ENDORMISSEMENT DE L'ARDUINO

Réalisé par Bruno PIQUEMAL

24 septembre 2020

Ce guide vous permettra d'enrichir vos connaissances sur les différentes façons d'économiser de l'énergie sur l'Arduino.

Introduction

L'Arduino est un microcontrôleur permettant de traiter des informations de son environnement afin d'effectuer des tâches précises. Cependant, il n'est pas toujours possible de laisser l'Arduino branché à une source de tension reliée au secteur. Dans des applications embarquées, il est nécessaire de réduire le temps d'éveil du Micro Controller Unit (MCU) pour augmenter la durée de vie de la batterie.

Nous verrons les six modes différents d'économie d'énergie que propose Arduino. Aussi, nous allons voir comment modifier les registres afin de consommer le moins possible.

Table des matières

⋆ ⋆ ⋆

A Présentation des différents modes d'endormissement

Sur les prochains exemples je vais parler du Sleep Mode Control Register (SMCR) qui est un registre clé en ce qui concerne l'économie d'énergie. C'est lui qui gère les différents états d'endormissement choisis.

9.11.1 SMCR – Sleep Mode Control Register

The sleep mode control register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
0x33 (0x53)	–	–	–	–	SM2	SM1	SM0	SE	SMCR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7..4 Res: Reserved Bits**

These bits are unused bits in the Atmel® ATmega328P, and will always read as zero.

FIGURE 1 – Tableau du Sleep Mode Control Register

Lorsque nous commandons au MCU de s'endormir en utilisant les fonctions prédéfinies par les librairies <avr/sleep.h> ou <avr/power.h>, elles modifient automatiquement le tableau ci-dessous. Cependant il est intéressant de comprendre ce que le MCU fait en arrière plan notamment concernant la manipulation des registres.

Table 9-2. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC noise reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	External standby(1)

Note: 1. Standby mode is only recommended for use with external crystals or resonators.

FIGURE 2 – Tableau du choix des modes d'endormissement

A.1 IDLE mode

(Consommation de l'ordre de 15 mA)

Quand les bits SM2-1-0 sont à 000, le Micro Controller Unit (MCU) se met en mode IDLE ce qui bloque le CPU mais laisse en marche le SPI, USART, le comparateur analogique, ADC, I2C, Timer/Compteurs, watchdog et le système d'interruption. Ce que fait ce mode c'est d'arrêter le clkCPU et le clkFLASH alors que les autres horloges fonctionnent encore. Aussi, il permet au MCU de se réveiller avec des interruptions externes et internes (timer overflow, USART). Si le réveil avec le comparateur analogique n'est pas utilisé, il peut être désactivé afin de réduire la consommation (voir le registre ACD et modifier l'ACSR).

Ce mode crée des coupures toutes les microsecondes ce qui permet d'économiser un peu d'énergie.

A.2 ADC mode

(Consommation de l'ordre de 6,5 mA)

Quand les bits SM2-1-0 sont à 001, le MCU se met en mode ADC réduction de bruit arrêtant le CPU mais permettant à l'ADC, les interruptions externes, l'I2C, le Timer/Compteur2 (en mode asynchrone) et au Watchdog de fonctionner. Ce que fait ce mode c'est d'arrêter le clkCPU, le clkI/O et le clkFLASH alors que les autres horloges marchent. Toutefois, il améliore la résolution des mesures en diminuant le bruit de l'ADC. Si ce dernier est activé, la conversion se fait de manière automatique dès que le MCU est plongé dans ce mode. Les seules choses qui peuvent réveiller l'Arduino sont : l'interruption suite à fin de conversion ADC, un redémarrage extérieur, le redémarrage ou l'interruption du watchdog, le redémarrage brown-out, la correspondance d'adresse via l'I2C, l'interruption avec le Timer/Compteur2, une interruption via le SPMM/EEPROM, un changement de niveau dans les pins INT0 ou INT1 ou tout autre pin.

