Thème n°1: Introduction aux capteurs intelligents et à la technologie arduino

CELECT7 : Applications des microcontrôleurs

M. Zwingelstein

rev. 2020

Travail préparatoire

Lectures de la partie préparatoire :

L1. Lire attentivement la page [ici] sur l'utilisation des opérations binaires (bit-wise) et booléens. Vous devez maîtriser les opérations de masquage sur les mots binaires, et bien comprendre la différence entre opérateurs binaires et opérateurs booléens. Si besoin, vous pouvez également vous aider de la page wikipedia [ici]

L2. Télécharger la datasheet du micro contrôleur ATmega 328p [ici] et lire les deux premières pages (« Features ») ainsi que le chapitre 2 (« Overview »).

Questionnaire de la partie préparatoire :

Q1. Questions sur les opérations binaires et sur les masques

- Ecrire l'opération permettant de forcer à 1 les bits d'indice 2 et 4 d'un mot binaire *w* tout en conservant les valeurs de autres bits
- Ecrire l'opération permettant de forcer à 0 les bits d'incice 2 et 4 d'un mot binaire *w* tout en conservant les valeurs des autres bits
- Ecrire l'opération permettant de stocker le 4ème bit de la variable `x` dans `y` (`y` vaut 0 ou 1)
- Que vaut `y` dans `y = x & ((1<<3)-1);` si `x = 0B01011011` et que les variable sont des variable entières signées sur 8 bits ? (on rappelle que -1 = 0B11111111)

Q2. Questions sur l'architecture de l'ATmega 328p (rechercher dans l'ensemble de la datasheet les réponses aux questions en vous aidant de Ctrl+F pour rechercher des mots-clé dans le pdf)

- Que signifie l'acronyme *ALU*?
- Quels sont les trois types d'opération que réalise une *ALU* ? ?
- Quels types de données sont stockées dans la pile (*stack*) ? (voir section 6.5 : Stack pointer)
- Dans quelle mémoire la pile est-elle physiquement implémentée ?
- Quel intérêt y-a-t-il à stocker les variables dans la pile plutôt que dans l'EEPROM ?
- Le *watchdogtimer* (WDT) peut être utilisé dans deux modes: « interrupt » et « reset ». Quel est l'usage typique du *watchdog timer* en mode reset ?

Q3. Rechercher chez le fournisseur Sparkfun [ici] un exemple de :

- Capteur analogique,
- Capteur numérique avec interface I2C

— Capteur numérique avec interface SPI.

Expliquer sur quel principe physique repose chacun de ces capteurs.

Consigner vos résultats dans un tableau récapitulatif : nom du capteur, nature de l'interface, principe physique.

Cours 1: Notion de capteur intelligent

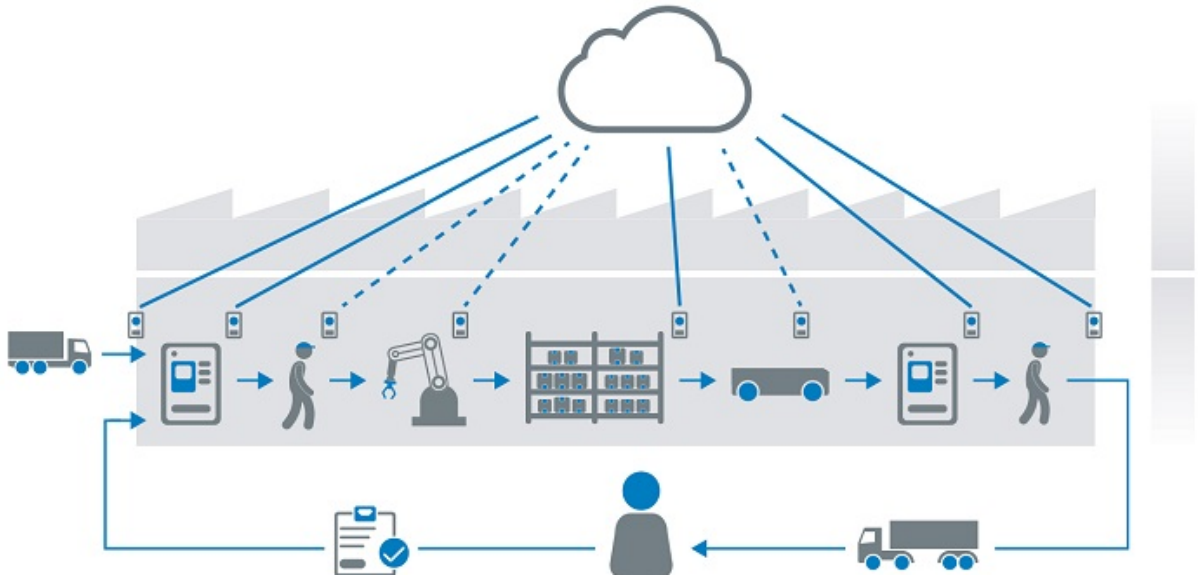
Présentation

Les « capteurs intelligents » sont un concept basé sur l'utilisation conjointe de capteurs, d'électronique numérique pour les fonctions de calcul et d'électronique radio-fréquence pour les fonctions de communication.

