Compte rendu de Travaux Pratiques

COMMENT PROGRAMMER UN ATMEGA 328P

I. Objectif du TP

Il est question d'installer un fichier de démarrage (ou Bootloader) sur un microcontrôleur Atmega-328P avant de pouvoir le programmer pour une application.

II. Mode Opératoire

A) Réaliser avant toute chose, une planche d'essais qui fixe de façon définitive un Arduino UNO et une plaquette « BREADBOARD » pour faire les essais. Cette planche servira à chaque fois de Banc de programmation.
B) Raccorder le microcontrôleur « Vierge » au milieu de la Breadboard ; puis câbler le Quartz (16Mhz) et les deux condensateurs de résonance, la résistance de Pull-up et l'alimentation sur la carte Arduino suivant le schéma (Figure 1).
C) Raccorder le câble USB entre l'Arduino et l'ordinateur qui contient l'I.D.E. (version 1.8.13 minimum) ; puis ouvrir l'IDE. On vérifie que le carte Arduino est bien associée à un port com du PC ; puis on va sélectionner à partir des fichiers exemples, le programme « ArduinoISP.ino ».
D) A partir de la nouvelle fenêtre, menu « outils », sous-menu « Programmateur », puis « AVRISP mkII ». On téléverse ensuite le code sur la carte Arduino ; ce qui a pour effet de transformer la carte Arduino en programmateur « ISP » (pour In-circuit Serial Programer).
E) L'interface logicielle est devenue « ARDUINO ISP ». On retourne sur le menu « outils », puis sur le sous-menu « Programmateur » , puis « Arduino as ISP ». Encore une fois « outils », puis « Graver la séquence d'initialisation ».
F) Une fois réalisée, le microcontrôleur Atmega 328P « cible » est doté d'un Bootloader.
G) On peut passer à la programmation d'un applicatif en remplaçant l'Atmega de l'Arduino, par l'Atmega « cible » et faire fonctionner le programme « Blink » pour le vérifier.

III. Schéma du Montage

Le schéma suivant est théorique. Il convient de l'adapter avec un schéma de câblage en fonction des conditions de l'expérimentation.

