

# Apprendre à utiliser un capteur de pluie sur Arduino en langage C

Par Francesco Balducci  - f-leb (traducteur)

Date de publication : 16 décembre 2017

CONFIRMÉ

Ce tutoriel, le cinquième **de la série**, montre un nouvel exemple de **programmation de carte Arduino en langage C**, sans passer par le « langage Arduino » proposé dans l'EDI standard.

- Partie 1 : **Programmer l'Arduino en langage C**
- Partie 2 : **Utiliser un buzzer avec Arduino en langage C**
- Partie 3 : **Apprendre à dessiner sur une matrice de LED avec Arduino en langage C**
- Partie 4 : **Apprendre à utiliser les interruptions sur Arduino en langage C**

À chaque fois, l'idée est de s'affranchir des facilités offertes par le fameux « **langage Arduino** » dans l'EDI standard. Ici, les programmes seront développés en langage C « pur » grâce aux outils de la chaîne de compilation *avr-gcc*.

L'objectif est double :

- développer des codes optimisés, efficaces et compacts ;
- démystifier le fonctionnement d'un microcontrôleur et prendre le contrôle des entrées-sorties, sans fard, en attaquant directement les registres du microcontrôleur.

Cette fois, l'auteur a voulu interfacer l'Arduino avec un capteur de pluie. Cet article décrit une application qui aura les fonctionnalités suivantes :

- lire un signal sur une broche d'entrée numérique (facile) ;
- acquérir un signal analogique avec le CAN (Convertisseur Analogique-Numérique) ;
- faire varier la luminosité d'une LED avec la PWM (*Pulse Width Modulation* ou Modulation en Largeur d'Impulsion), et visualiser le niveau du signal analogique ;
- transmettre les données à un terminal d'un PC via une liaison série en utilisant l'USART, et visualiser le niveau du signal analogique.

Un espace vous est proposé sur le forum pour réagir sur ce tutoriel.

**Commentez**

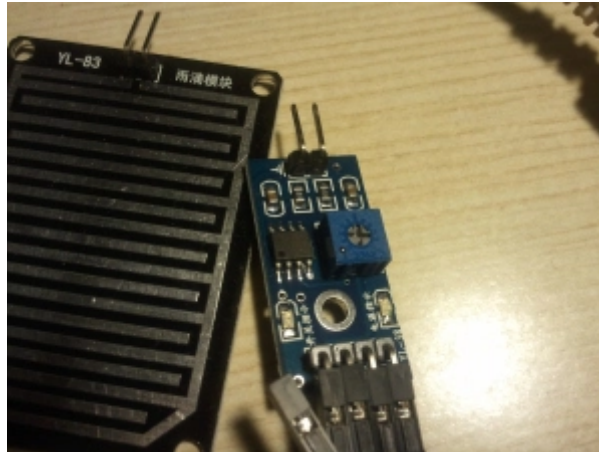
En complément sur Developpez.com

- [Programmer l'Arduino en langage C](#)

I - Interfacer le capteur de pluie.....	4
II - Lecture du signal analogique d'humidité et du signal numérique de détection de pluie.....	5
III - Faire varier la luminosité d'une LED avec un signal PWM.....	6
IV - Configuration de l'USART.....	6
V - Écriture du programme.....	7
VI - Génération, téléversement et exécution.....	8
VII - Conclusion.....	9
VIII - Notes de la Rédaction Developpez.com.....	9

## I - Interfacer le capteur de pluie

La première étape consiste d'abord à identifier les caractéristiques du capteur de pluie utilisé, car ses origines ne sont pas connues sauf si on fait exception du fait que des caractères chinois sont gravés dessus.