Attention

Comme l'horloge du CPU est stoppée, la fonction millis() dérivera à chaque conversion de l'ADC. Voici des idées pour contrecarrer ceci :
`https://arduino.stackovernet.xyz/fr/q/9358`

A.3 Power Down mode

(Consommation de l'ordre de 0.36 mA)

Quand les bits SM2-1-0 sont à 010, le MCU se met en mode Power Down arrêtant l'oscillateur extérieur mais permettant aux interruptions externes, l'I2C et au Watchdog de marcher. Ce que fait ce mode c'est d'arrêter toutes les horloges ne laissant en fonctionnement que les modules asynchrones.

Les seules choses qui peuvent réveiller l'Arduino sont : un redémarrage extérieur, le redémarrage ou l'interruption du watchdog, le redémarrage brown-out, la correspondance d'adresse via l'I2C et un changement de niveau dans les pins INT0 ou INT1 ou une interruption dans les autres pins. Dans le cas d'une interruption par dépassement de niveau, cette dernière doit être suffisamment longue pour réveiller le MCU dans ce mode. Le réveil est un peu retardé afin de relancer les horloges internes et de les stabiliser (se référer à la datasheet pour les temps).

A.4 Power Save mode

(Consommation de l'ordre de 1.62 mA)

Quand les bits SM2-1-0 sont à 011, le MCU se met en mode Power Save qui est similaire au mode Power Down. La différence est que dans notre cas, il est possible d'activer le Timer/Compteur2 qui marchera pendant la phase d'endormissement. Ainsi, il peut se réveiller par dépassement d'un temps imposé ou par comparaison avec une valeur prédéfinie lorsque TIMSK2 est activé.

Si Timer/Compteur2 n'est pas utilisé, il est préférable d'utiliser le mode Power Down. Sachez que ce Timer peut être utilisé en mode synchrone et asynchrone (se référer à la datasheet).

A.5 Standby mode

(Consommation de l'ordre de 0.84 mA)

Quand les bits SM2-1-0 sont à 110 et qu'un oscillateur/résonateur est choisi, le MCU se met en mode Standby qui est similaire au mode Power Down. La différence est que dans notre cas, l'oscillateur/résonateur marchera pendant la phase d'endormissement. L'Arduino se réveille tous les 6 cycles de l'horloge.

A.6 Extended Standby mode

(Consommation de l'ordre de 1.62 mA)

Quand les bits SM2-1-0 sont à 111 et qu'un oscillateur/résonateur est choisi, le MCU se met en mode Extended Standby qui est similaire au mode Power Down. La différence est que dans notre cas, l'oscillateur/résonateur marchera pendant la phase d'endormissement mais aussi le Timer/Compteur2. L'Arduino se réveille tous les 6 cycles de l'horloge.

A.7 Tableau résumant les modes et leurs caractéristiques

Table 9-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes.

	Active Clock Domains					Oscillators		Wake-up Sources							
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O	Software BOD Disable
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X	
ADC noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X		
Power-down								X(3)	X				X		X
Power-save					X		X(2)	X(3)	X	X			X		X
Standby(1)						X		X(3)	X				X		X
Extended Standby					X(2)	X	X(2)	X(3)	X	X			X		X

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

FIGURE 3 – Caractéristiques des modes d'endormissement

B Modifications supplémentaires pour de la très basse consommation

Pour toute explication concernant le réveil suite à une interruption liée au module RTC, veuillez vous référer au tutoriel RTC DS3231.

B.1 Modification de l'état de pins

Démarrons cette section en constatant que le simple fait de mettre les pins en OUTPUT ou INPUT puis en HIGH ou LOW impacte la consommation du MCU.

Codage

```
#include <avr/sleep.h>

void setup ()
{
  for (byte i = 0; i <= A5; i++){
    pinMode (i, OUTPUT); // changed as per below
    digitalWrite (i, LOW); // ditto
    }
  // disable ADC
  ADCSRA = 0;
  set_sleep_mode (SLEEP_MODE_PWR_DOWN);
  noInterrupts ();      // timed sequence follows
  sleep_enable();
  // turn off brown-out enable in software
  MCUCR = bit (BODS) | bit (BODSE);
  MCUCR = bit (BODS);
  interrupts ();        // guarantees next instruction executed
  sleep_cpu ();         // sleep within 3 clock cycles of above
} // end of setup

void loop () { }
```

Avec le mode Power Down, les résultats sont les suivants :

- Pins en OUTPUT et LOW : 0.35 uA
- Pins en OUTPUT et HIGH : 1.86 uA
- Pins en INPUT et LOW : 0.35 uA
- Pins en INPUT et HIGH : 1.25 uA

B.2 Modification des registres

(Consommation de l'ordre de 0.12 mA)

Le code ci-dessous est là pour définir certains paramètres et présenter la structure utilisée pour modifier le temps du module RTC DS3231.