Ils représentent un dispositif essentiel pour :

La production inustrielle : IoT industriel, industrie 4.0, industrie connectée :

- Capteurs installés sur des machines de production pour mesurer, analyser, communiquer
  - optimisation du taux d'occupation des machines (cadence de fabrication adaptée en temps réel)
  - optimisation des délais de maintenance par un diagnostic rapide et à distance des machines

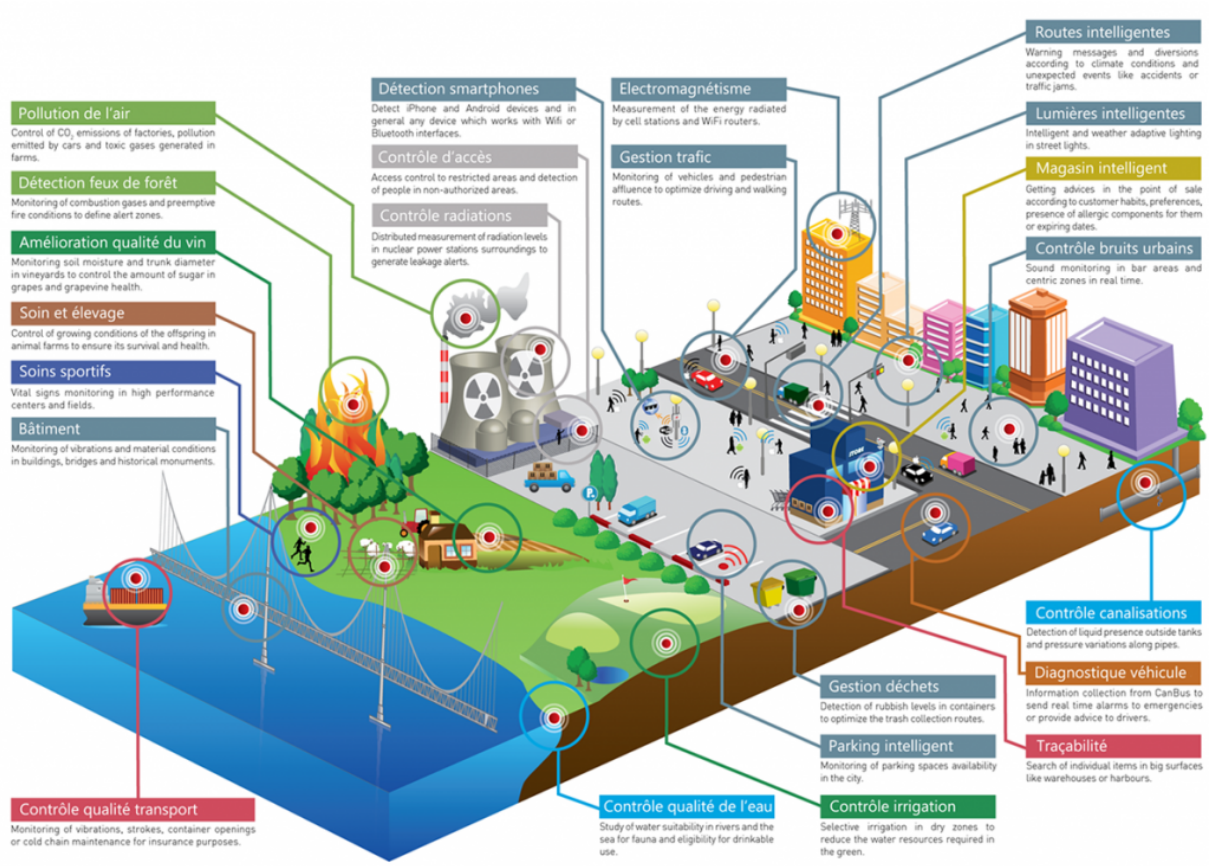


La maintenance prédictive qui devient de la maintenance prédictive *connectée*

- La SNCF déploie des capteurs sur 50 000 kilomètres de voies, dans 40 000 centres techniques, sur 2 200 systèmes d'aiguillages, et dans l'ensemble des gares et des rames de son réseau
  - suivi des usures mécaniques (caténaires, enfoncement de la voie sur le ballast…)
  - surveillance géolocalisée (ex : température des rails)
  - …

— Diminution des coûts par 10

La ville intelligente



Le futur :

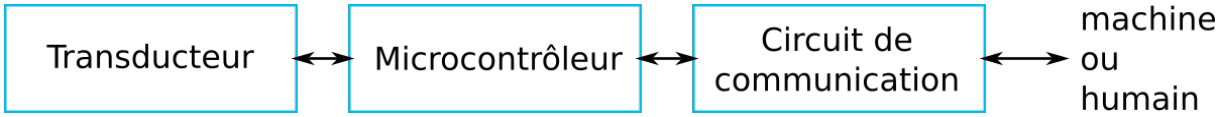
— [projet Man Machine Teaming] qui explore la possibilité de développer un Système Aérien Cognitif de Combat où la relation homme-machine serait élargie par l'usage de machines autonomes grâce à l'IA.
-> Capteurs cognitifs : apprenants et auto-adaptatifs

Point faible des capteurs intelligents

Sécurité (cyberattaques), du fait de la communication (risque accru pour les communications sans fil et les capteurs connectés à internet).

