Microprocesseur

Introduction

L'objectif de cette série de travaux pratiques est de vous faire découvrir l'usage d'un système à microprocesseur 32 bits plus puissant que le PIC 16F84 dans ses possibilités, mais également plus complexe dans sa mise en œuvre. Cette introduction a pour but de vous familiariser avec certains nouveaux concepts, et de permettre d'en réviser certains autres.

Avant d'écrire chaque programme, vous rédigerez un algorithme suffisamment commenté sur papier. En fin de séance, vous rendrez les algorithmes de chaque programme ainsi que les observations qui s'y rapportent s'il y a lieu.

Sur le site http://bornat.vvv.enseirb.fr , rubrique 'E2 - TPs de micro-contrôleur', vous trouverez les ressources nécessaires au suivi de ces TPs. Toute information venant de votre encadrant est prioritaire sur les textes qui vous sont fournis.

1) Environnement de travail

Bien que les TPs soient affichés TP microprocesseur, nous utiliserons des microcontrôleurs pour les réaliser, comme dans le cas des TPs de première sur PIC. La différence vient du fait qu'un microcontrôleur constitue à lui seul un système complet (processeur, RAM, périphériques de communication). Nous utiliserons des microcontrôleurs de la famille AT91SAM7X conçus et commercialisés par ATMEL sous licence ARM. Ces processeurs sont très semblables aux processeurs de serveur dans leur utilisation. Ils ont cependant été développé avec l'objectif de réduire la consommation globale du système le plus possible. À ce titre, on les retrouve plus particulièrement dans les applications embarquées (Téléphones portables, box ADSL, points d'accès WiFi...)

La carte de travail est un kit de développement conçu par le constructeur. Sa référence est AT91SAM7X-EK, elle est équipée du microcontrôleur AT91SAM7X256.

La structure interne des AT91SAM7 est donnée par la figure 1. On y retrouve le processeur qui accède aux mémoires et aux périphériques à travers le contrôleur mémoire. Pour des raisons de performance énergétique (rapport entre la puissance de calcul et la puissance électrique consommée), les bus de mémoire Flash, mémoire vive et périphériques sont électriquement disjoint. Vu du processeur, il n'existe cependant qu'un seul bus, et une seule représentation des adresses.

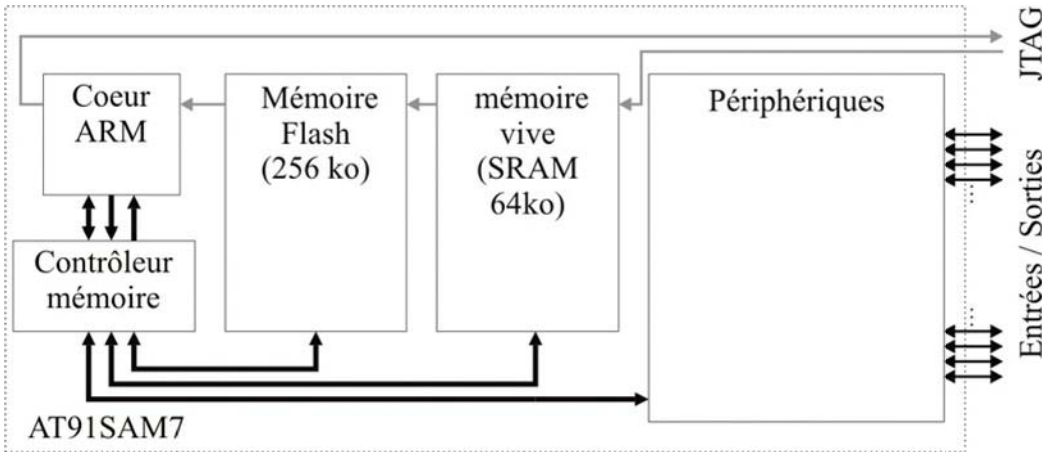

Figure 1: Structure interne du microcontrôleur AT91SAM7

Les Processeur ARM utilisent un bus mémoire 32 bits cadré. Ils peuvent être configurés pour utiliser la mémoire en mode décadré, mais ce fonctionnement ne s'applique pas aux périphériques. Les données sont stockées au choix en mode Big ou Little Endian. Pour information, nous fonctionnerons en mode Little Endian, mais ce paramètre ne devrait avoir aucune influence sur vo programmes.

Les TPs se feront en environnement de *cross-compilation*. C'est à dire que les phases d'écriture du programme et de compilation se font sur une machine d'architecture différente de celle où le programme est exécuté. Ici, ces phases se feront sur votre poste de travail PC. Ensuite seulement, le programme compilé est envoyé sous forme binaire dans la mémoire vive du microcontrôleur pour y être exécuté. L'envoi se fait au moyen du bus JTAG. Ce bus est dédié au contrôle du composant et n'est pas accessible depuis le composant lui-même.