Capteur de pluie 100 YL-83

En photographiant ces lettres pour les passer à un logiciel de reconnaissance de caractères, on retrouve ceci : ### #, que Google traduit par « *raindrop module* ». Après des recherches avec la référence YL-83, [la page de référence officielle du capteur de pluie 100Y YI-83](#) a pu être trouvée avec ses caractéristiques. La documentation contient les informations suivantes, dont on recopie un extrait traduit ici pour plus de commodité :

### Caractéristiques

- Alimentation : 5 V.
- LED d'indication d'alimentation.
- LED d'indication de sortie.
- Sortie au niveau TTL, la sortie TTL au niveau bas a une capacité de 100 mA et peut commander directement un relais, un buzzer, un petit ventilateur, etc.
- Le seuil de détection est ajustable par un potentiomètre.
- En l'absence de pluie, la sortie est au niveau haut et la LED est éteinte.
- Les circuits de contrôle et de détection sont distincts et reliés par des fils pour une grande facilité d'installation.
- Circuit avec une grande surface pour une meilleure détection de pluie.
- Les circuits comportent des trous pour une fixation simple par vis.
- Dimension du circuit de contrôle : 3 x 1,6 cm.
- Dimension du circuit de détection de pluie : 5,4 x 4,0 cm.

En étudiant les schémas, on comprend les choses suivantes :

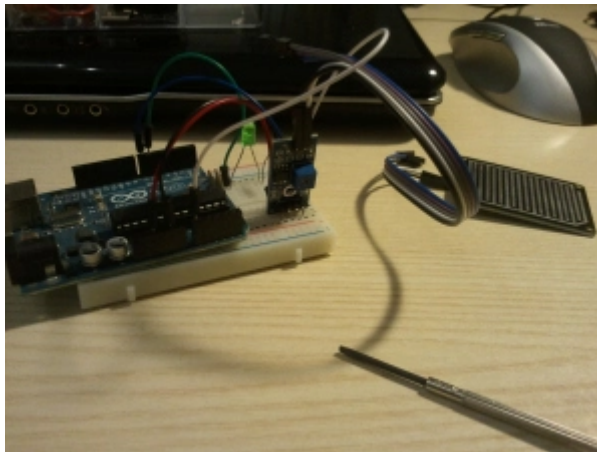
- le capteur est un dipôle résistif dont la valeur de résistance est faible à l'état humide et élevée à l'état sec ;
- il y a un diviseur résistif qui produit en sortie une faible tension lorsque le capteur est humide, et une tension plus élevée lorsque le capteur est sec. Cette tension analogique est l'image du niveau d'humidité ;
- la tension image du niveau d'humidité est comparée avec une tension de référence (IN) avec un amplificateur opérationnel en mode comparateur qui produit en sortie un signal « numérique » (OUT) à l'état haut (VCC) lorsque le capteur est sec et à l'état bas (GND) lorsque le capteur est humide ;
- la tension de référence (IN) peut être ajustée avec un potentiomètre ;
- le connecteur à quatre broches est utilisé pour alimenter le module avec VCC (5 V) et GND, pour récupérer le niveau « analogique » d'humidité (AC) et l'indicateur de pluie « numérique » (OUT) ;

- une LED bleue (D1) indique que le module est alimenté, et une LED rouge (D2) s'allume lorsque de la pluie est détectée (lorsque OUT est à l'état bas).

On connecte le module à l'Arduino Uno de la façon suivante :

- VCC au 5 V ;
- GND au... GND ;
- OUT au connecteur 7, c'est-à-dire la broche PD7 de l'ATmega ;
- AC au connecteur A0, c'est-à-dire la broche PC0 (ADC0) de l'ATmega.

Une LED est également placée sur le connecteur 6, c'est-à-dire la broche PD6 (OC0A) de l'ATmega afin de visualiser le niveau du signal analogique grâce au signal PWM.