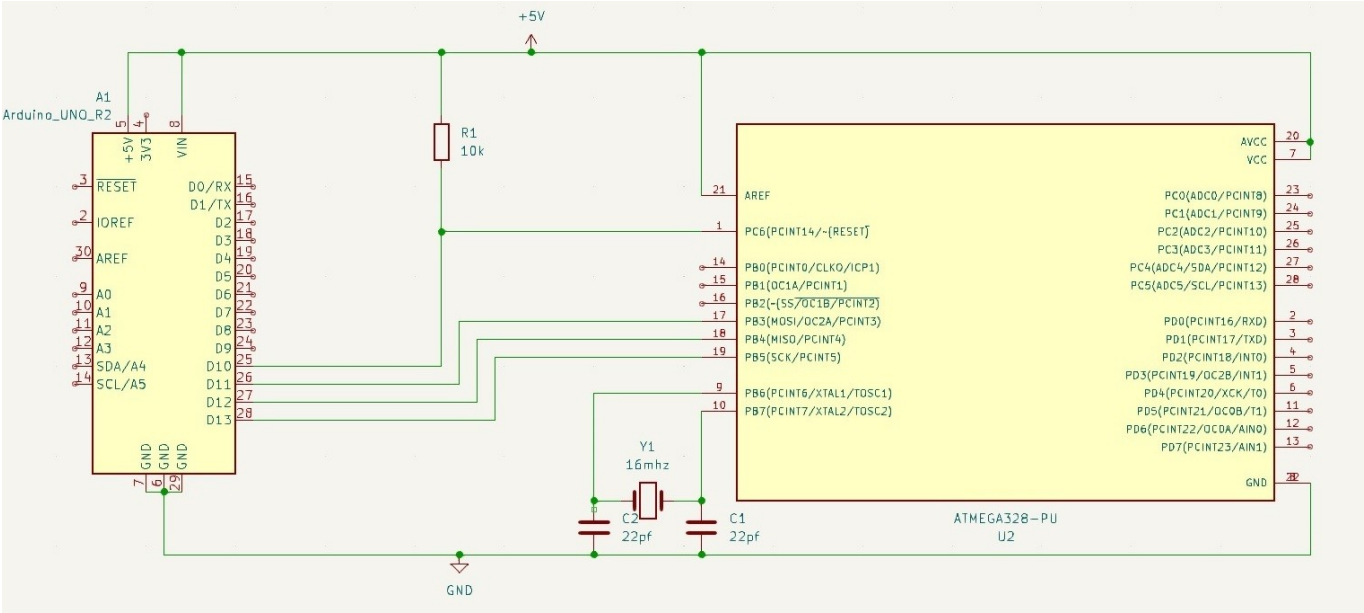


Figure 1: Schéma de principe du montage

Ci-dessous, un schéma de câblage parmi d'autres :

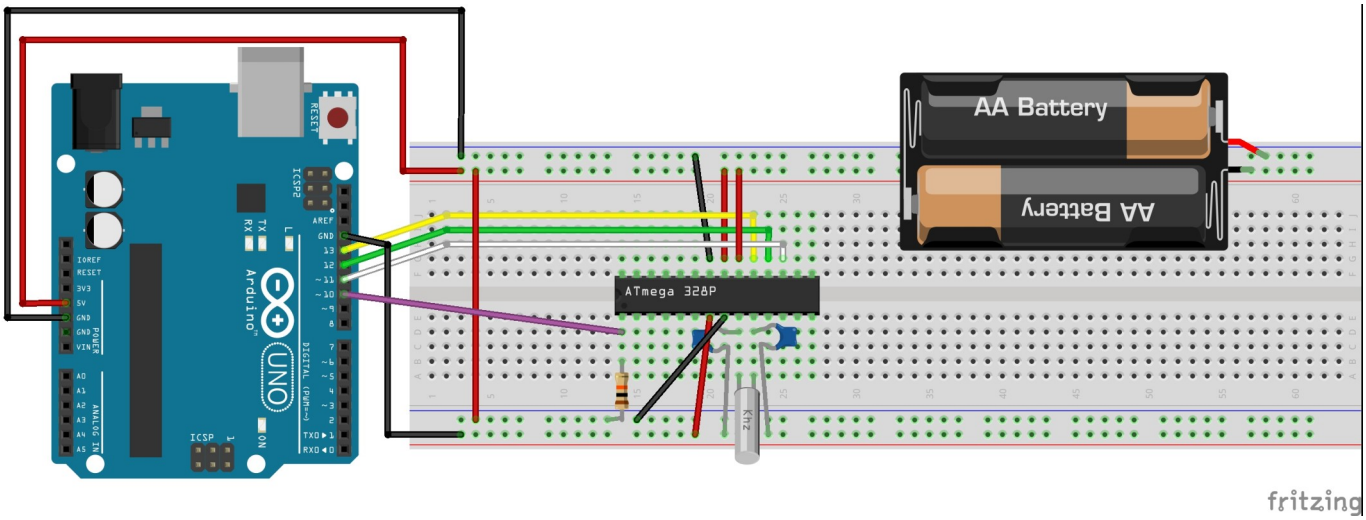


Figure 2: Plan de câblage avec "Breadboard"

...

IV. Liste du Matériel

Voici la liste minimale du matériel dont vous avez besoin.

Désignation	Qté.	Marque / Modèle	Fournisseur
Planchette MDF 10x20 cm ep 8 m/m	1	----------	Weldom
Arduino UNO R3	1	Original ou clone	Gotronic
Atmega 328PU	1	Atmel	Gotronic
Quartz 16 Mhz bas profil	1	Farnell	Gotronic
Condensateur céramique 22pf	2		Gotronic
Résistance 10KΩ 1/4 w	1		Gotronic
Breadboard 470 points	1	EXP355	
IDE Arduino	1	Arduino	Site web
Fils de connection	**10**	**---------**	

V. Essais & Mesures :

Quelques explications théoriques peuvent éclaircir la compréhension de l'expérimentation ; principalement sur le brochage de l'Arduino et celui de l'Atmega ; mais également sur le protocole SPI.