Les différentes liaisons sont schématisées par la figure 2. Le boîtier ICE (bleu) permet de contrôler le kit (par le bus JTAG) depuis le poste de travail (par le bus USB). Le kit dispose de sa propre liaison USB qui ne nous servira que pour l'alimentation électrique de la carte.

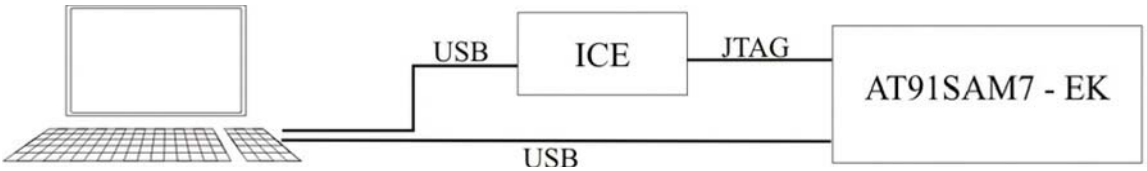

Figure 2: Connexions du kit de programmation

Le bus JTAG permet de charger des programmes dans les mémoires du microcontrôleur, de déboguer le programme pendant son exécution, ainsi que de pratiquer des tests électriques (entre autres).

2) Accès aux périphériques du microcontrôleur

Comme il a été vu en cours et dans les TPs précédents, seul le bus de données permet au processeur de communiquer avec le monde extérieur. Cela signifie que, selon l'adresse qui leur est associée, les données sont prises en compte soit par la mémoire, soit par un autre composant (unique pour chaque adresse). En conséquence, tout périphérique, quel qu'il soit, est vu comme une zone

mémoire par le processeur. La configuration de ces périphériques s'effectue par les valeurs qui seront envoyés dans les différents registres.

Chaque registre d'un périphérique correspond à une fonction précise. Le registre auquel les données sont envoyées est déterminé par son adresse (ou pointeur). Les registres d'un même périphérique sont regroupés dans une zone. La plus faible de ces adresses (le pointeur) est appelé adresse de base du périphérique.

Les différentes adresses de chaque périphériques sont définies par le manuel du composant (ou *datasheet*). Ces valeurs changent selon la catégorie et la génération. Pour produire du code portable d'un composant à l'autre, un fichier à inclure définit l'ensemble de ces valeurs aux moyens de structures, de macros et de constantes. Dans le cadre de ces TPs, nous utiliserons deux de ces fichiers : AT91SAM7X-EK.h et AT91SAM7X256.h, le premier appelant le second. Le fichier AT91SAM7X256.h contient toutes les définitions de périphériques internes au microcontrôleur (Adresses de base, adresses de registres, masques de bits...). Le fichier AT91SAM7X-EK.h contient les définition propres à la carte de développement (position des LEDs, Boutons...). Ainsi, pour chaque périphérique est défini un type construit. Ce type contient un champ pour chaque registre de configuration. Il est construit de telle sorte que sa représentation mémoire coïncide avec celle du périphérique concerné. Il reste alors à déclarer un pointeur constant vers cette structure. La valeur de ce pointeur est égale à l'adresse de base du périphérique. Dès lors, écrire (ou lire) dans un registre consiste à affecter (ou consulter) une valeur de cette structure.

3) Manipulation des bits en C

La plupart du temps, les paramètres qui permettent de configurer un périphérique sont contrôlés par un seul bit d'un registre donné. Les autres bits du registre codant d'autres paramètres, il faut faire attention à ne pas les modifier si ce n'est pas nécessaire. C'est pourquoi il est essentiel de savoir manipuler les bits en C.

a) masquage de bits

La première source d'erreur est la confusion entre les numéros de bits et leur position. Comme pour les tableaux en C, les bits sont numérotés à partir de 0. Donc le premier bit est le numéro 0, le 32^{e} bit le numéro 31. De façon générale : le n^{ieme} bit est le bit numéro *n-1*.

Seuls des accès 32 bits sont possibles, donc, pour modifier un seul bit d'un registre, il faut :
- lire la valeur du registre entier,
- modifier le bit correspondant grâce à des opération logiques
- écrire la valeur modifiée du registre entier.