Codage

```
#include <Wire.h>
#include <ds3231.h>

#define wakePin 2  // when low, makes 328P wake up, must be an interrupt pin (2 or
    3 on ATMEGA328P)
#define ledPin 4  // output pin for the LED (to show it is awake)

// DS3231 alarm time
uint8_t wake_HOUR;
uint8_t wake_MINUTE;
uint8_t wake_SECOND;
#define BUFF_MAX 256

//  /* A struct is a structure of logical variables used as a complete unit
//  struct ts {
//      uint8_t sec;         /* seconds */
//      uint8_t min;         /* minutes */
//      uint8_t hour;        /* hours */
//      uint8_t mday;        /* day of the month */
//      uint8_t mon;         /* month */
//      int16_t year;        /* year */
//      uint8_t wday;        /* day of the week */
//      uint8_t yday;        /* day in the year */
//      uint8_t isdst;       /* daylight saving time */
//      uint8_t year_s;      /* year in short notation */
//  # ifdef CONFIG_UNIXTIME
//      uint32_t unixtime;   /* seconds since 01.01.1970 00:00:00 UTC*/
//  #endif
//  };
struct ts t;
```

Codage

```
void setup() {
  Serial.begin(9600);

//Save Power by writing all Digital IO LOW
//– note that pins just need to be tied one way or another, do not damage devices!

  for (int i = 0; i < 20; i++) {
    if(i != 2)//just because the button is hooked up to digital pin 2
    pinMode(i, OUTPUT);
    digitalWrite(i,LOW);
  }

  pinMode(wakePin, INPUT_PULLUP);

  // Flashing LED just to show the Controller is running
  digitalWrite(ledPin, LOW);
  pinMode(ledPin, OUTPUT);

  // Clear the current alarm (puts DS3231 INT high)
  Wire.begin();
  DS3231_init(DS3231_CONTROL_INTCN);
  DS3231_clear_a1f();

  Serial.println("Setup completed.");
}
```

Le code ci dessus permet de faire de l'économie d'énergie en mettant les pins à l'état OUTPUT et LOW.
C'est une manière simple d'éviter de perdre quelques microampères inutilement.

Codage

```
1  // The loop blinks an LED when not in sleep mode
2  void loop() {
3    char buff[BUFF_MAX]; //to print the time
4
5    // Just blink LED ONCE to show we're running
6    digitalWrite(ledPin, HIGH);
7    delay(500);
8    digitalWrite(ledPin, LOW);
9
10   // Is the "go to sleep" pin now LOW?
11   // if (digitalRead(sleepPin) == LOW)
12   //{
13     // Set the DS3231 alarm to wake up in X seconds
14     setNextAlarm();
15
16     //Disable ADC - don't forget to flip back
17     // after waking up if using ADC in your application ADCSRA |= (1 << 7);
18     ADCSRA &= ~(1 << 7);
19
20     //ENABLE SLEEP - this enables the sleep mode
21       SMCR |= (1 << 2); //power down mode
22       SMCR |= 1;//enable sleep
23
24     attachInterrupt(digitalPinToInterrupt(wakePin), sleepISR,
          FALLING);
25
26     // Send a message just to show we are about to sleep
27     Serial.println("Good night!");
28     // display current time
29     snprintf(buff, BUFF_MAX, "%d.%02d.%02d %02d:%02d:%02d\n", t.
          year,
30     t.mon, t.mday, t.hour, t.min, t.sec);
31     Serial.print(buff);
32     Serial.flush();
```

En ce qui concerne la première partie de la fonction loop() présentée ci-dessus, on voit que le programme va allumer puis étendre une LED, dans un premier temps. Ensuite, on met en place une alarme de dix secondes.
La première manipulation du registre se fait en déconnectant le convertisseur analogique-numérique puisqu'il est gourmand en énergie.
On prend le mode Power Down qui est celui qui permet de faire les meilleures économies d'énergies au nivea du MCU puis on prévient le microcontrôleur qu'on est prêt à nous endormir.
Il est nécessaire de rajouter une interruption afin que l'Arduino se réveille.

