

Mode veille et gestion d'énergie

É. Carry, J.-M Friedt

26 février 2021

Questions sur ce TP :

1. Quelle est la consommation d'une LED?
2. Quel est le courant de fuite traversant une résistance de tirage R connectée à V_{cc} lorsque l'interrupteur relié à la masse qui commande un GPIO est fermé? Application numérique pour $V_{cc} = 5\text{ V}$ et $R = 5600\ \Omega$. Comment réduire ce courant de fuite?
3. Quelle est la relation entre la fréquence de cadencement d'un périphérique et sa consommation?
4. Un circuit consommant 4 mA est alimenté par une batterie de capacité 2000 mA.h. Quelle sera la durée de fonctionnement de ce circuit sur cette batterie?
5. Compte tenue de la réponse précédente, quel est le bénéfice de mettre le circuit en veille profonde (consommation 0,1 mA) entre deux activités (consommation 4 mA) avec un rapport cyclique de 80%? Quel est l'autonomie résultant de ce mode de fonctionnement? (on pourra raisonner en terme de courant moyen consommé)

L'objectif de ce TP est d'illustrer diverses stratégies de gestion d'énergie lors de l'utilisation d'un microcontrôleur dans une application alimentée par batterie ou pile. Il arrive souvent que les périphériques du microcontrôleur ou son unité de calcul (ALU) ne soient sollicités que par intermittence et que la majorité du temps se passe à attendre un événement (date d'acquisition de donnée, événement ponctuel à observer).

La consommation électrique d'un système numérique synchrone est avant tout déterminé par les transitions d'état de ses portes logiques. En effet, chaque porte logique se comporte comme un condensateur C , et une transition d'état se traduit par l'évacuation des charges accumulées pour passer d'un potentiel de la tension d'alimentation V_{cc} à 0 V. Dans ces conditions, l'énergie associée à chaque transition [1] est $E = \frac{1}{2} \cdot C \cdot V_{cc}^2$ – cette valeur est de l'ordre de 1 pJ pour les architectures actuelles. Pour une fréquence de cadencement des traitements numériques de f , la puissance consommée est donc $P = \frac{1}{2} \cdot C \cdot V_{cc}^2 \cdot f$ W, et cette valeur est multipliée par le nombre de portes logiques. Évidemment, plus le microprocesseur contient de portes logiques et plus il consomme, mais pour une architecture donnée de microcontrôleur la réduction de puissance passe par la réduction de f , voir par l'arrêt total de l'horloge cadencant divers périphériques. En effet, nous constatons que la consommation électrique d'un circuit numérique synchrone est proportionnelle à sa fréquence de cadencement.

Lors de l'alimentation par une pile ou batterie de capacité B exprimée en A.h, l'autonomie du circuit est de l'ordre de B/I avec I le courant moyen consommé par le circuit numérique. Plus la valeur moyenne de I est réduite, plus l'autonomie est allongée : maximiser la durée du mode veille est une garantie d'autonomie importante. Cependant, on prendra soin de considérer les fuites de courant en dehors du microcontrôleur lui-même : une LED (Diode ÉlectroLuminescente) présente une chute de tension typiquement de l'ordre de 1,6 V, et sa consommation sera donc $(V - 1,6)/R$ avec R la résistance de limitation de courant lorsque la LED est polarisée par une tension V . En pratique, nous choisirons souvent $R \approx 400\ \Omega$ afin de limiter le courant à une dizaine de milliampères tel que nécessaire pour illuminer une LED.

En mode veille profonde (SLEEP_MODE_PWR_DOWN), les interruptions logicielles ne permettent pas le réveil : seules les interruptions sur broches ou chien de garde permettent de réactiver le microcontrôleur (voir p.45 de la datasheet [2] pour la liste des sources de réveil selon les modes). En effet, tous les périphériques du microcontrôleur ont été arrêtés : le réveil doit nécessairement se faire par un stimulus externe. C'est la solution la plus économe en énergie mais la plus contraignante en utilisation.