Architecture d'un capteur intelligent

3 éléments:



Le transducteur (ou capteur)

- Capte une grandeur physique et la transforme en signal porteur d'information (en général, signal électrique)
- Est utilisé à des fins de mesure ou de commande
- Certains capteurs fournissent un signal *analogique*,
- D'autres intègrent un *CAN* (Convertisseur Analogique Numérique) et s'interfacent avec le microcontrôleur au moyen d'une *interface série* (UART/USART, I2C, SPI, 1-Wire…)

Vous avez déjà pu voir toute la diversité des capteurs disponibles ainsi que leur documentation chez le fournisseur Sparkfun : www.sparkfun.com/categories/23

- Les transducteurs *actifs* fonctionnent en générateurs et sont basés sur des effets physiques de transformation de l'énergie de la grandeur à mesurée en énergie électrique
  - effet thermoélectrique
  - effet piezzo-électrique
  - effet d'induction électromagnétique
  - effet photo-électrique
  - effet Hall
  - effet photovoltaïque
- Les transducteurs *passifs* sont équivalents à une *impédance* dont la valeur est sensible à la grandeur physique mesurée, et nécessitent un circuit électrique alimenté
  - inductance à noyau mobile
  - condensateur à armature mobile
  - jauge d'extensiométrie
  - …

Le microcontrôleur

Bases vues en 2A :

- **Architecture** (ALU, registres, interruptions, timers…)

- Programmation assembleur
- **Programmation C** (vue en 1A)

(en **gras** : prérequis)

Le circuit de communication

Nombreuses possibilités, *en réseau* ou *point à point*.

Exemples de systèmes de communication radio :

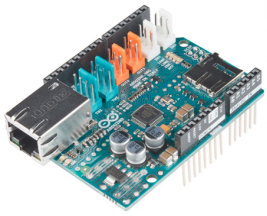
- LAN (Local Area Network)
  - WiFi
  - Bluetooth
  - Zigbee (adapté aux réseaux de capteurs sans fil)
- WAN (Wide Area Network)
  - réseaux cellulaires : 3G, 4G
- LPWAN (Low Power WAN)
  - Lora, Sigfox, LoraWAN (propre à l'IoT)
- Point à point
  - NFC (Near Field Communications)
  - Communications sub-GHz. Ex : circuit Texas Instrument CC1101 - bandes ISM

Cours 2: L'environnement arduino

Définition

- L'environnement arduino a été **à l'origine** conçu pour mettre **à la portée de tous** la technologie des microcontrôleurs.
  Cela a été obtenu notamment en réalisant une *abstraction* de tout le détail interne de l'architecture du microcontrôleur (périphériques, registres…).
- On parle d'*environnement arduino* car cela comprend :
  - un ensemble de cartes de développement à base de uC (AVR mais aussi ARM, esp…)
  - un IDE (Integrated Development Environment)
  - des bibliothèques (*couche d'abstraction* arduino)
- **Aujourd'hui**, arduino est utilisé par des communautés très diverses allant du domaine amateur (makers, DIY) au domaine professionnel (industriel ou académique/recherche) notamment pour tout ce qui concerne le **prototypage rapide**.

UNO board
8-bits AVR ATmega328p
horloge 16 MHz
FLASH 32 KB
SRAM 2KB



ethernet shield

zero board
32-bits ARM Cortex M0+
horloge 48 MHz
FLASH 256 KB
SRAM 32 KB

industrial 101 board

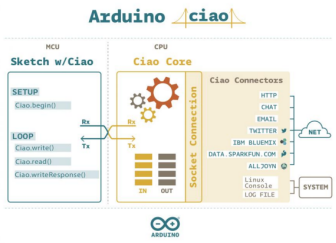


FIGURE 1 – Illustration très partielle de la diversité des cartes de développement arduino.

Abstraction logicielle d'arduino

L'abstraction logicielle est réalisée par les bibliothèques arduino dont le code source peut être vu sur github [ici] (ou bien directement dans l'arborescence du répertoire d'installation d'arduino).

```ino
/*
 *                      Arduino style exemple
*/

// the setup function runs once when you press reset or power the board
void setup() {
  pinMode(13, OUTPUT);  // initialize digital pin 13 as an output.
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn pin 2 HIGH
  delay(1000);              // wait for 1 s
  digitalWrite(13, LOW);
  delay(1000);
}
```

```c
/*
 *                      C style exemple
*/

# define F_CPU 16000000UL // necessary for the delay function
# include <avr/io.h>
# include <util/delay.h>

int main(void) {

    DDRB = 0b00100000;  // DDRB : Data Direction Register of PORTB

    while(1) {
        PORTB = 0b00100000; // turns on pin 5 in port B
        _delay_ms(1000);    // wait 1 s
        PORTB = 0b00000000;
        _delay_ms(1000);
    }
}
```

FIGURE 2 – comparaison d'un code arduino avec un code C classique

Il est utile de remarquer sur la figure 3 que la broche arduino numéro 13 correspond à la proche PB5 du uC.

