Déboguer la consommation dans les applications embarquées

par
Tiago Monte, Ingénieur Application Energy Micro

Le nombre d'applications embarquées où l'économie d'énergie et l'efficacité énergétique arrivent en tête des priorités des développeurs ne cesse d'augmenter. Les régulations gouvernementales (dans le cas des applications de comptage d'énergie par exemple), la nécessité d'allonger l'autonomie des batteries, ou le simple besoin de réduire les factures d'électricité, constituent autant d'explications à ce phénomène.

Signe symptomatique de cette tendance, le marché a vu se multiplier les microcontrôleurs (MCU) à « ultrabasse consommation ». Malheureusement, jusqu'à une époque très récente, ces circuits n'étaient pas commercialisés avec des outils permettant aux développeurs d'ajuster avec précision la consommation de leurs équipements. Car le fait de disposer d'un MCU éco-énergétique n'est pas, à lui seul, synonyme d'une diminution de la consommation à l'échelle du système. L'astuce consiste plutôt à optimiser le logiciel embarqué, non seulement en termes de fonctionnalités, mais également dans un souci d'efficacité énergétique. La surveillance étroite des composants environnant le MCU, tout comme l'optimisation globale du logiciel embarqué et de l'usage des périphériques, sont des facteurs essentiels à une réduction générale de la consommation. Le logiciel embarqué n'est généralement pas perçu comme une source de consommation et, pourtant, chaque cycle d'horloge consomme de l'énergie. Il faut donc prendre en compte cet aspect si l'on souhaite minimiser la consommation globale du système.

Le développement de systèmes embarqués éco-énergétiques peut être scindé en trois étapes: le débogage du matériel, le débogage du logiciel et le débogage de la consommation.

Le défi de l'éco-conception logicielle

Pour les développeurs d'applications MCU, les principaux enjeux consistent typiquement à réduire l'usage de la mémoire et à concevoir le code le plus compact qui soit. Pour satisfaire les contraintes éco-énergétiques, l'objectif recherché est donc de confiner le système en mode veille le plus longtemps possible. Ce scénario est classique mais c'est loin d'être le seul moyen d'économiser l'énergie. Les MCU éco-énergétiques disposent souvent d'autres caractéristiques qui permettent de réduire encore leur consommation. Au-delà des différents modes veille proposés par ces MCU, c'est dans l'utilisation efficace de ces caractéristiques que réside le vrai secret en matière d'économie d'énergie.

Au fur et à mesure de l'état d'avancement du processus de développement, la taille du code tend inexorablement à augmenter. Optimiser le logiciel pour le rendre éco-efficace devient alors une tâche de plus en plus complexe. Il s'avère notamment de plus en plus difficile d'identifier les erreurs telles que l'utilisation impropre de périphériques ou les cycles d'attente qui pourraient être remplacés par des routines de service d'interruptions (ISR). Or, si ces « bogues de consommation » ne sont pas repérés et corrigés pendant la phase de

développement, il est pratiquement impossible de les détecter durant les tests de fatigue (burn-out) ou les tests opérationnels.

Le meilleur moyen de surveiller le niveau de consommation d'un système consiste à échantillonner le courant électrique durant un certain laps de temps, puis à moyenner et extrapoler les valeurs résultantes sur des durées plus longues. Un oscilloscope ou un multimètre suffit pour effectuer ce type de mesure, mais les résultats obtenus ne peuvent être corrélés aux fonctionnalités logicielles. D'un autre côté, il est possible d'utiliser un analyseur logique pour conserver la trace des activités logicielles, mais il n'existe aucun moyen de les associer aux variations de la consommation énergétique. Si l'on désire simplement estimer l'autonomie de la batterie, les résultats obtenus par extrapolation ne devraient pas sensiblement différer de ceux obtenus lors d'un scénario réel d'utilisation. Lorsqu'il s'agit d'optimiser le code pour améliorer l'efficacité énergétique, cette approche conventionnelle s'avère toutefois peu utile.

Une approche plus productive

Les problèmes liés au débogage de la consommation peuvent aujourd'hui être résolus grâce à des outils matériels et logiciels qui s'avèrent capables d'afficher sous forme graphique les données de consommation en temps réel, et de corréler à tout moment le courant électrique consommé avec le code en cours d'exécution. Avec à la clé une réduction considérable de la durée et des efforts de développement.