Tableau de Correspondance des broches

C'est souvent une source de confusion dans beaucoup de schémas. Il ne faut pas confondre la nature du signal avec sa broche correspondante :

Signal ou Fonction	Broches du A328P	E/S Arduino
GND (masse)	22 et 8	GND
VCC (+5v)	7 et 20	+5v ; 3v3 ; Vin
SPI-mosi	17 (Port B3)	D11
SPI-miso	18 (Port B4)	D12
SPI-clk	19 (Port B5)	D13
SPI-ss	16 (Port B2)	D10
I2C-sda	27	A4
I2C-scl	28	A5
Usart-TX	3	D1
Usart-RX	2	D0
Xtal-1	9 (Port B6)	/
Xtal-2	10 (Port B7)	/
Reset	1	Reset
Aref	21	Aref
Analog Input	De 23 à 28	A0 to A5
Digital In/Out	De 2 à 6 ; de 11 à 19	D0 to D13

On peut également représenter cette correspondance avec une illustration :

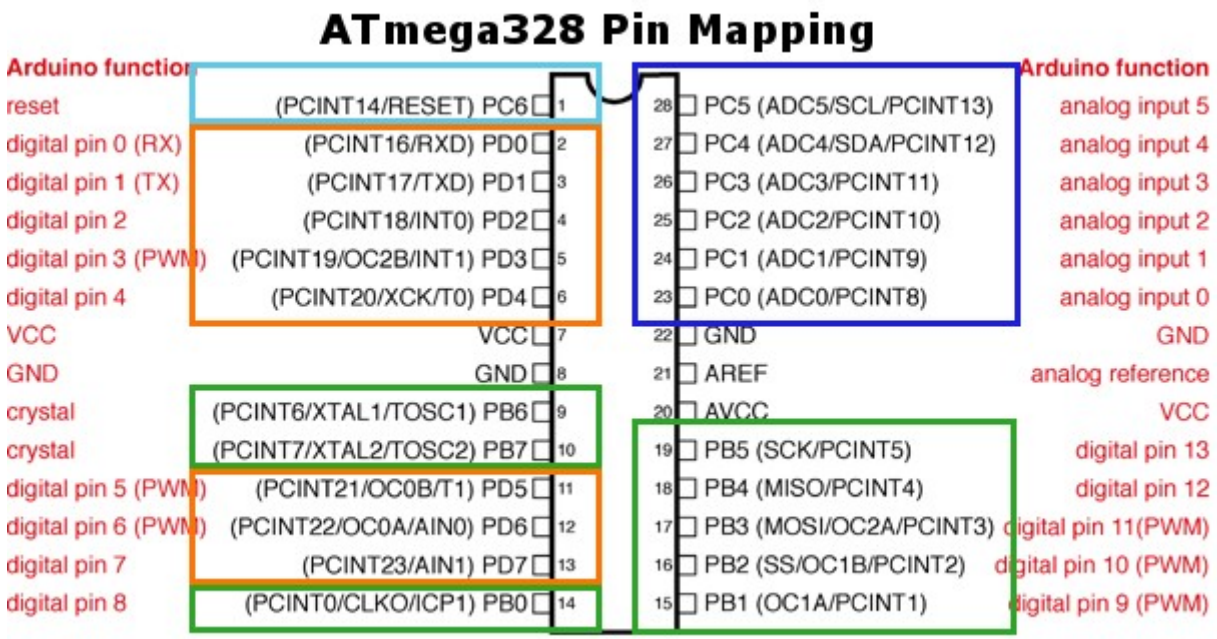


Figure 3: Le groupe "B" en vert.