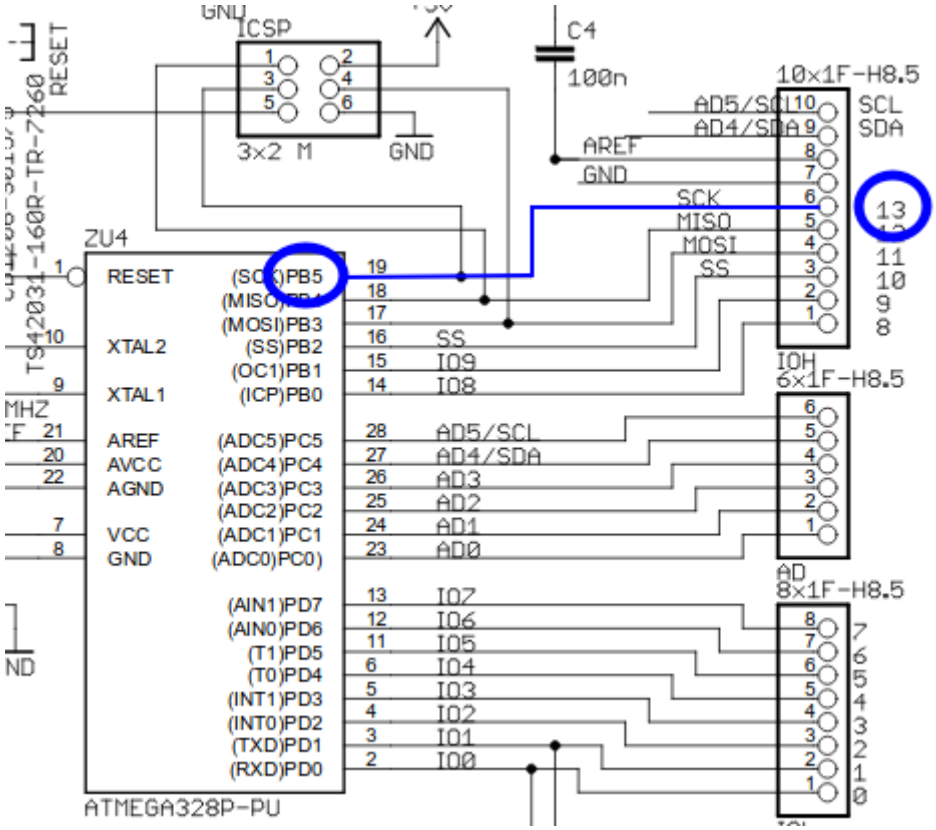


FIGURE 3 – Schéma partiel de la carte arduino uno à base d'ATmega328p

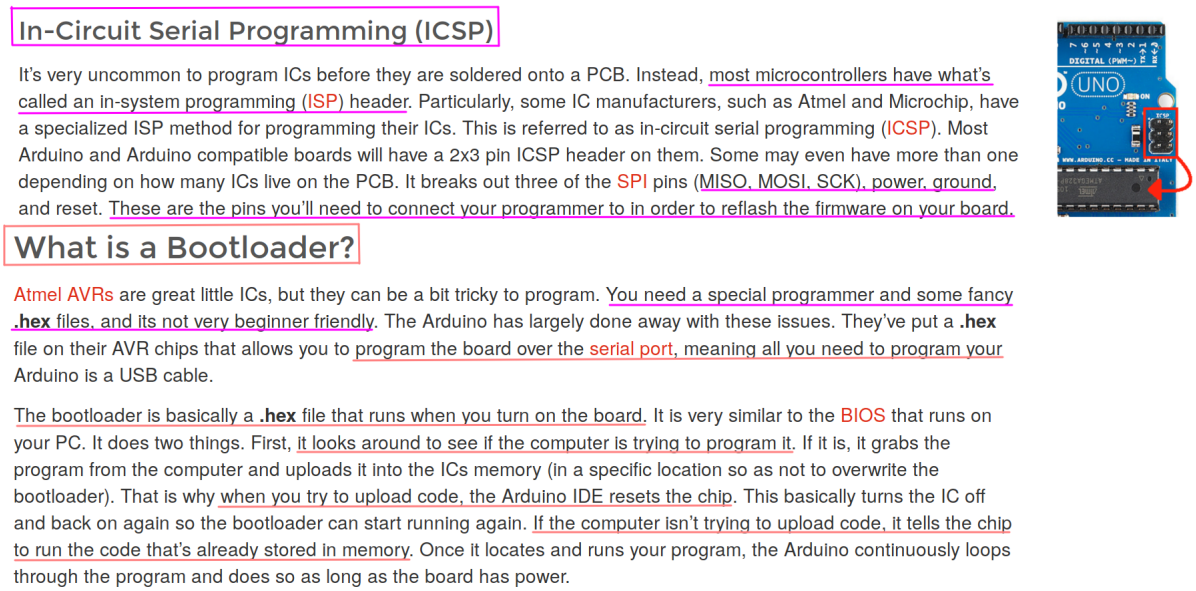
Bootloader

Utilité

— Permet de s'affranchir d'un programmeur en programmant le uC directement depuis l'IDE arduino

Principe de fonctionnement

— Bootloader = programme situé dans la mémoire flash du uC, automatiquement exécuté lors de sa mise sous tension (à l'image d'un BIOS)
  — détecte via la liaison série si l'IDE arduino souhaite programmer le uC
  — télécharge le programme via le lien série et l'écrit dans la flash du uC



In-Circuit Serial Programming (ICSP)

It's very uncommon to program ICs before they are soldered onto a PCB. Instead, most microcontrollers have what's called an in-system programming (ISP) header. Particularly, some IC manufacturers, such as Atmel and Microchip, have a specialized ISP method for programming their ICs. This is referred to as in-circuit serial programming (ICSP). Most Arduino and Arduino compatible boards will have a 2x3 pin ICSP header on them. Some may even have more than one depending on how many ICs live on the PCB. It breaks out three of the SPI pins (MISO, MOSI, SCK), power, ground, and reset. These are the pins you'll need to connect your programmer to in order to reflash the firmware on your board.