*Arduino Uno connecté avec le capteur de pluie YL-83*

## II - Lecture du signal analogique d'humidité et du signal numérique de détection de pluie

L'ATmega comprend un ADC (NDLR *Analogic Digital Converter* ou Convertisseur analogique-numérique) qui peut être configuré pour lire une des broches « analogiques » de la puce, dans notre cas ADC0, et convertir l'entrée en une valeur numérique sur 10 bits. La valeur sur 10 bits est présentée dans deux registres 8 bits contenant la partie de poids faible (ADCL) et la partie de poids fort (ADCH). Dans mon cas, 8 bits devraient largement suffire, et l'ADC est donc configuré de sorte que 8 bits soient dans le registre de poids fort ADCH et 2 bits dans le registre de poids faible ADCL qui ne sera pas lu. Étant donné que le timing n'est pas un problème, la gestion de la conversion est simplifiée et la lecture est « bloquante » : la conversion est lancée et une boucle attend qu'elle soit terminée. La tension de référence de l'ADC est AVCC, qui est reliée au 5 V sur la carte Arduino Uno. De cette façon, on peut lire le signal AC du capteur de pluie depuis le registre ADCH, avec une valeur comprise entre 0 et 255 (0xFF).

Le signal OUT de détection de pluie peut être lu directement depuis le bit PIND7 du registre PIND, soit à 1 soit à 0. Voici un bout de code qui lit les données du capteur :

```

#define BVV(bit, val) ((val)?_BV(bit):0)

static void rain_init(void)
{
    DDRD &= ~_BV(DDD7); /* OUT pin connected to PORTD7 */
    DDRC &= ~_BV(DDC0); /* Analogue pin connected to PORTC0(ADC0) */
    ADMUX =
        BVV(MUX0, 0) | BVV(MUX1, 0) | BVV(MUX2, 0) | BVV(MUX3, 0) /* Read ADC0 */
        | BVV(ADLAR, 1) /* Left justify */
        | BVV(REFS0, 1) | BVV(REFS1, 0); /* AVCC as reference voltage */
    ADCSRA =
        BVV(ADPS0, 1) | BVV(ADPS1, 1) | BVV(ADPS2, 1) /* clk/128 */
        | BVV(ADIE, 0) | BVV(ADIF, 0) /* No interrupt */
        | BVV(ADATE, 0) /* No auto-update */
        | BVV(ADSC, 0) /* Don't start conversion */
    
```

```

        | BVV(ADEN, 1); /* Enable */
    }

    static bool rain_is_raining(void)
    {
        return bit_is_clear(PIND, PIND7); /* OUT = 0 means rain */
    }

    static uint8_t rain_get_humidity(void)
    {
        ADCSRA |= _BV(ADSC); /* Start conversion */
        loop_until_bit_is_clear(ADCSRA, ADSC); /* Wait for end of conversion */
        return 255 - ADCH; /* lower means more humidity */
    }

```



La macro *BVV* définie en début de programme simplifie l'écriture bit à bit dans tout le registre avec une succession d'opérations logiques OR (`|`).

### III - Faire varier la luminosité d'une LED avec un signal PWM

Ensuite, on connecte une LED verte à la broche 6 (PD6/OC01) avec une résistance en série. Cette broche a été choisie car elle offre des fonctionnalités en relation avec le *Timer/Counter0* de l'ATmega, en particulier la possibilité de générer un signal modulé en largeur d'impulsion (ou PWM pour *Pulse Width Modulation*). En résumé, elle peut générer un signal rectangulaire sur la broche OC0A avec un rapport cyclique choisi, et plus le rapport cyclique se rapprochera des 100 % et plus la LED sera lumineuse. Le rapport cyclique est configuré grâce au registre 8 bits OCR0A (*Output Compare Register* du timer 0, unit A), entre 0 et 255 (rapport cyclique entre 0 % et 100 %).

Voici un bout de code qui contrôle la luminosité de la LED grâce à la PWM :



```

static void pwm_set_duty_cycle(uint8_t oc)
{
    OCR0A = oc;
}

static void pwm_init(uint8_t oc)
{
    DDRD |= _BV(DDD6); /* set pin 6 of PORTD for output*/
    TCCR0A =
        BVV(WGM00, 1) | BVV(WGM01, 1) /* Fast PWM update on OCRA */
        | BVV(COM0A1, 1) | BVV(COM0A0, 0) /* non-inverting OC0A */
        | BVV(COM0B1, 0) | BVV(COM0B0, 0); /* OC0B not connected */
    pwm_set_duty_cycle(oc);
    TCCR0B =
        BVV(CS00, 1) | BVV(CS01, 0) | BVV(CS02, 1) /* F_CPU/1024 */
        | BVV(WGM02, 0) /* Fast PWM update on OCRA */
        | BVV(FOC0A, 0) | BVV(FOC0B, 0); /* ignored */
}