Codage

```
    //BOD DISABLE - this must be called right before the __asm__ sleep instruction

     MCUCR |= (3 << 5); //set both BODS and BODSE at the same ti timeme
     MCUCR = (MCUCR & ~(1 << 5)) | (1 << 6); //then set the BODS bit and
         clear the BODSE bit at the same

     // And enter sleep mode as set above
      __asm__ __volatile__("sleep");//in line assembler to go to sleep

    // ------------------------------------
    // uController is now asleep until woken up by an interrupt
    // ------------------------------------

    // Wakes up at this point when wakePin is brought LOW - interrupt routine is
        run first
    Serial.println("I'm awake!");

    ADCSRA &= ~(0 << 7);

    // Clear existing alarm so int pin goes high again
    DS3231_clear_a1f();
}

// When wakePin is brought LOW this interrupt is triggered FIRST (even in
    PWR_DOWN sleep)
void sleepISR() {

  // Detach the interrupt that brought us out of sleep
  detachInterrupt(digitalPinToInterrupt(wakePin));

  // Now we continue running the main Loop() just after we went to sleep
}
```

Pour cette deuxième partie de la fonction loop(), on désactive le Brown-Out Detector (BOD) qui se charge de superviser la source de tension (inutile en sleep mode). Puis on demande au MCU de s'endormir et dix secondes plus tard, il se réveille. Le BOD s'active automatiquement donc il n'y a pas besoin de le manipuler. Cependant le convertisseur analogique-numérique nécessite que la modification soit faite à la main.

Puis nous remettons le pin INT0 en mode HIGH en attendant le prochain cycle.

Codage

```
1  // Set the next alarm
2  void setNextAlarm(void)
3      {
4    // flags define what calendar component to be checked against the current time in
         order
5    // to trigger the alarm - see datasheet
6    // A1M1 (seconds) (0 to enable, 1 to disable)
7    // A1M2 (minutes) (0 to enable, 1 to disable)
8    // A1M3 (hour)    (0 to enable, 1 to disable)
9    // A1M4 (day)     (0 to enable, 1 to disable)
10   // DY/DT          (dayofweek == 1/dayofmonth == 0)
11   uint8_t flags[5] = { 0, 0, 0, 1, 1 };
12
13   // get current time so we can calc the next alarm
14   DS3231_get(&t);
15
16   wake_SECOND = t.sec;
17   wake_MINUTE = t.min;
18   wake_HOUR = t.hour;
19
20   // Add a some seconds to current time. If overflow increment minutes etc.
21   wake_SECOND = wake_SECOND + 10;
22   if (wake_SECOND > 59)
23   {
24     wake_MINUTE++;
25     wake_SECOND = wake_SECOND - 60;
26
27     if (wake_MINUTE > 59)
28     {
29       wake_HOUR++;
30       wake_MINUTE -= 60;
31     }
32   }
33
34   // Set the alarm time (but not yet activated)
35   DS3231_set_a1(wake_SECOND, wake_MINUTE, wake_HOUR, 0, flags);
36
37   // Turn the alarm on
38   DS3231_set_creg(DS3231_CONTROL_INTCN | DS3231_CONTROL_A1IE);
39 }
```

Le code ci-dessus nous présente la fonction qui permet de mettre en place l'alarme qui réveillera l'Arduino.

Quelques exemples d'utilisation

Capteurs IoT, modules placés dans des endroits difficiles d'accès, robots embarqués...

C Pour aller plus loin

D'autres méthodes existent pour économiser de l'énergie comme par exemple la réduction de la fréquence de fonctionnement de l'Arduino et aussi la diminution de la tension de sa source d'alimentation. Toutefois, ces mesures doivent être réfléchies car des endommagements ou dysfonctionnements peuvent être causés.

D Bibliographie

http://www.gammon.com.au/power

https://arduino.stackovernet.xyz/fr/q/9358

https://www.youtube.com/watch?v=urLSDi7SD8M

https://www.youtube.com/watch?v=-dW4XsBo3Mk