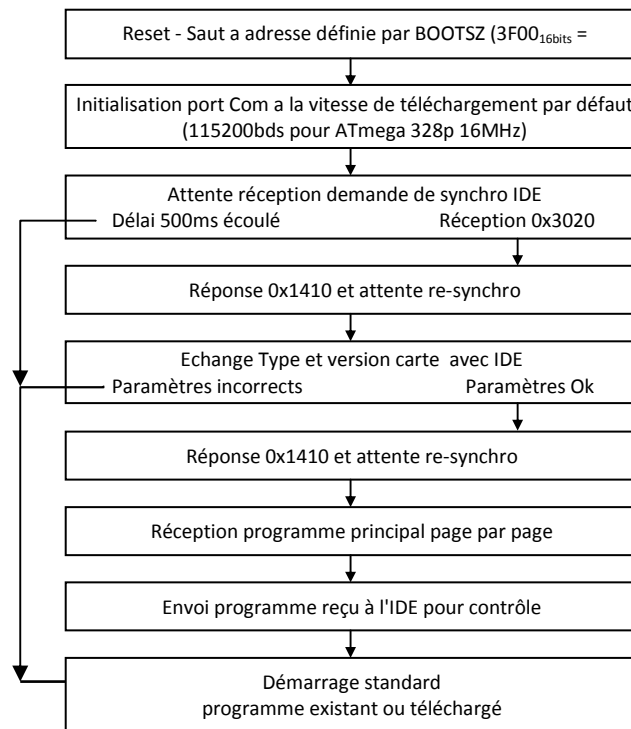
Carte	Cpu principale	ICSP Cpu	Bootloader	Cpu Série	ICSP Serie	Firmware série
UNO	328P	x	optiboot	16U2	x	
Leonardo	32U4	x		Accès USB direct à la cpu principale		
Mega	2560	x		8U2	x	
Due	Sam3X8E	x		16U2	x	
Esplora	32U4	x		Accès USB direct à la cpu principale		
ProMini	328P	non		Utilisation convertisseur FTDI externe		
Nano	328P	x		FT232	non	sans objet
Micro	32U4	x		Accès USB direct à la cpu principale		
Lilypad	328P	x		Utilisation convertisseur FTDI externe		

Chargement programme et bootloader Arduino

Un des agréments du système Arduino est la capacité de chargement d'un programme dans une carte directement à partir de l'IDE et d'un port USB standard sans l'aide d'un programmeur externe. Un des inconvénients de ce mode de fonctionnement est l'impossibilité d'utiliser un processeur vierge le code du boot loader devant y être présent.

Comme cela a été décrit dans le chapitre "Programmation des microcontrôleur AVR" ceux-ci proposent une place réservée en mémoire flash dédiée à héberger un programme de chargement lancé au démarrage du système.

Lors d'un reset initié soit a la mise sous tension, soit par une impulsion générée sur la broche DTR de l'adaptateur USB-Série par l'IDE en mode télé-versement ce programme est appelé et effectue la procédure simplifiée suivante :



Les échanges entre l'IDE et le bootloader sont basés sur le protocole STK500, les significations des différentes balises utilisées peuvent être trouvées dans le fichier stk500.h du répertoire optiboot.

Fusibles Atmel et cartes Arduino

	Low	High	Ex	Lock	Cal
328p vierge 2	62	D9	FF	FF	FF4A
Arduino 328p org	FF	DE	FD	FF	FF9A
Arduino 16U2 avec type 16 incorrect	FF	D9	-	FF	FF34
Arduino Leonardo avec type 32 incorrect	FF	D8	-	FF	6769
Arduino Mega 2560	FF	D8	FD	CF	FF52
Arduino Mega 8U2 serie type 8 incorrect	FF	D9	-	CF	FF98

#####

atmega328bb.name=ATmega328 on a breadboard (8 MHz internal clock)

```
atmega328bb.upload.protocol=stk500
atmega328bb.upload.maximum_size=30720
atmega328bb.upload.speed=57600

atmega328bb.bootloader.low_fuses=0xE2
atmega328bb.bootloader.high_fuses=0xDA
atmega328bb.bootloader.extended_fuses=0x05
atmega328bb.bootloader.path=arduino:atmega
atmega328bb.bootloader.file=ATmegaBOOT_168_atmega328_pro_8MHz.hex
atmega328bb.bootloader.unlock_bits=0x3F
atmega328bb.bootloader.lock_bits=0x0F

atmega328bb.build.mcu=atmega328p
atmega328bb.build.f_cpu=8000000L
atmega328bb.build.core=arduino:arduino
atmega328bb.build.variant=arduino:standard
```