Nous allons donc aborder deux approches : le mode de veille profonde qui sera réveillé par le transfert d'un caractère sur le port RS232 – se traduisant par le déclenchement d'une interruption matérielle sur broche – et le mode de veille légère dans lequel un *timer* réveille périodiquement le cœur de calcul, avec par conséquent une économie énergétique moindre. Dans tous ces exemples, un ampèremètre mesure un courant sur le connecteur VR2¹ et les mesures se feront soit en alimentant le microcontrôleur sous 5 V, soit sous 3,3 V.

1. https://www.olimex.com/Products/Duino/AVR/OLIMEXINO-32U4/resources/OLIMEXINO-32U4_rev_A3.pdf

1 Réveil par communication en mode SLEEP_MODE_PWR_DOWN

Le programme ci-dessous se réveille chaque fois qu'un caractère est émis sur le port de communication asynchrone, et renvoie un message avant de rendormir le microcontrôleur.

Listing 1 – Passage en mode veille

```
1 #include "libttycom.h"
2 #include <avr/io.h> // voir /usr/lib/avr/include/avr/iom32u4.h
3 #include <avr/interrupt.h>
4 #include <avr/sleep.h>
5 #define F_CPU 16000000UL
6 #include <util/delay.h> // _delay_ms
7
8 #define USART_BAUDRATE 9600
9 volatile char flag=0;
10
11 ISR(INT2_vect) {flag=1;}
12
13 void uart_transmit( unsigned char data )
14 {while (!(UCSR1A&(1<<UDRE1))) ;
15  UDR1 = data;
16 }
17
18 void uart_puts(char *s)
19 {int k=0;
20  while (s[k]!=0) {uart_transmit(s[k]);k++;}
21 }
22
23 int main(void){
24  unsigned short baud;
25  char s[8]="hello\r\n\0";
26  init_olimaxIno();
27  DDRB |=1<<PORTB5; // LEDs
28  DDRE |=1<<PORTE6;
29  PORTB |= 1<<PORTB5;
30  PORTE &= ~1<<PORTE6;
31
32  UCSR1A = 0; // importantly U2X1 = 0
33  UCSR1B = (1 << RXEN1) | (1 << TXEN1); // enable receiver and transmitter
34  UCSR1C = _BV(UCSZ11) | _BV(UCSZ10); // no parity, 8 data bits, 1 stop bit
35  UCSR1D = 0; // no cts, no rts
36  baud = ((( F_CPU / ( USART_BAUDRATE * 16UL)) - 1));
37  UBR1H = (unsigned char)(baud>>8);
38  UBR1L = (unsigned char)baud;
39
40  EIMSK = 1<<INT2; //enable int2
41  sei();
42
43  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
44  sleep_enable();
45  while (1){
46    uart_puts(s);
47    _delay_ms(100);
48    if (flag!=0) {flag=0;PORTB^=1<<PORTB5;} // pourquoi ca ne fait rien ?
49    PORTE^=1<<PORTE6;
50    sleep_mode();
51    // sleep_disable(); // premiere chose a faire quand on se reveille -- include dans _mode
52  }
53  return 0;
54 }
```

L'interruption UART ne permet pas de sortir du mode veille : nous déclenchons ici le réveil sur l'interruption GPIO INT2 qui s'avère se déclencher sur la même broche que celle en charge de la réception de données sur l'UART. On notera que le brochage du port UART sur le connecteur UEXT a été fourni dans l'énoncé du second TP disponible à http://jmfriedt.free.fr/TP_Atmega32U4_interrupt.pdf.

Retirer la commande `sleep_mode` de la boucle infinie et observer le comportement ainsi que la consommation du circuit.