Étude du Protocole S.P.I. :

Préliminaire : les registres

On en profite pour comprendre l'articulation ou le mécanisme principal de l'expérimentation. Il faut toujours garder à l'esprit que la programmation de fonctions délicates peut se réaliser avec un langage de haut niveau, mais au détriment des performances. De plus, dans certains cas, il faudra utiliser un langage qui manipule directement les registres du microprocesseur. Il est toujours possible d'utiliser …

Microchip Studio 7 (Version: 7.0.2594 -) © 2020 Microchip Technology, Inc.

Package GUID: e874ffe4-fbe3-4624-9a17-61014ede02d0

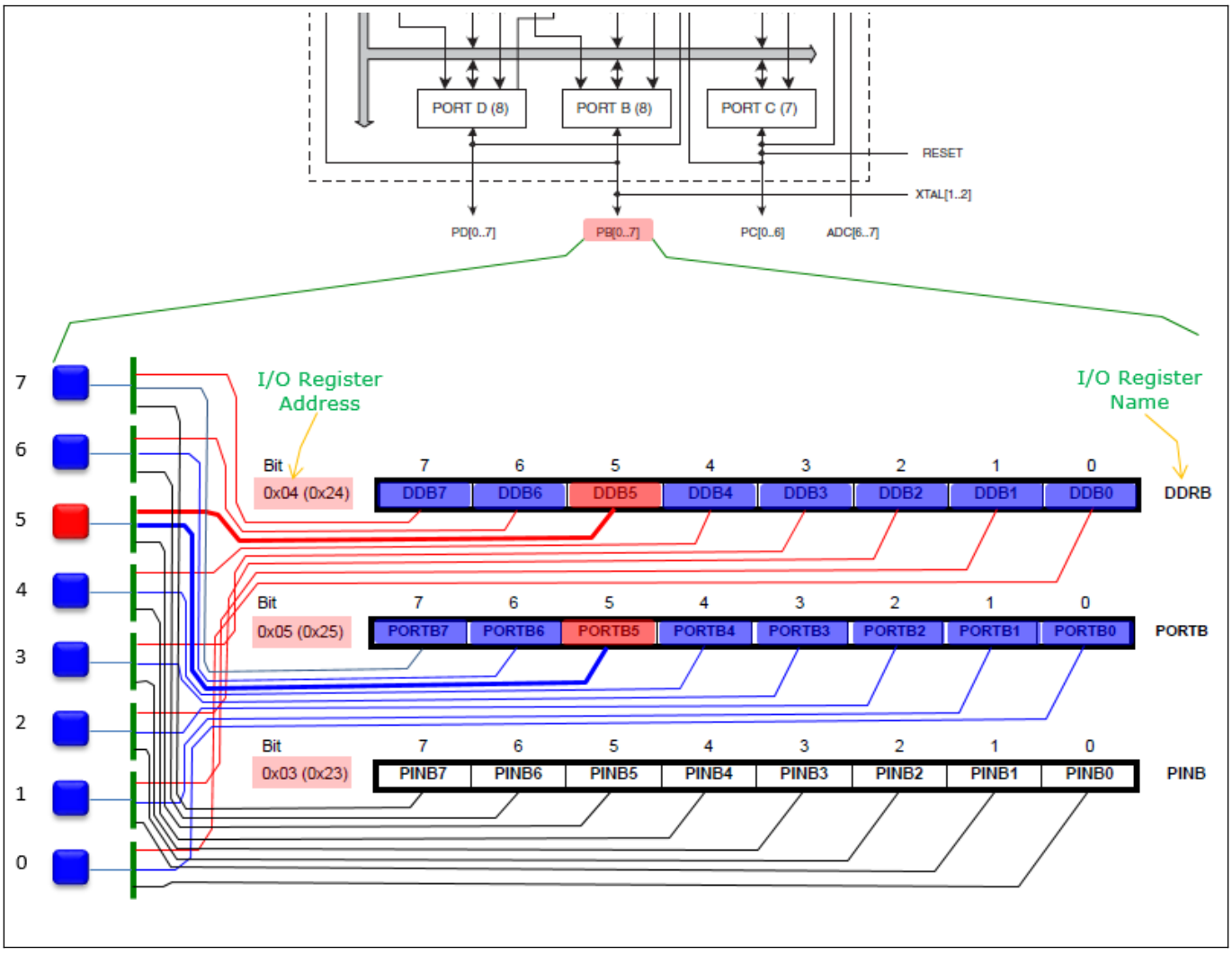
Company: Atmel Corporation

...