```

### IV - Configuration de l'USART

Il existe un  **fichier d'entête** dans la chaîne de compilation *AVR libc* afin de configurer la vitesse de transmission en bauds de l' **USART**. On choisira la vitesse standard de 57 600 bauds, et bien que l'horloge de l'Arduino Uno ne tourne qu'à 16 MHz, il n'y a aucun problème à atteindre une telle fréquence. La configuration par défaut convient donc, il suffit d'activer la ligne TX de l'USART :

```

#define BAUD 57600
#include <util/setbaud.h>

static void usart_init(void)
{
    UBRR0H = UBRRH_VALUE;
}

```

```

    UBRRL0L = UBRRL_VALUE;
    #if USE_2X
        UCSR0A |= _BV(U2X0);
    #else
        UCSR0A &= ~_BV(U2X0);
    #endif
    UCSR0B = BVV(TXEN0, 1) | BVV(RXEN0, 0); /* Only TX */
}

static void usart_tx(char c) {
    while(!(UCSR0A & _BV(UDRE0)));
    UDR0 = c;
}

static void usart_puts(const char *s)
{
    while(*s != '\0')
    {
        usart_tx(*s++);
    }
}

```

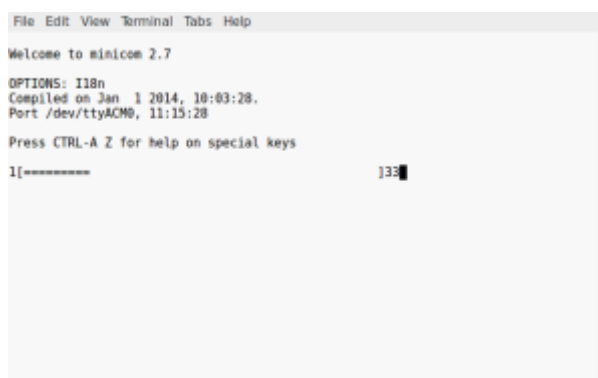
## V - Écriture du programme

Maintenant que l'on dispose de toutes les fonctions pour communiquer avec le matériel, il reste à les rassembler de façon cohérente.

On voudrait visualiser les données du capteur de pluie à la fois avec la LED et sur le port série.

Une augmentation de la luminosité de la LED lorsque l'humidité augmente peut être réalisée simplement en jouant avec le rapport cyclique de la PWM.

Par le port série, on propose de visualiser le niveau avec une « jauge » qui indique le niveau de pluie (0 : pas de pluie, 1 : pluie), une barre qui progresse avec le niveau d'humidité, et le niveau d'humidité donné par une valeur hexadécimale (plus facile à programmer). La chaîne de caractères contenant la jauge comporte un caractère de retour `\r` pour revenir en début de ligne, mais sans saut de ligne afin de permettre le rafraîchissement de celle-ci. Ci-dessous, une copie d'écran du résultat :



*Terminal minicom avec jauge d'humidité*

La fonction principale est une simple boucle qui lit les valeurs du capteur et en sortie, allume la LED et dessine la jauge via le port série avec une courte temporisation.

```

static void gauge(char *dst, uint8_t size, uint8_t val, uint8_t max_val)
{
    uint8_t i;
    uint8_t levels;
    uint8_t gauge_level;

    levels = size - 2 - 1; /* start/end markers and null char */