Dans la pratique, le positionnement d'un capteur de courant sur le rail d'alimentation pour mesurer l'intensité électrique permet de surveiller la consommation. A chaque top d'horloge, le courant est échantillonné, puis la valeur échantillonnée subit une conversion analogique/numérique et la donnée résultante est transmise via un port USB avec, éventuellement, la valeur de tension et l'information temporelle. Pour la majorité des applications éco-énergétiques, la capacité à mesurer avec précision des valeurs comprises grosso modo entre 100nA et 50mA est en général suffisante. Il est d'ailleurs possible de mesurer simultanément le courant électrique consommé par le MCU et par les autres composants du système, dès lors qu'ils sont alimentés au travers du rail d'alimentation placé sous surveillance. Avec cette approche, le débogage de la consommation s'apparente à un processus de développement cyclique où l'association d'un MCU éco-énergétique et d'outils matériels et logiciels garantit une réduction maximale de la consommation de l'application.

Un exemple d'application où la consommation est réduite d'un facteur 1000

Tous les kits Energy Micro associés au MCU éco-énergétique EFM32 Gecko (kits de démarrage Starter Kits STK et kits de développement Development Kits DVK) intègrent un système de gestion évoluée de la consommation AEM (Advanced Energy Monitoring). Les informations de consommation peuvent être visualisées en temps réel de deux manières différentes : soit directement sur l'écran LCD embarqué (si le DVK est utilisé), soit sur l'écran d'un PC par le biais de l'analyseur energyAware Profiler d'Energy Micro (après connexion de l'un des kits à un port USB de l'ordinateur).

Une vue classique des données affichées par l'analyseur energyAware Profiler est présentée sur la Figure 1.

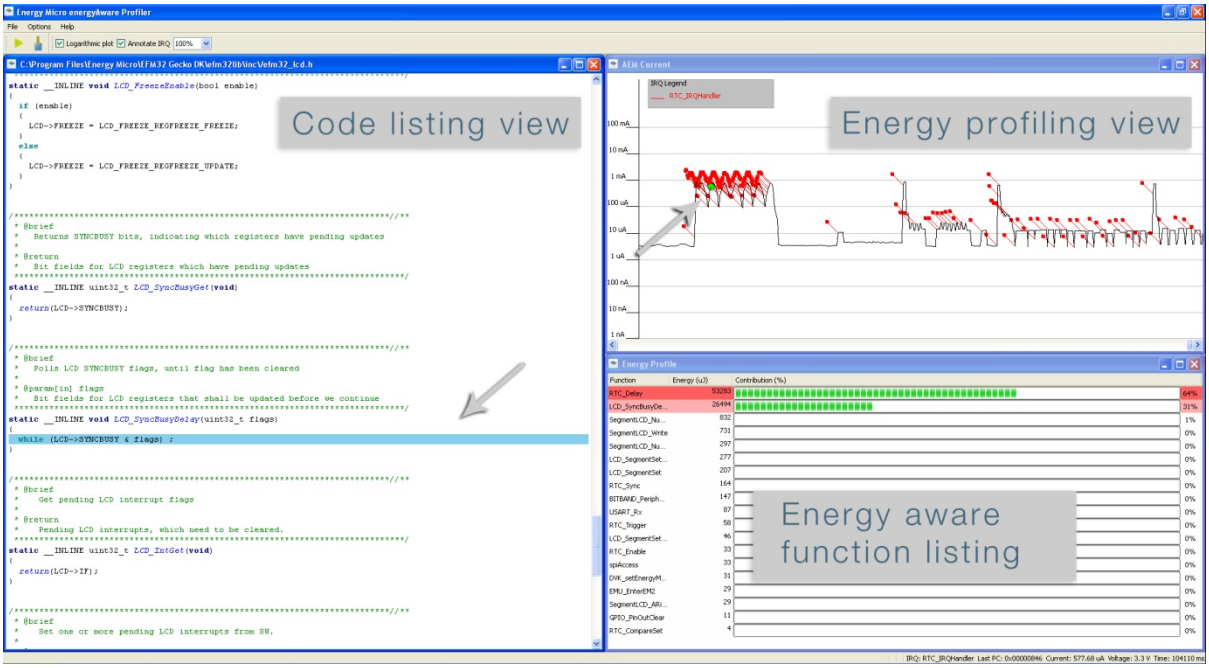


Figure 1: Le débogage de la consommation est simplifié car les conséquences de toute modification sur le code sont surveillées en temps réel

Dans la pratique, l'analyseur logiciel energyAware Profiler reçoit au travers du port USB les données diffusées par le système AEM des kits de développement. Les informations sont affichées sur l'écran sur une courbe graphique avec le temps en abscisse et le courant consommé en ordonnée.