Par exemple, si on veut mettre à 1 le 5^{e} bit d'un registre et à 0 son 7^{e} bit.
- on lit d'abord la valeur du registre que l'on met dans une variable temporaire (tmp)
  tmp vaut : xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx (32 bits de valeur inconnue)
- on modifie le bit 4 (qui est le 5^{e} bit)
  - il faut créer la valeur A = 00000000 00000000 00000000 00010000
  - l'opération bit à bit 'tmp OU A' va donner :
    tmp = xxxxxxxx xxxxxxxx xxxxxxxx xxx1xxxx

(une valeur inconnue OU 1 vaut 1, une valeur inconnue OU 0 laisse cette valeur inchangée)

- on modifie le bit 6 (qui est le 7^{e} bit)

- il faut créer la valeur B = 00000000 00000000 00000000 01000000
- l'opération bit à bit 'tmp ET NON B' va donner :

tmp = xxxxxxxx xxxxxxxx xxxxxxxx x0x1xxxx

(une valeur inconnue ET 0 vaut 0, une valeur inconnue ET 1 laisse cette valeur inchangée)

- il ne reste qu'à écrire tmp dans le registre concerné.

A et B sont appelés masques. Ce sont des valeurs exprimées sur 32 bits qui permettent de ne manipuler qu'un bit à la fois. Il est possible de faire un masque pour plusieurs bits à la fois, mais il ne sera alors plus possible d'effectuer des opérations différentes.

b) syntaxe de gestion des masques en C

Il y a deux représentations pour les opérations logiques en C. Chacune de ces représentations code un opérateur différent.

- opérateurs booléens (logiques) :

ces opérateurs sont prévus pour agir sur des entiers considérés comme valeurs booléennes (un entier de valeur 0 est interprété comme FAUX et toute autre valeur interprétée comme VRAI). S'ils renvoient FAUX, ces opérateurs renvoient 0, sinon, ils renvoient 1.

||	représente l'opération OU
&&	représente l'opération ET
!	représente l'opération NON

ATTENTION : ces opérateurs ne doivent pas être utilisés avec des masques car ils considèrent les valeurs entières (sur 32 bits).

- opérateurs bits à bits :

ces opérateurs sont prévus pour agir sur des entiers considérés comme supports de bits. L'opération est effectuée sur le n^{eme} bit de chaque argument et le résultat est renvoyé dans le n^{eme} bit de l'entier renvoyé.

|	représente l'opération OU,
&	représente l'opération ET,
^	représente l'opération OU EXCLUSIF
~	représente le NON

exemple de différence de comportement :

avec un ET bit à bit :	0010 ET 0001	donne 0000
avec un ET logique :	0010 ET 0001 (*VRAI* ET *VRAI*)	donne 0001 (*VRAI*)

c) création d'un masque

La création d'un masque se fait soit en écrivant une constante (par exemple A= 0x0010) soit en effectuant des décalages de bits (A = 1 << 4).

Les opérateurs << et >> effectuent des décalages de bits vers la gauche ou vers la droite respectivement. Étant donné que la valeur 1 se code en binaire : 0000000000000001 (sur 16 bits par exemple), en effectuant des décalages de la valeur '1' on décale un et un seul bit.

Par exemple :

(1 << 0) représente un masque dont seul le bit 0 est à 1.

(1 << 2) représente un masque dont seul le bit 2 est à 1 (le troisième bit).

(1 << n) représente un masque dont seul le bit n est à 1 (le $(n+1)^{eme}$ bit).

D'un point de vue mathématique, << est équivalent à une multiplication par une puissance de deux. En effet :

n<<0 = n;
1<<3 = 2<<2 = 4<<1 = 8<<0 = 8
n << m = 2*n << m-1 = $n*2^m$;

de même, l'opérateur >> est équivalent à une division par une puissance de deux :

8>>3 = 4 >> 2 = 2 >> 1 = 1 >> 0 = 1;
n >> m = n/2 >> m-1 = $n/2^m$;

d) combinaison de masques

Il est possible de créer un nouveau masque de bits en combinant plusieurs autres masques. Par exemple, s'il faut mettre a 1 des bits représentés par les masques A, B et C. Il est plus immédiat d'effectuer une seule opération de masquage avec un masque construit par (A | B | C).

e) test d'un unique bit à l'aide d'un masque

Il est souvent nécessaire de tester un bit bien défini dans un entier de 32 bits. Pour cela les masques associés au fonctionnement du C sont d'une grande aide.

Supposons qu'il faille tester le bit 9 de tmp dont la valeur est inconnue : Le masque de ce bit est M=1<<9. L'opération tmp2 = (tmp & M) met à 0 tous les autres bits (ceux qui ne nous intéressent pas). *tmp2* est donc nul si et seulement si le bit 9 est nul.