Applications pratique

Le but de final de ce document est bien sur de proposer des méthodes permettant de programmer des processeurs AVR, soit pour remplacer un circuit intégré défectueux sur une carte existante, soit pour utiliser ce processeur seul dans un projet personnel. En étant réaliste ce dernier type de manipulation n'a qu'un intérêt limité, il sera souvent plus rentable d'utiliser un module déjà monté que des composants bruts, une carte Arduino Pro mini coutant souvent moins cher qu'un processeur ATmega 328 en boîtier DIL seul.

Les possibilités et combinaisons permettant d'arriver a ces résultats sont infinies, les exemples proposés seront à prendre pour ce qu'ils sont, juste des guides et non pas des procédures à suivre absolument pas a pas.

Le matériel utilisé sera celui décrit dans les chapitres précédents, pour plus de détail sur ceux-ci se reporter a ces informations plus spécifiques.

Préparation AT mega328 neuf avec programmeur standard

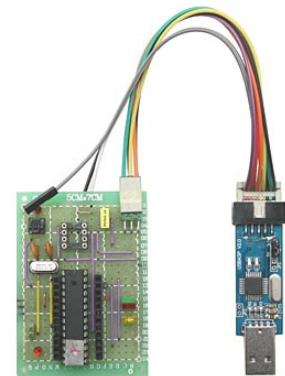
Cette manipulation est la plus facile, le processeur sera a destination exclusive d'une carte Arduino UNO ou pour etre utilisé sur une carte personnelle munie d'un quartz externe 16Mhz et programmée par une interface serie FT232 (configuration des cartes pro mini).

Matériel utilisé :

- Programmeur usbAsp
- Adaptateur ICSP - AT mega328 dil avec quartz ou carte UNO.
- Logiciel eXtrem Burner AVR.

Le microcontrôleur ATmega 328 neuf est inséré dans le support DIL de l'adaptateur ou de la carte Arduino Uno, le programmeur est relié sur la prise ICSP par l'intermédiaire de son cordon.

L'ensemble est connecté à un PC et le logiciel eXtrem Burner AVR est lancé.



La programmation du microcontrôleur va suivre les étapes suivantes :

- Menu Chip : Sélection AT méga 328p.
- Menu Erase : Chip erase pour un effacement complet d'éventuelles scories dans le cas d'un processeur de récupération.
- Menu Open : Ouvrir le fichier [optiboot_atmega328.Hex](#) contenant le bootloader contenu dans le répertoire hardware\arduino\avr\bootloader de l'IDE. Des données doivent apparaitre à partir de l'adresse 0x7E00 en mémoire flash.
- Dans la fenêtre des données, onglet Fuse settings modifier les valeurs de fusibles et cocher l'option Write pour celles modifiées avec les valeurs suivantes (voir tableau précédent) : [Low = 0xFF](#) [High = 0xDE](#) [Extended = 0xFD](#)
- Lancer l'écriture par le bouton Write All.
- Le processeur peut être retiré (hors tension) et utilisé sur une carte UNO standard.

Si un quartz n'est pas présent sur la carte adaptateur, la programmation d'une puce vierge sera possible l'oscillateur interne étant actif par défaut, lors de la modification du Low Fuse et la sélection d'un quartz externe tout nouvel accès deviendra impossible.

FAQ et résolution de problèmes

Problèmes IDE Arduino ou Avrdude.

Liens et révisions document

Liens

<https://www.arduino.cc/en/Hacking/Bootloader?from=Tutorial.Bootloader>
<https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard>
<http://thepolyscope.com/fr/graver-un-bootloader-sur-atmega328p-pu-avec-un-arduino-uno-et-sans-cristal/>
<https://www.arduino.cc/en/Hacking/ParallelProgrammer>
<https://learn.adafruit.com/usbtinyisp>
http://www.chicoree.fr/w/Arduino_sans_Arduino

Révision

v1.00 30/08/2016 Première diffusion.

```
#include <Wire.h>                                //Déclaration bibliothèque
const uint8_t ETest1=7;                          //Déclaration entrées de test, leur mise
const uint8_t ETest4=4;
```

```
const uint8_t Add_I2C=0x27;
```

```
//Déclaration adresse périphérique
```