What is a Bootloader?

Atmel AVRs are great little ICs, but they can be a bit tricky to program. You need a special programmer and some fancy **.hex** files, and its not very beginner friendly. The Arduino has largely done away with these issues. They've put a **.hex** file on their AVR chips that allows you to program the board over the serial port, meaning all you need to program your Arduino is a USB cable.

The bootloader is basically a **.hex** file that runs when you turn on the board. It is very similar to the BIOS that runs on your PC. It does two things. First, it looks around to see if the computer is trying to program it. If it is, it grabs the program from the computer and uploads it into the ICs memory (in a specific location so as not to overwrite the bootloader). That is why when you try to upload code, the Arduino IDE resets the chip. This basically turns the IC off and back on again so the bootloader can start running again. If the computer isn't trying to upload code, it tells the chip to run the code that's already stored in memory. Once it locates and runs your program, the Arduino continuously loops through the program and does so as long as the board has power.

FIGURE 4 – Programmation ISP vs Bootloader. Source : https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader#what-is-a-bootloader

Avantages de l'Arduino

— Bas coût (20 EUR pour une carte UNO à base de uC AVR ATmega328p)
— Open-source (matériel et logiciel)
— Facilité de prise en main grâce à l'abstraction logicielle et au bootloader
— Grande diversité de uC (AVR, ARM…)

— Nombreux *shields* disponibles avec les bibliothèques associées
— Très large communauté

Inconvénients de l'Arduino

— Comme conséquence du caractère « user friendly » du code arduino, il arrive parfois que le code soit difficile à debugger (on ne maîtrise pas tout)
— Pour des performances accrues il peut être utile de court-circuiter la couche d'abstraction arduino en accédant directement aux registres du microcontrôleur (**voir lab**)
— Certaines applications ne peuvent pas être développées avec les fonctions et bibliothèques standard de l'IDE arduino

Pour conclure

Il ne s'agit pas de dire si arduino est bon ou mauvais, mais de trouver l'outil le plus efficace pour réaliser le travail.

Arduino répond très bien à la problématique de *prototypage rapide*.

Cours 3: Architecture du microcontrôleur ATmega328P :

Vue générale

Datasheet de l'ATmega : 660 pages! [lien]

A utiliser comme une **encyclopédie** (pas comme un roman).