L'exemple suivant montre comment l'analyse de la consommation, associée aux caractéristiques du microcontrôleur EFM32, peut être mise à profit pour réduire la consommation d'énergie. Dans le cas présent, seul le module LEUART est mis en œuvre. Il gère la communication UART à une vitesse maximale de 9600 bauds tout en affichant une consommation réduite au minimum.

Pour récupérer les informations dans la mémoire-tampon de réception (buffer RX), l'usage veut que l'on sonde le buffer jusqu'à obtention de données valides et enclenchement du processus de lecture. Mais cette opération exige que le processeur reste en mode actif, d'où une consommation relativement élevée.

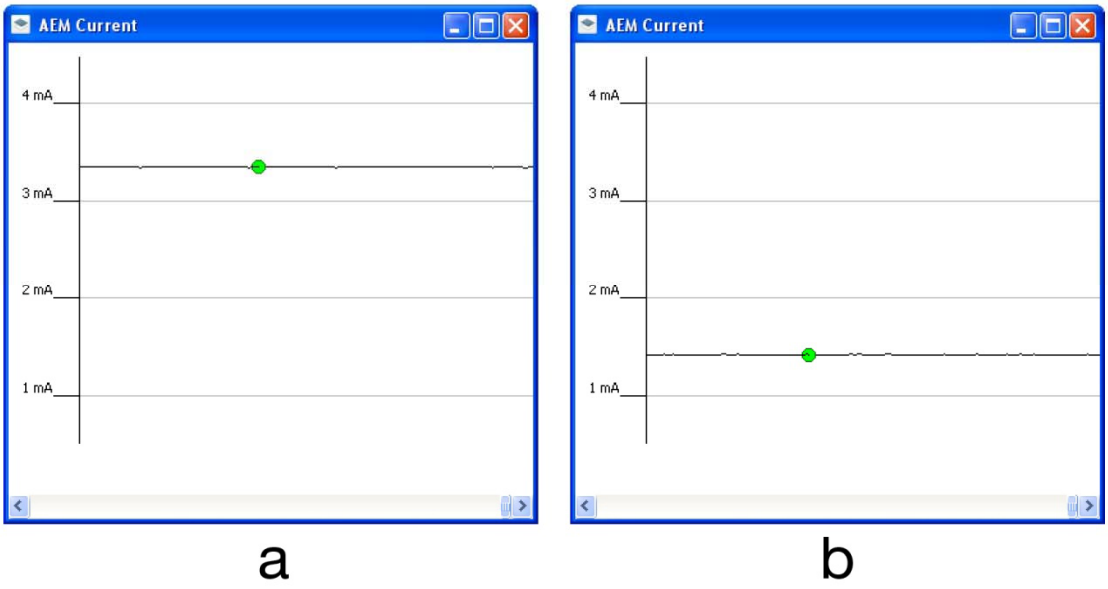


Figure 2: La consommation en mode actif (a) et en mode veille (b) engendrée par le polling du buffer LEURAT RX

Le profil de consommation de cette boucle de polling (présenté Figure 2a) affiche un courant constant de 3.33mA. En cliquant sur le graphique, la fonction à l'origine de la surconsommation est mise en évidence.

```
void pollLEUARTRx(void)
{
    while ( !( LEUART0 -> STATUS & LEUART_STATUS_RXDATAV ) );
}
```

La ligne de code surlignée n'est autre que la boucle de polling qui vérifie si la mémoire-tampon a réceptionné des données valides. L'analyseur donne également la liste de toutes les fonctions ainsi que les contributions de chacune d'entre elles à la consommation globale. Dans le cas présent, pollLEUARTRx() étant la seule fonction contenue dans le code, elle est seule responsable de toute la consommation d'énergie (Figure 3).

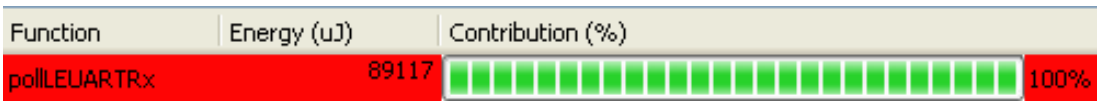


Figure 3: La consommation de la fonction pollLEUARTRx

Une manière classique d'éviter le polling du buffer RX consiste à mettre en œuvre les interruptions RX et de placer le MCU en mode sommeil. Si l'on valide l'interruption LEUART RX et si l'on place l'EFM32 en mode sommeil (avec le CPU inactif), on s'aperçoit immédiatement que l'économie d'énergie résultante est significative.

