Faible consommation

Pierre-Yves Rochat

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

La consommation des microcontrôleurs

Les microcontrôleurs ont en général une consommation comprise entre moins d'un mA et quelques dizaines de mA en fonctionnement normal. On se trouve donc souvent devant un montage où une simple LED consomme beaucoup plus que le microcontrôleur !

Bien qu'elle semble faible, cette consommation est souvent incompatible avec l'usage de piles. Par exemple, une petite pile ronde au Lithium CR1620 (16 mm de diamètre, 2.0 mm d'épaisseur) peut avoir une capacité d'environ 100mAh. Il n'y a de quoi alimenter un microcontrôleur que durant quelques heures, ou quelques jours dans le meilleur des cas.

Comment résoudre ce problème ? Regardons d'abord les facteurs qui influencent la consommation : la tension d'alimentation et la fréquence du processeur. Voilà les courbes données pour le MSP430G2231 :

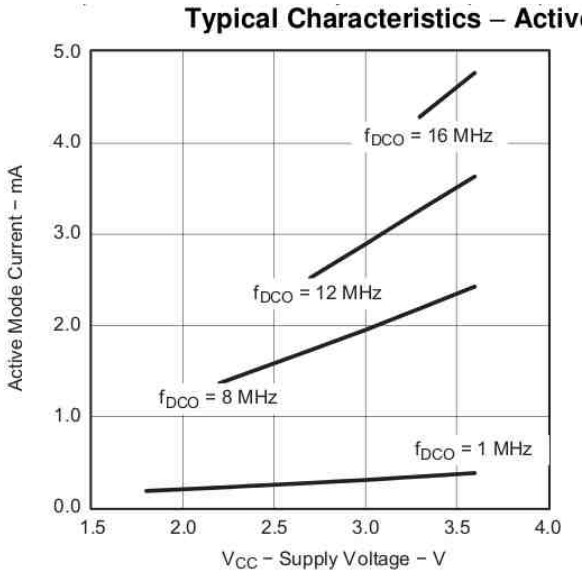

Figure 2. Active Mode Current vs V_{CC}, T_A = 25°C

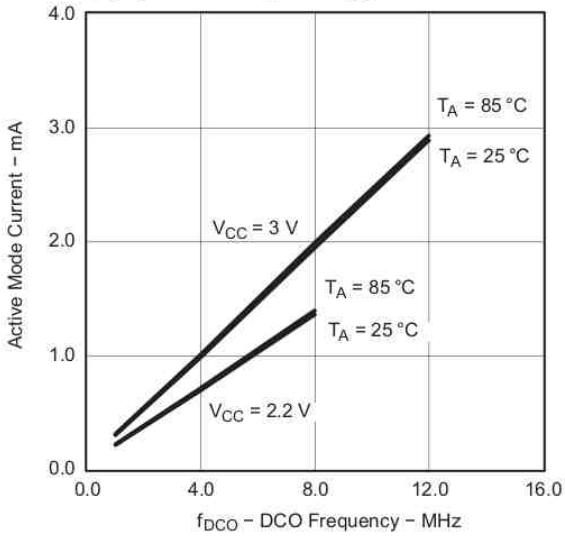

Figure 3. Active Mode Current vs DCO Frequency

Dépendance de la tension

La tension d'alimentation joue un rôle dans la consommation. C'est pourquoi les fabricants donnent souvent la consommation typique à une tension basse. On voit sur la figure ci-dessus que la consommation va presque doubler en passant de 2.2 V à 3.5 V.

Mais dans le même temps, la puissance (P = U * I) augmente de manière encore plus significative : elle va plus que tripler !

Dépendance de la fréquence

Mais la consommation dépend encore davantage de la fréquence de l'horloge du processeur : pour une même tension de 3 V, on observe sur la courbe ci-dessus une consommation presque dix fois plus élevée à 12 MHz qu'à 1 MHz.

C'est la raison pour laquelle les fabricants offrent la possibilité de changer la fréquence du processeur sur presque tous les microcontrôleurs. Mais les solutions proposées diffèrent beaucoup d'une famille de microcontrôleur à l'autre.

Les AVR offrent une ajustage de la fréquence par une division de l'horloge par 8, commandé par un bit de configuration en mémoire EEPROM protégée (programmé

généralement en même temps que le programme).

Les MSP430 proposent un ajustage de la fréquence par écriture dans des registres. Il est donc possible de changer à tout instant la vitesse du processeur, dans des facteurs importants (par exemple de 128 kHz à 16 MHz).

Les processeurs ARM offerts par différents fabricants ont généralement une gestion d'horloge très complexe. La documentation du STM32 contient 37 pages sur ce sujet ! Il devient très tentant d'utiliser des librairies fournies par les fabricants pour initialiser les horloges des différente parties du microcontrôleur. Notons toutefois que ces librairies sont souvent loin d'être parfaites...