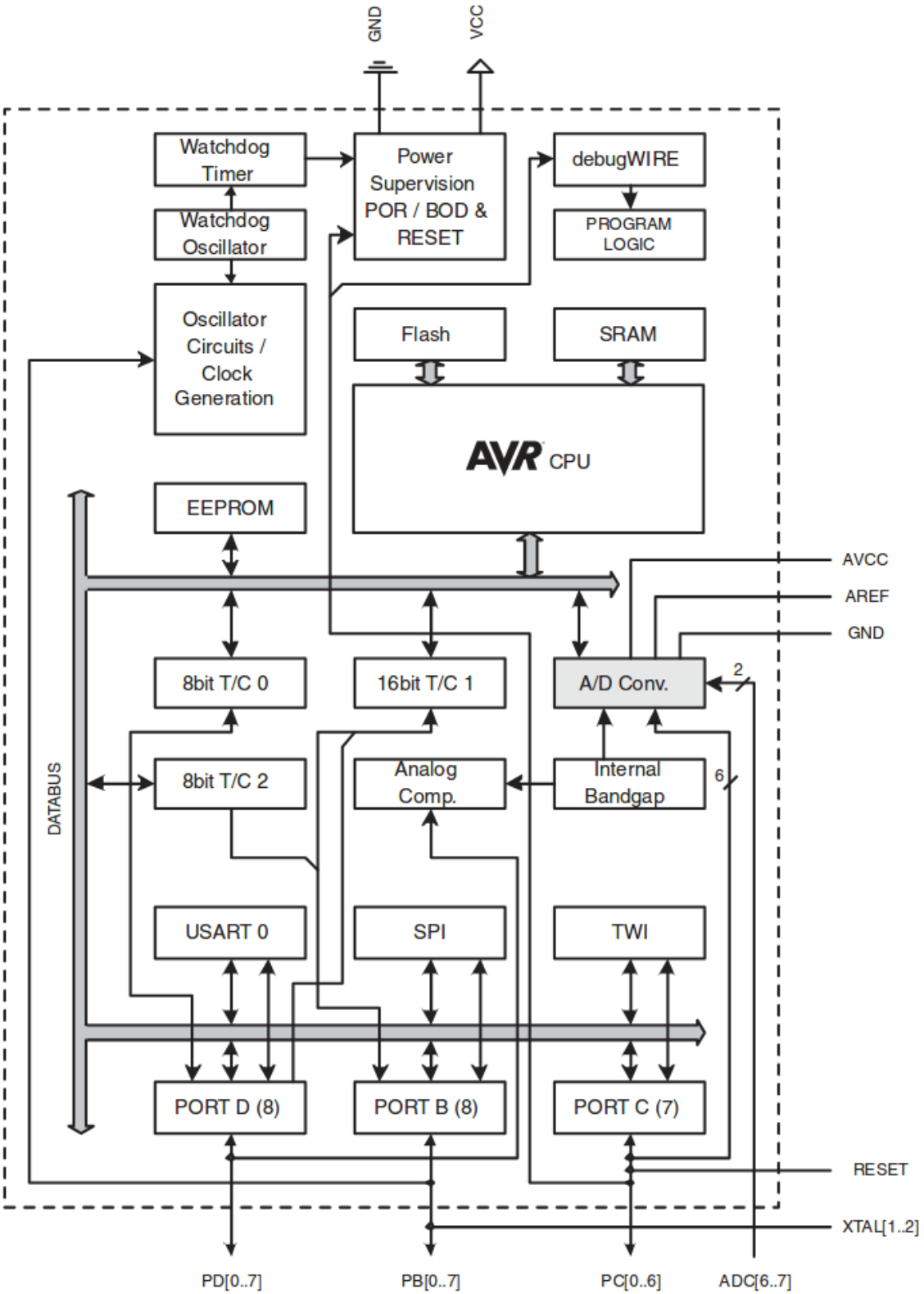


Figure 5 – diagramme en bloc

Eléments constitutifs

- CPU (Central Processing Unit).
- Mémoire du programme (Flash) et mémoire pour les variables (SRAM)
  - le programme, ainsi que tout ce qui n'est pas changé pendant l'exécution du programme sont stockés dans la mémoire flash
  - les variables modifiées pendant l'exécution du programme sont stockées dans la mémoire SRAM (qui contient la pile (stack))
  - une variable déclarée *volatile* sera toujours stockée dans la SRAM
- Timer/Counter (au nombre de trois dans les uC AVR ATmega) :
  - comptage, gestion du temps (timing), gestion d'évènements
  - servent à déclencher des interruptions
  - génération de signal PWM
- Mécanismes d'interruptions :
  - interruptions déclenchées par un évènement :
    - interne au uC. Ex: timer/counter atteignant une certaine valeur
    - externe. Ex: changement d'état d'une broche)
  - exécution d'une *routine d'interruption* (ISR: Interruption Sub-Routine) lors du déclenchement de l'interruption
  - les adresses des ISR associées aux différentes sortes d'interruptions sont stockées sous forme de *vecteur d'interruptions* dans la mémoire programme (Fig. 6).
- Serial I/O : UART, USART, SPI, TWI/I2C/IIC
- ADC (Analog to Digital Converter)
- EEPROM : mémoire non volatile en lecture/écriture

Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A
13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

FIGURE 6 – Vecteurs d'interruption de l'AVR 328p (section 11.1 de la datasheet).

Nous aurons l'occasion d'étudier plus en détail certains éléments de l'architecture de l'ATmega 328p dans la suite de ce cours CELECT7.

Exercice : Lecture de documentation technique

Enoncé

Consulter la section de la [datasheet] de l'ATmega328p portant sur les registres associés à la gestion des broches d'E/S, et répondre aux questions suivantes :

1. Il y a quatre ports d'E/S sur le microcontrôleur AVR 328p. Vrai ou Faux ? Nommer les ports d'E/S.
2. Il est écrit au début de la section 13.2: « The ports are bi-directional I/O ports with optional internal pull-ups ». Ainsi, chaque broche d'E/S peut être reliée à une résistance de tirage vers le haut (Vcc).

   Illustrer l'intérêt de la résitance de tirage en proposant deux schéma de câblage d'un interrupteur sur une broche d'E/S :
   - un schéma où résistance de tirage interne n'est pas activée (mais faisant apparaître une résistance externe)
   - un schéma supposant la résitance de tirage est activée.
3. Les trois registres associés au port D se nomment PORTD, DDRD et PIND.
   - Que signifie l'accronyme DDRD ?
   - Voici un descriptif du rôle des trois registres PORTD, DDRD et PIND, mais donnés dans un ordre aléatoire. Associer le numéro de la description au nom du registre qui lui correspond :
   - description n°1 :
     - contient l'état courant des broches (pins) du port (1 si la pin est à l'état haut, et 0 si la pin est à l'état bas)
   - description n°2 :
     - configure les broches (pins) d'E/S du port D en entrée (0) ou en sortie (1). La configuration initiale est 0 (entrée)
   - description n°3 :
     - si la broche (pin) est configurée en sortie, positionne la pin à l'état haut (1) ou bas (0)
     - si la broche (pin) est configurée en entrée, active (1) ou non (0) la résistance de pull-up interne

Labs

Présentation

- Lab 1
  - Utilisation des registres de configuration des broches d'E/S en lieu et place des fonctions arduino. Constatation du gain en mémoire programme
  - Manipulations binaires : décalages, masques
- Lab 2
  - Mise en évidence de certaines limites d'arduino et indications pour les contourner, à travers un exemple simple : génération d'une impulsion de durée fixe et précise sur une broche d'E/S
- Lab 3
  - Utilisation de l'IDE Atmel Studio pour programmer en langage C une carte arduino UNO

Lab 1

1. Compiler et téléverser le programme `seance1_a1.ino` fourni.

   — Noter la valeur de la taille du programme compilé (en octets),

   — Tester le programme : pour cela, ouvrir le terminal série depuis l'IDE arduino et vérifier que le débit est bien de 57600 bps comme paramétré dans le code arduino.
   Prendre un fil, relier un côté à la masse, et l'autre côté à une des broches d'entrée arduino du programme (2 à 9).