Dans le cas du protocole SPI ; ce sont les broches D8, D9, D10, D12 et D13 qui font partie du groupe B (PB0 à PB5).

À chacun de ces groupes est associé trois registres :

1. Port_x pour positionner l'état d'une broche (haut ou bas)
2. Pin_x pour lire l'état d'une broche
3. DDR_x (Data Dir. Reg.) pour positionner le sens d'une broche (Input ou Output)



A titre d'exemple, on peut re-écrire le programme « blink.ino » en utilisant les registres. Dans le cas de la LED interne, connectée en D13, c'est à dire Port_B(5), on peut modifier son état :

```
PORTB = B00100000 ; // mettre D13 à High

PORTB = B00000000 ; // mettre D13 à Low
```

Mais cette façon de procéder peut affecter les autres broches du groupe. On lui préférera l'utilisation d'un opérateur logique ; la fonction « OU » logique pour mettre un bit à 1 et la fonction « ET » logique pour mettre un bit à 0 :

```
PORTB |= B00100000 ; // « OU » bit à bit sur D13 à High

PORTB &= B00100000 ; // « ET » bit à bit sur D13 à Low
```

Il ne faut pas oublier de positionner D13 en sortie (mode Output) en utilisant le registre DDR :

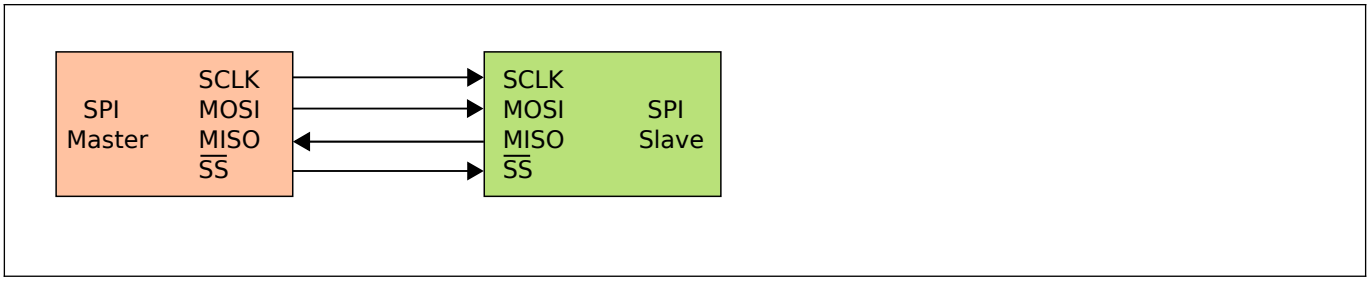
```
DDRB |= B00100000 ; // force bit 5 (D13) en mode Output
```

On assemble le tout :

```
void setup() {
// set pin 13 as output pin
DDRB |= B00100000;
}
// the loop function runs over and over again forever
void loop() {
PORTB |= B00100000;// turn the LED on (HIGH is the voltage level)
delay(800); // wait for a 0.8 second
PORTB &= B00000000; // turn the LED off by making the voltage LOW
delay(2000); // wait for a 2 second
}
```

Fonctionnement du protocole SPI :

Une liaison SPI (pour Serial Peripheral Interface) est un bus de données série synchrone baptisé ainsi par Motorola, au milieu des années 1980 qui opère en mode full-duplex. Les circuits communiquent selon un schéma maître-esclave, où le maître contrôle la communication. Plusieurs esclaves peuvent coexister sur un même bus, dans ce cas, la sélection du destinataire se fait par une ligne dédiée entre le maître et l'esclave appelée « Slave Select (SS) ».