Mode Sleep

Pour limiter au maximum la consommation, les fabricants proposent de passer à certain moments à une fréquence nulle, c'est-à-dire arrêter le processeur du microcontrôleur, vu qu'il est la cause de la plus grande partie de la consommation. A une fréquence nulle, la consommation va alors se limiter au courant de fuite des transistors. On tombe à des valeurs souvent inférieures à 10 µA, voire 1 µA. C'est parfait pour économiser les piles !

Mais qui va alors remettre en marche le processeur quand ce sera nécessaire ? Ce sont les interruptions !

Avant d'endormir le processeur, on va donc initialiser des interruptions. Lorsqu'elles proviennent d'une patte externe (Interrupt Pins ou Pin Change Interrupt), c'est un événement externe qui réveille le processeur. Mais il est aussi possible d'initialiser l'interruption d'un Timer. Il s'agit alors d'un sommeil de durée limitée.

De nombreux modes de sommeil

Les microcontrôleurs modernes ont en fait plusieurs modes de sommeil différents. Chacun a sa particularité. Ils laissent actifs différentes parties du microcontrôleur pour répondre à des besoins différents, avec chaque fois des consommations différentes.

A titre d'exemple, les ATmega168/328 en ont six, appelés Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby.

Sommeil pour les ADC

Endormir le processeur a aussi un autre intérêt très spécifique avec les Convertisseurs Analogiques Numériques (ADC en anglais), qui sont sensibles aux parasites. On choisit souvent d'endormir le processeur durant la conversion, pour éviter les parasites générés par le processeur. Dans ce cas, ce sera l'interruption de fin de conversion qui va réveiller le processeur.

Voici les instructions correspondantes pour un MSP430 :

```
int main() {
   WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog Timer
   // ADC 10bits ON, mode sample and hold et interrupt enable  :
   ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE;
   ADC10CTL1 = INCH_1; // Choix de l'entrée A1
   ADC10AE0 |= (1<<1); // choix de l'option pour PA.1 : ADC

   while (1) {  {
      ADC10CTL0 |= ENC + ADC10SC; // Début de la conversion
      // Arrêt du processeur (mode LPM0) :
```

```
        __bis_SR_register (CPUOFF + GIE); // et interrupt enable
        // ... suite du programme, le résultat est dans ADC10MEM
    }
}
```

Il faut s'accrocher pour comprendre chaque ligne... ou faire confiance au copier-coller à partir des exemples du fabricant !

Notez que cette manière de lire une valeur analogique est bloquante, au sens le plus fort du terme : le processeur est arrêté.

Exemple du Dé électronique

Pour le dé électronique que je fais monter à des enfants, j'ai choisi de ne pas mettre d'interrupteur pour couper le courant. On oublie si facilement d'éteindre ce genre de gadgets que les piles sont toujours plates ! Le montage doit donc s'endormir après un certain temps (souvent 20 à 30 secondes). Une pression sur le bouton-poussoir génère une interruption qui réveille le processeur.

Voici le programme correspondant pour unATtiny13 :

```
void Sommeil() { // Mise en veille du processeur
     PortLeds = Zero; // Extinction de toutes les LEDs.
    // Configuration de l'interruption de réveil :
     PCMSK |= (1<<BitPoussoir); // Pin Change sur PB3.
     GIMSK |= (1<<PCIE); // Pin Change Interupt Enable
     sei(); // Activation générale des interruptions
     // Choix et activation du mode veille :
     set_sleep_mode(SLEEP_MODE_PWR_DOWN);
     sleep_enable();
     sleep_cpu(); // Passage en mode veille.
}
```

Vérifier la consommation

Il est facile de mesurer la consommation d'un microcontrôleur lorsqu'il travaille. La mesure de courant doit se faire en série sur l'alimentation. Une solution simple est d'insérer une résistance en série et de mesurer la tension à ses bornes. La loi d'Ohm donne alors le courant.

Mais quand le processeur est arrêté, comment procéder pour contrôler que la consommation est bien aussi faible que le fabricant l'annonce ? Il n'est pas facile de mesurer des courant de l'ordre du µA !

Voici une solution astucieuse :

- Placer un condensateur en parallèle avec l'alimentation. 100 µF convient bien.
- Laisser le montage passer en *Sleep*.
- Enlever la pile.
- Mesurer après un certain temps la tension au borne du condensateur.

Laisser le multi-mètre pour observer en continu la décroissance de la tension fausserait sérieusement la mesure !

Sur un Dé électronique (avec un AVR ATtiny13 et avec une pile de CR1620), la tension mesurée était d'environ 2 V après une minute. La mesure sur un MMSP430G2231 a donné des résultats similaires.

La charge électrique consommée en une minute a donc été de :

$Q = C * \delta U$

La décroissance est en fait une exponentielle décroissante.

Mais nous pouvons approximer : $Q = 100\ \mu F * 1\ V$

Or l'autre formule pour la charge dit que : $Q = I * t$

$I = Q / t = 100\ \mu F * 1\ V / 60\ s = 1.6\ \mu A$. L'ordre de grandeur semble correct !

PYR, version 2013/10/12