```

```

gauge_level = (val * (uint16_t)levels) / max_val;
*dst++ = '[';
for(i = 0; i < levels; i++)
{
    char c;

    c = (i < gauge_level)?'=':' ';
    *dst++ = c;
}
*dst++ = ']';
*dst = '\\0';
}

static void byte2hex(char *dst, uint8_t src)
{
    const char hexdigits[16] = "0123456789ABCDEF";

    *dst++ = hexdigits[(src >> 4) & 0xF];
    *dst++ = hexdigits[src & 0xF];
    *dst = '\\0';
}

static void update_gauge(bool state, uint8_t lvl)
{
    const uint8_t gauge_strlen = 50;
    char line[1+1+gauge_strlen+2+1];
    char *pline = &line[0];

    *pline++ = '\\r';
    *pline++ = state?'1':'0';
    gauge(pline, gauge_strlen+1, lvl, 255);
    pline += gauge_strlen;
    byte2hex(pline, lvl);
    pline += 2;
    *pline++ = '\\0';
    usart_puts(line);
}

int main (void)
{
    rain_init();
    usart_init();
    pwm_init(0);


    while (true)
    {
        bool raining;
        uint8_t humidity;

        raining = rain_is_raining();
        humidity = rain_get_humidity();

        pwm_set_duty_cycle(humidity);
        update_gauge(raining, humidity);
        _delay_ms(100);
    }
}

```

## VI - Génération, téléversement et exécution

Tout le code montré précédemment est rassemblé dans le fichier source *rain.c*, disponible aussi sur la plateforme  **GitHub**. Pour compiler et téléverser le programme, j'ai lancé les commandes suivantes depuis ma machine Debian :

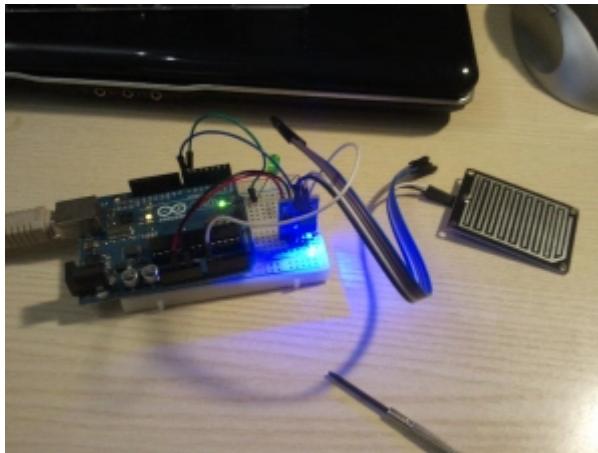
```

avr-gcc -g -Os -Wall -Wextra -DF_CPU=16000000UL -mmcu=atmega328p -c -o rain.o rain.c
avr-gcc -Wall -Wextra -mmcu=atmega328p rain.o -o rain
avr-objcopy -O ihex -R .eeprom rain rain.hex
avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:rain.hex

```



On peut tester le capteur en versant quelques gouttes d'eau sur sa partie sensible ou en plaçant ses doigts dessus.



Arduino Uno avec capteur de pluie

Afin d'établir la liaison série entre ma machine sous Debian et l'UART de l'ATmega, j'ai utilisé l'utilitaire *minicom* avec la commande suivante :

```
minicom -D /dev/ttyACM0 -b 57600
```

Soulignons que la liaison série est aussi utilisée pour le téléversement, et vous devez donc quitter *minicom* avant de téléverser un nouveau programme.

## VII - Conclusion

En jouant avec un capteur de pluie et ses sorties, je vous ai montré dans ce billet comment on pouvait utiliser certaines fonctionnalités de l'ATmega en langage C. La taille du programme fait moins de 600 octets, montrant ainsi l'avantage d'utiliser le langage C, en particulier lorsque la taille du programme et sa rapidité deviennent des obstacles.

## VIII - Notes de la Rédaction Developpez.com

Cet article est la traduction du billet écrit par 🇫🇷 **Francesco Balducci (alias Balau)**. Retrouvez **la version originale de cet article** ainsi que les autres chapitres consacrés à Arduino sur **son blog**.

Nous remercions les membres de la Rédaction de **Developpez.com** pour le travail de traduction et de relecture qu'ils ont effectué, en particulier : **f-leb, Delias, Vincent PETIT** et **Claude Leloup**.