Consulter `/usr/lib/avr/include/avr/sleep.h` et en particulier la fonction suivante :

```
#define sleep_cpu() \
do { \
    __asm__ __volatile__ ( "sleep" "\n\t" :: ); \
} while(0)

#define sleep_mode() \
do { \
    sleep_enable(); \
    sleep_cpu(); \
    sleep_disable(); \
} while (0)
```

Ce mode de fonctionnement est efficace lors de l'interaction d'un système embarqué avec un utilisateur : la consommation électrique n'augmente que lorsque l'utilisateur sollicite le dispositif. Un cas classique de système embarqué consiste à observer périodiquement un phénomène (*data logger*) : dans ce cas, un mécanisme de réveil périodique est plus approprié. Cette approche est présentée ci-dessous.

2 Réveil par *timer 1*

Si le réveil doit se faire par une interruption logicielle, le mode de veille profonde vu ci-dessus ne peut être utilisé car les périphériques y sont désactivés. Un mode de veille intermédiaire est le mode *idle* dans lequel seuls quelques fonctions du processeur sont désactivées.

Listing 2 – Passage en mode veille

```
1 #include <avr/io.h> //E/S ex PORTB
2 #include "libttypcom.h"
3 #define F_CPU 16000000UL
4 #include <avr/interrupt.h>
5 #include <avr/sleep.h>
6 #include <util/delay.h> // _delay_ms
7 #include "USBAPI.h" // USB_writestr()
8
9 volatile int f_timer=0;
10
11 ISR(TIMER1_OVF_vect) {if(f_timer == 0) f_timer = 1;}
12
13 void enterSleep(void)
14 {
15     set_sleep_mode(SLEEP_MODE_IDLE);
16     sleep_enable();
17     sleep_mode(); // enter sleep mode ... good night
18     sleep_disable(); // wake up from interrupt
19 }
20
21 void setup()
22 {
23     DDRB |=1<<PORTB5; // LED
24     PORTB |= 1<<PORTB5;
25
26     TCCR1A = 0x00; // normal timer (overflow)
27     TCNT1=0x0000; // clear timer counter
28     TCCR1B = 0x05; // prescaler: 1 s
29     TIMSK1=0x01; // timer overflow interrupt
30 }
31
32 int main()
33 {init_olimaxIno();
34     setup();
```

```

35 sei();
36
37 while (1)
38 {
39
40   if(f_timer==1)
41   {f_timer = 0;
42     PORTB ^= 1<<PORTB5; // toggle LED
43     enterSleep();      // 34 mA si inactif, 26 mA si actif
44   }
45 }
46 }

```

Tel que proposé ici, le *timer* induit un changement trop rapide d'état de la LED pour qu'un ampèremètre puisse effectuer une mesure précise. **Modifier le programme afin de réduire la fréquence de commutation de la LED à moins de 1 Hz – une période de 4 s est accessible par une modification appropriée des registres de configuration du *timer*.**

Finalement, il est possible de combiner les diverses méthodes de génération d'interruptions vues tout au long de ces TPs. Le microcontrôleur peut être sorti de son mode veille soit par le Timer 1 en mode Output Compare (donc avec une meilleure résolution que par Overflow), soit par INTO. Les deux cas sont distingués par la définition de drapeaux différents qui sont testés indépendamment dans la fonction `main()` et déclencher l'affichage de messages différents sur le port USB.

Proposer un programme dans lequel la boucle principale place le microcontrôleur en veille, et affiche deux messages distincts sur port UART selon que le réveil se fasse sur un événement *timer*, soit sur la liaison par un fil de INTO à la masse. Mesurer la consommation électrique lors de l'exécution de ce programme.

Consommations électriques mesurées

À titre de comparaison, les mesures de courant dans les divers modes de fonctionnement de la carte Atmega32U4 de Olimex est de l'ordre de

Condition de mesure	$V_{cc} = 5\text{ V}$	$V_{cc} = 3,3\text{ V}$
En marche	32 mA	15 mA
Idle	24-27 mA	11 mA
Power down	5 mA	2 mA

Chaque LED consomme entre 1 et 2 mA.

Références

- [1] M. Rafiquzzaman, *Microprocessors and microcomputer-based system design, 2nd Ed.*, CRC Press (1995)
- [2] <http://www.atmel.com/images/doc7766.pdf>