Une fois le processeur inactif, le courant électrique consommé chute en effet à 1,40mA (Figure 2b). A partir de maintenant, dès que le module LEUART réceptionne une trame d'octets, le processeur repasse en mode actif et réémet les données à travers le buffer TX.

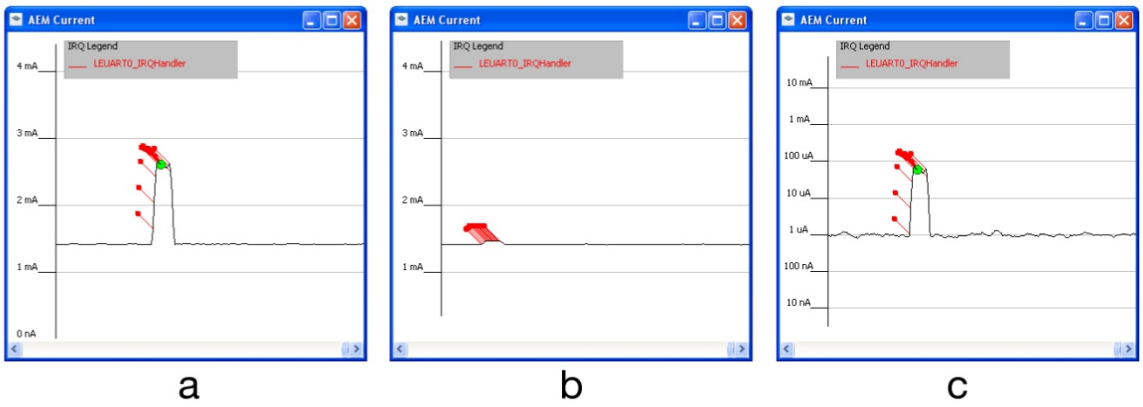


Figure 4: L'interruption LEUART RX avec polling du buffer LEUART TX (a), avec l'EFM32 en mode sommeil entre deux réceptions d'octets (b), et avec l'EFM32 en mode sommeil profond (c)

Une fois l'interruption déclenchée, la consommation monte brusquement jusqu'à 2,5mA environ et l'analyseur affiche les routines d'interruption en question (Figure 4a). On remarque alors que le courant électrique consommé reste longtemps bloqué sur cette valeur haute. En cliquant sur le graphique, il est possible de détecter une autre erreur classique dans la programmation des communications UART.

```
void pollLEUARTTx(void)
{
    while ( !( LEUART0 -> STATUS & LEUART_STATUS_TXC) );
}
```

Juste après l'envoi des données, le programme contient une boucle d'attente, le temps que la transmission se termine. Le processeur reste donc en mode actif plus longtemps qu'il n'est nécessaire. La boucle peut être avantageusement remplacée par une interruption qui réveille le processeur au moment où la transmission s'achève. Une telle option diminue encore la consommation (Figure 4b).

Dès lors, le processeur passe en mode sommeil entre chaque octet transmis, abaissant d'autant la consommation moyenne. La transmission des données se fait sans intervention du processeur et il n'est plus nécessaire de sonder le buffer pour connaître le moment où la transmission s'achève. Remplacer la boucle par une routine d'interruption est une solution plus élégante et plus éco-efficace, comme le prouvent les différents profils de consommation associés aux deux approches.

A noter que le module LEUART des microcontrôleurs EFM32 peut fonctionner même en mode sommeil profond. Dans ce mode, les oscillateurs à haute fréquence sont inactifs et seuls les oscillateurs à basse fréquence (quartz ou RC) fonctionnent encore et cadencent le

LEUART. Dans l'exemple précité, si l'EFM32 est placé en mode sommeil profond, la consommation d'énergie chute à des niveaux de l'ordre du µA.

Pour visualiser ces faibles valeurs de courant, l'analyseur est mis en mode d'affichage logarithmique et non plus linéaire. Le courant est désormais de 1µA en mode sommeil profond et de seulement 80µA lors de la réception des trames (Figure 4c).

Entre la programmation logicielle initiale et la configuration finale, l'économie d'énergie a pu être réduite d'un facteur 1000 !

Au sujet d'Energy Micro

Energy Micro conçoit, développe et commercialise les microcontrôleurs les plus énergétiquement efficaces au monde, basés sur l'architecture 32 bits ARM® Cortex-M3. La société a été créée en 2007 par des spécialistes de l'industrie des semiconducteurs, forts d'une expertise acquise chez Chipcon, Texas Instruments, Atmel et Nordic Semiconductor. Pour plus d'information, rendez-vous sur le site www.energymicro.com.