   Qu'observez-vous ?

   — Expliquer le fonctionnement détaillé du programme (lien entre les instructions, les valeurs observées et la configuration des broches)

2. Consulter le schéma électronique de carte arduino uno [ici] afin de :
   — Déterminer les ports et les numéros des E/S (entrées/sorties, I/O en anglais) associés aux numéros « arduino » des broches utilisés dans le programme seance1_a1.ino,
   — En déduire la configuration des registres DDRD, DDRB, PORTB et PORTD qui correspond aux lignes de code n°3 à n°10 du programme
      — PORTB:
      — PORTD
3. Enregistrer le programme `seance1_a1.ino` sous le nom `seance1_a2.ino`.
4. Modifier le programme en supprimant les lignes n°3 à n°10, et en les remplaçant par une configuration directe des registres DDRD, DDRB, PORTB et PORTD telle que vous l'avez déterminée à la question 2 :
   — Première façon :
      — `PORTD = 0B...`
      — `PORTB = 0B...`
      — `DDRD = 0B...`
      — `DDRB = 0B...`
   — Deuxième façon :
      — faire explicitement référence aux bits PORTD7, PORTD6, ..., PORTB0 des registres PORTD et PORTB ainsi qu'aux bits DDRD7, DDRD6, ..., DDRB0 des registres DDRD et DDRB.
      Vous utiliserez les opérateur de décalage (`<<`) et de OU logique (`|`).
      On rappelle que, par exemple, le bit PORTD6 étant en septième position à partir de la droite dans le registre PORTD, `(1<<PORTD6)` est équivalent à `1<<6`.
      — tester cette deuxième façon (mettre en commentaire la première façon dans le code).

   Pour chacune des deux façons, vérifier que le programme fonctionne correctement (c'est-à-dire

à l'identique de `seance1_a1.ino`), noter la valeur de la taille du programme (en octets) et calculer en % le gain obtenu par rapport au programme `seance1_a1.ino`.

5. Afin de gagner encore en taille de programme, vous allez modifier les lignes 14 à 22 du programme initial en accédant directement aux contenus des registres PIND et PINB.
   - Enregistrer le programme `seance1_a2.ino` sous le nom `seance1_a3.ino`.
   - Modifier les lignes de code correspondant aux lignes 14 à 22 du programme seance1_a1.ino en accédant directement aux contenus des registres PIND et PINB. Penser à utiliser un opérateur de décalage (<<, >>).
   - Vérifier que le programme fonctionne correctement (c'est-à-dire à l'identique de seance1_a1.ino), noter la valeur de la taille du programme (en octets) et calculer en % le gain obtenu par rapport au programme `seance1_a1.ino`.

Lab 2

1. Ouvrir le programme `seance1_b1.ino` fourni dans la carte arduino UNO.
   Dessiner sur une feuille le signal que le programme est sensé générer sur la broche 13 de la carte.
2. Compiler et téléverser le programme, puis noter la valeur de sa taille (en octets).
3. Régler l'oscilloscope de façon à observer avec le maximum de précision le signal effectivement généré. Vous devez observer deux différences avec le signal attendu :
   — la durée de l'impulsion est trop élevée
   — la durée de l'impulsion n'est pas constante

   Mesurer la durée min et la durée max de l'impulsion, et noter ces durées.
4. Il s'agit ici de *comprendre* l'origine de la première différence : la durée de l'impulsion est supérieure aux 50 us attendues. La deuxième différence relative à la fluctuation de la durée de l'impulsion sera traité dans un second temps.

   — Consulter la documentation arduino (sur le site arduino.cc) relatif à la fonction `delayMicroseconds()`.

   Quelle explication trouvez-vous par rapport au problème soulevé ?
   — Vérifier en écrivant la ligne `digitalWrite(13, HIGH)` une deuxième fois juste en dessous et en mesurant la durée min de l'impulsion, puis une troisième fois.

   Qu'en concluez-vous ?
   En fait, du code est automatiquement ajouté du fait de l'abstraction arduino de la fonction digitalWrite().
   — Consulter le fichier `wiring_digital.c` que vous trouverez dans l'arborescence de l'installation de l'IDE arduino, en recherchant là où est définie la fonction digitalWrite() (…/hardware/arduino/avr/cores/arduino).
   — Vous pouvez constater que le code est plutôt volumineux pour le seul objectif qui est de positionner une broche de l'ATmega 328p dans un état donné.
5. Il s'agit ici de *corriger* ce premier problème, en réalisant l'impulsion *sans passer* par la fonction `digitalWrite()`
   — Enregistrer le programme `seance1_b1.ino` sous le nom `seance1_b2.ino`.
   — En vous appuyant sur ce que vous avez appris au Lab 1 relativement au rôle des registres liés aux ports d'E/S, modifier le programme en remplaçant l'utilisation de la fonction `digitalWrite()` par une configuration adéquate du registre PORTB. Vous utiliserez des

masques | (OU) et & (ET) afin de forcer respectivement à 1 et à 0 le bit du registre PORTB correspondant à la broche « arduino » 13, sans changer la configuration des autres broches du port B.