Le bus SPI utilise quatre signaux logiques :

- SCLK — Serial Clock, Horloge (généré par le maître)
- MOSI — Master Output, Slave Input (généré par le maître)
- MISO — Master Input, Slave Output (généré par l'esclave)
- SS — Slave Select, Actif à l'état bas (généré par le maître)

Il existe d'autres noms qui sont souvent utilisés :

- SCK, SCL — Horloge (généré par le maître)
- SDI, DI, SI — Serial Data IN, MISO
- SDO, SDA, DO, SO — Serial Data OUT, MOSI
- nCS, CS, nSS, STE, CSN— Slave Select

ARDUINOISP.ino

On regarde l'en-tête du fichier arduino principal : ARDUINOISP.ino

```
#include "Arduino.h"
```

Ce qui renvoie à un autre fichier de bibliothèque :

```
#include "SPI.h"
```

L'Arduino UNO possède une liaison SPI (SCLK = D13 ; MISO = D12 ; MOSI = D11 ; et SS =D10 ; et autres si nous avons plusieurs composants esclaves) et une bibliothèque SPI() qui permet de gérer les échanges d'informations entre la carte Arduino et les circuits SPI connectés. Cette bibliothèque est installée de base dans l'IDE Arduino. Ses principales fonctions sont :

```
SPI.begin() ;
```

Qui initialise la communication sur le bus.

```
SPI.setBitOrder(sens) ;
```

Qui définit le premier bit qui est transmis où sens peut être LSBFIRST si le bit de poids faible est transmis en premier ou MSBFIRST si bit de poids fort est transmis en premier.

```
SPI.setClockDivider(diviseur) ;
```

Qui définit la fréquence d'horloge où diviseur est un multiple de 2 et peut prendre les valeurs SPI_CLOCK_DIV2 à SPI_CLOCK_DIV128.

```
SPI.setDataMode(mode) ;
```

: définit le mode de fonctionnement du bus où mode peut prendre les valeurs SPI_MODE0 à SPI_MODE3.

```
SPI.transfer(octet) ;
```

Qui envoie l'octet sur la ligne MOSI ou reçoit l'octet sur la ligne MISO.

```
SPI.end() ;
```

Qui arrête la communication.

Bon, on arrête ici pour la théorie ; mais certains aspects seront étudiés dans un prochain TP.

VI. Interprétation des Résultats

1) En date du 19 septembre 2023, le premier essai est concluant avec un programme « Blink » modifié sur le microcontrôleur « Cible ».
2) Dès le lendemain, la réalisation d'une platine avec supports ZIF (Zero Insertion Force) est réalisée pour faciliter les remplacements de composants sans les abîmer.
3) Puisqu'il y a deux opérations distinctes :
   i. Installation d'un Bootlaoder
   ii. Installation d'un programme applicatif

⇒ Il faut donc prévoir deux platines dédiées.

Compte tenu du succès de l'expérimentation, mais aussi des difficultés de mise en œuvre ; une « ligne de production » n'est pas envisageable pour le moment. Il faut donc étudier des solutions plus pratiques. De plus il y a des risques d'erreur (Placement de la broche n°1) qu'il faut absolument supprimer.

VII. Conclusion

On envisage déjà une suite à ce TP avec une étude préalable dans le but de fiabiliser les montages.

VIII. ADDENDUM

If you want to use an Arduino with its entire GUI infrastructure, libraries, etc. then it's best to use C/C++. You can mix C and assembly using the "asm" pseudo-function. By using the "naked" attribute for functions, you can write your entire program in assembly code, but from within a C file so that the compiler takes care of all the labels, section directives, etc. For example:

```
void somefunction(void) __attribute__((naked))
{
  asm volatile ("
 ; your assembly code here
  ");
}
```

Final word of advice: it's really not worth it. I understand the desire to be as close to the machine as possible, but the AVR architecture makes hand-assembly not that much less efficient than C-language-generated code. Unlike other processors (mostly from the past) where a 10-to-1 improvement is fully expected when comparing higher-level languages to assembly, in my experiences this just no longer holds. The underlying architecture is so C-friendly (lots of registers, orthogonal instructions) that the compiler-generated code is really good.