4)Gestion des masques avec les processeurs ARM

La famille de microcontrôleurs AT91 utilise un bus d'adresses de 32 bits, soit 4Giga adresses. La mémoire disponible utilise très peu de cet espace d'adresses, ce qui laisse une grande partie de cet espace pour les périphériques. Afin d'optimiser la performance énergétique, certains registres proposent une interface en rupture avec l'usage dans le monde des microcontrôleurs classiques. Le principe est d'utiliser des mémoires de type RS pour les bits de configuration des registres (au lieu de bascules D). Les avantages sont multiples :

- une mémoire RS requiert moins de transistors à réaliser qu'une bascule D (gain en surface et en consommation) cf TD TCIN sur les composants numériques intégrés .
- une mémoire RS fonctionne sans horloge (gain en consommation et en performances)
- L'ensemble des signaux de mise à 1 (Set) et ceux de mise à 0 (Reset) sont accessibles à deux adresses différentes. Sur chacune de ces adresses, l'écriture d'un 0 est sans effet. On écrit donc directement les masques dans les registres pour modifier les bits isolés. Les opérations de masquage de bits logicielles deviennent inutiles (économies en puissance de calcul)

L'inconvénient principal de cette technique est que chaque registre nécessite plusieurs adresses. C'est très limitant sur un système ayant un nombre d'adresses limité (adresses sur 16 bits par exemple) ou doté de beaucoup de mémoire (PC / serveur). Nous ne sommes pas dans cette situation.

Certains registres obéissent donc à ce système (pas tous). Si c'est le cas pour un registre, il possède au moins trois adresses d'accès :

- mise à un des bits (écriture de 1s correspondants; 0 est sans effet : Enable Register)
- mise à zéro des bits (écriture de 1s correspondants; 0 est sans effet : Disable Register)
- lecture de la valeur du registre (Status Register), ce registre est en lecture seule.

Ces accès permettent de manipuler les registres du microcontrôleur sans effectuer les opérations de

masquage coté logiciel. Ils réduisent ainsi le nombre d'opérations à effectuer et les sources d'erreur. Pour modifier ce genre de registre, il suffit d'écrire directement les masques dans les accès de mise à zéro et de mise à un. La valeur écrite n'est pas mémorisée, mais envoyée sur les entrées de mise à zéro ou mise à un du périphérique. Il reste possible de consulter la valeur du registre à la troisième adresse.

ATTENTION :
Les accès de mise à zéro et mise à un sont en écriture seule. La valeur obtenue en lisant cette adresse ne provoque pas d'erreur système, mais n'est pas documentée, elle doit donc être considérée comme aléatoire. Au mieux, elle sera constante, dans tous les cas, elle sera indépendante de la valeur du registre. De même, l'accès de lecture est un accès en lecture seule. Toute écriture dans cet accès sera sans effet.

5)Apprivoiser la Documentation

Le document de description technique du micocontrôleur AT91SAM7X fait 671 pages. Savoir utiliser ce document est essentiel pour mener à bien les TPs (et fait partie de l'objectif des compétences à acquérir).

- Les chapitres 1 à 9 présentent le microcontrôleur dans son ensemble en tant que composant.
- Le chapitre 10 présente les différents modules et leurs méthodes d'interaction
- Les chapitres suivants présentent chacun un module du microcontrôleur.

Lire la documentation de façon linéaire comme un livre de chevet est inutile. N'hésitez pas à consulter la table des matières pour accéder directement à l'information recherchée. Lorsque vous avez besoin d'une information, elle est toujours liée à un module, tous les modules sont décrits de la même manière.

- Chaque chapitre présentant un module commence par une brève section qui en résume les fonctionnalités.
- Un diagramme est généralement présenté à la suite pour montrer comment ce module communique avec les autres (ou en dépend).
- Vient ensuite une description plus détaillée sur le comportement du module, avec généralement une section « Functional Description » qui en donne une vue schématique. Cette section décrit l'utilité de chaque registre en détail, pour chaque mode de fonctionnement. Il faut donc bien cerner le mode de fonctionnement qui vous intéresse pour ne pas perdre de temps sur les autres modes.
- La dernière section « User Interface » montre comment accéder aux registres et leur usage/comportement vu du processeur.

Par souci d'économie de papier, la version imprimée de la datasheet ne vous est pas proposée personnellement. Les exemplaires que vous utilisez ne doivent donc pas quitter la salle de TP, et doivent rester en bon état pour les binômes suivants. Veillez également à ne rien y écrire, les informations que vous laissez ne sont pas forcément pertinentes d'un TP à l'autre.

6)Lexique Anglais/Français

To set / To assert	: mettre à 1 / activer
To reset / To deassert	: mettre à zéro / désactiver
a nible	: un quartet (4 bits) / un digit hexadécimal
a field	: un champ (ensemble de bits dans un registre)