— Compiler et téléverser le programme `seance1_b2.ino`.
— Observer à l'oscilloscope l'impulsion générée. Commenter.

6. Il s'agit ici d'*analyser* la deuxième différence : impulsion de durée non déterministe.

Origine du problème : l'abstraction de la fonction `delayMicroseconds()` (le même problème existe avec les fonctions `millis()` et `delay()`):

— Ces fonctions exécutent périodiquement un code mettant à jour la conversion en unité lisible (ms) du délai écoulé (utile pour faciliter la compréhension humaine)
— C'est le *timer0* qui est utilisé afin de cadencer cette mise à jour selon le principe suivant :
  — chaque 1024 us, le timer0 crée un *overflow* qui déclenche une *interruption*,
  — la routine d'interruption ratachée à cette interruption gère la mise à jour de la conversion en unité lisible.

Vous pouvez **faire vous même cette analyse** en procédant de la façon suivante :

— Ouvrir le fichier `main.cpp` disponible dans l'arborescence du répertoire d'installation du logiciel arduino (. . . /hardware/arduino/avr/cores/arduino), et noter qu'avant la foncion `setup()`, une fonction `init()` est exécutée.
— Ouvrir le fichier `wiring.c` afin d'observer le code de cette fonction `init()`.
Observer la ligne : `//enable timer 0 overflow interrupt`
Le code qui suit ce commentaire sert à autoriser les interruptions déclenchées par l'overflow du timer0.
— Au début de ce même fichier, observer le code qui suit la ligne : `ISR(TIMER0_OVF_vect)`.
Il s'agit de la routine d'interruption, exécutée à chaque overflow du timer0, et qui effectue la mise à jour de la conversion en unité lisible, et qui prend un certain temps.
— Au vue des explications ci-dessus synthétiser en une ou deux phrases claires et concises la raison pour laquelle la durée de l'impulsion fluctue.

7. *Résolution* du second problème : impulsion de durée non déterministe

Une solution au second problème est de désactiver l'interruption d'overflow du timer0 après que la fonction `init()`, obligatoirement présente, ait activé cette interruption. Ainsi, la routine d'interruption `ISR(TIMER0_OVF_vect)` ne s'exécutera plus.

Comment faire ?

— On peut lire à la section 14.9.6 de la datasheet le rôle du bit TOIE0 du registre TIMSK0.
Résumer ce rôle en une phrase.

```
1   <!-- permet d'activer/désactiver l'interruption d'overflow du
         timer0. Il suffit donc de configurer correctement ce bit
        dans le setup() du code arduino. -->
```

— Expliquer ligne par ligne le code ci-dessous.

```
cli();
TIMSK0 &= ~(1<<TOIE0);
sei();
```

— Expliquer en quoi ce code peut résoudre le problème de fluctuation de durée de l'impulsion, et à quel endroit du code arduino il faut le placer (justifier).

— Enregistrer le programme `seance1_b2.ino` sous le nom `seance1_b3.ino` et apporter la modification proposée,

— Compiler et téléverser le programme `seance1_b3.ino`.

— Observer à l'oscilloscope l'impulsion générée. Commenter quant à l'efficacité de la solution.

7. Optimisation de la taille du programme

— Afin d'optimiser la taille du programme, dans le `setup()`, configurer directement le registre DDRB au lieu d'utiliser la fonction `pinMode()`, comme cela avait été fait dans le lab1.
Noter la valeur de la taille du programme (en octets).

— Calculer en % le gain obtenu par rapport au programme `seance1_b1.ino`.

Lab 3

Compilation C et programmation de la carte arduino depuis l'IDE Atmel Studio

Vous allez à présent utiliser l'IDE *Atmel Studio* afin de programmer la carte arduino UNO. Pour cela, le code doit être modifié pour utiliser une syntaxe 100% langage C, et plus du tout de code de l'abstraction arduino.

1. Ouvrir l'IDE Atmel Studio (si on vous le demande à l'ouverture, utiliser le mode *avancé*), puis créer un nouveau projet (de type GCC C executable project) et lui associer le microcontrôleur ATmega 328p
2. Reprendre le programme `seance1_a3.ino` et l' adapter en tenant compte des consignes suivantes. Vous nommerez le fichier seance1_a3.c.
   - le type arduino **`byte`** est en fait un alias du type C `uint8_t` et doit donc être modifié en conséquence
   - les fonctions arduino `delay()` et `delayMicroseconds()` doivent être remplacées respectivement par `_delay_ms()` et `_delay_us()`
   - ne pas transmettre la valeur de la variable `inputs` sur le port série
   - ajouter la fonctionnalité suivante: une led connectée à la broche 10 devra s'allumer si la variable `inputs` est impaire
3. Compiler (menu `Build\Build Solution`)
4. Noter la valeur de la taille du programme (en octets), et calculer en % le gain obtenu par rapport au programme `seance1_a1.ino`.
5. Programmer (menu `Tools\Arduino_uno_via_bootloader`). Si ce menu n'existe pas, sachez que la façon de configurer Atmel Studio afin que ce menu soit disponible est expliquée dans cette vidéo ici. Demander à l'enseignant de vous guider pour gagner du temps.

Bibliographie :

[Tech. Ing. 2015]

[Sick]

[Intesens]

[Ermeo]

[Zdnet SNCF]

[projet Man Machine Teaming]