Sylvain
THIERY
GE3

NOTE D'APPLICATION

Cas pratique de mise en basse consommation d'un M32C87

Projet P12AB01 : **Carte télécommande d'un Pylône Eolien**

Projet Polytech Clermont-Ferrand
Janvier 2013

Client : Entreprise WINDELA représentée par M. AUDUBERT

Tuteur industriel : M. FICKINGER
Tuteur technique : M. LAFFONT

Année Universitaire 2012 – 2013

Table des matières

Introduction

Le but de cette note d'application est de présenter des cas pratiques pour mettre le microcontrôleur M32C87 de Renesas en basse consommation. Dans un premier temps, les techniques pour passer en basse consommation seront présentées. Ensuite, pour analyser l'efficacité de ces différentes techniques, nous présenterons un cas pratique. Enfin, nous ferons une synthèse sur les solutions possibles pour réduire la consommation avant de conclure.

1) Techniques de mise en basse consommation

Le microcontrôleur M32C87 donne la possibilité d'utiliser plusieurs techniques pour réduire la consommation.

1.1 Configuration des ports d'entrées/sorties

Par défaut, les ports d'entrées/sorties du microcontrôleur sont figurés en entrées. Si les broches sont inutilisées et déconnectées, ce sont des entrées flottantes qui peuvent donc être source de consommation. Pour éviter cela et ainsi réduire la consommation, il est conseillé de connecter ces entrées à la masse par l'intermédiaire d'une résistance de pull-down ou de configurer ces broches en sorties. Pour définir la direction des ports, il faut utiliser les registres Pdi.

1.2 Réduction de la fréquence et de la tension d'alimentation

D'une manière générale, la puissance consommée par les microcontrôleurs est principalement la puissance dynamique. C'est la puissance qui est consommée lors des commutations des transistors de technologie CMOS. Lors de ces commutations, la tension aux bornes d'un transistor et le courant qui le traverse sont différents de 0 pendant un temps faible. Le transistor consomme donc une puissance pendant ces temps de commutation. Cette consommation est liée à la formule suivante :

$$\mathbf{P = \alpha * C * Vdd^2 * f}$$

avec :

- **α** : nombre de transitions logiques par cycle d'horloge,
- **C** : paramètre qui dépend de la technologie du circuit,
- **Vdd** : tension d'alimentation,
- **f** : fréquence.

On constate donc que si l'on veut baisser cette consommation, il faut réduire la tension d'alimentation et la fréquence. Les autres paramètres sont fixes car ils dépendent de la technologie du circuit et de l'algorithme que l'on souhaite implanter dans le microcontrôleur.

1.4 Mode wait

Le M32C87 possède un mode appelé mode « wait » qui permet de couper la fréquence du CPU et arrêter le watchdog timer. Lorsque cette fréquence est coupée,

les instructions du programme ne s'exécutent plus et le système passe dans un état d'attente. Les fréquences utilisées par les périphériques restent actives. Le CPU se réveille lors d'un reset matériel ou par une interruption générée par un périphérique et reprend l'exécution des instructions à l'endroit où il s'était arrêté.

Le niveau de priorité des interruptions doit être supérieur au niveau défini dans le registre RLVL pour pouvoir réveiller le microcontrôleur. Avant de passer dans le mode wait, il faut donc activer toutes les interruptions des périphériques susceptibles de réveiller le M32C87 et désactiver celles qui ne doivent pas le réveiller, cette activation se fait dans le registre d'interruption du périphérique. Ensuite, il faut passer dans le mode wait en exécutant l'instruction assembleur « wait ». Voici la démarche à suivre pour passer en mode wait correctement.

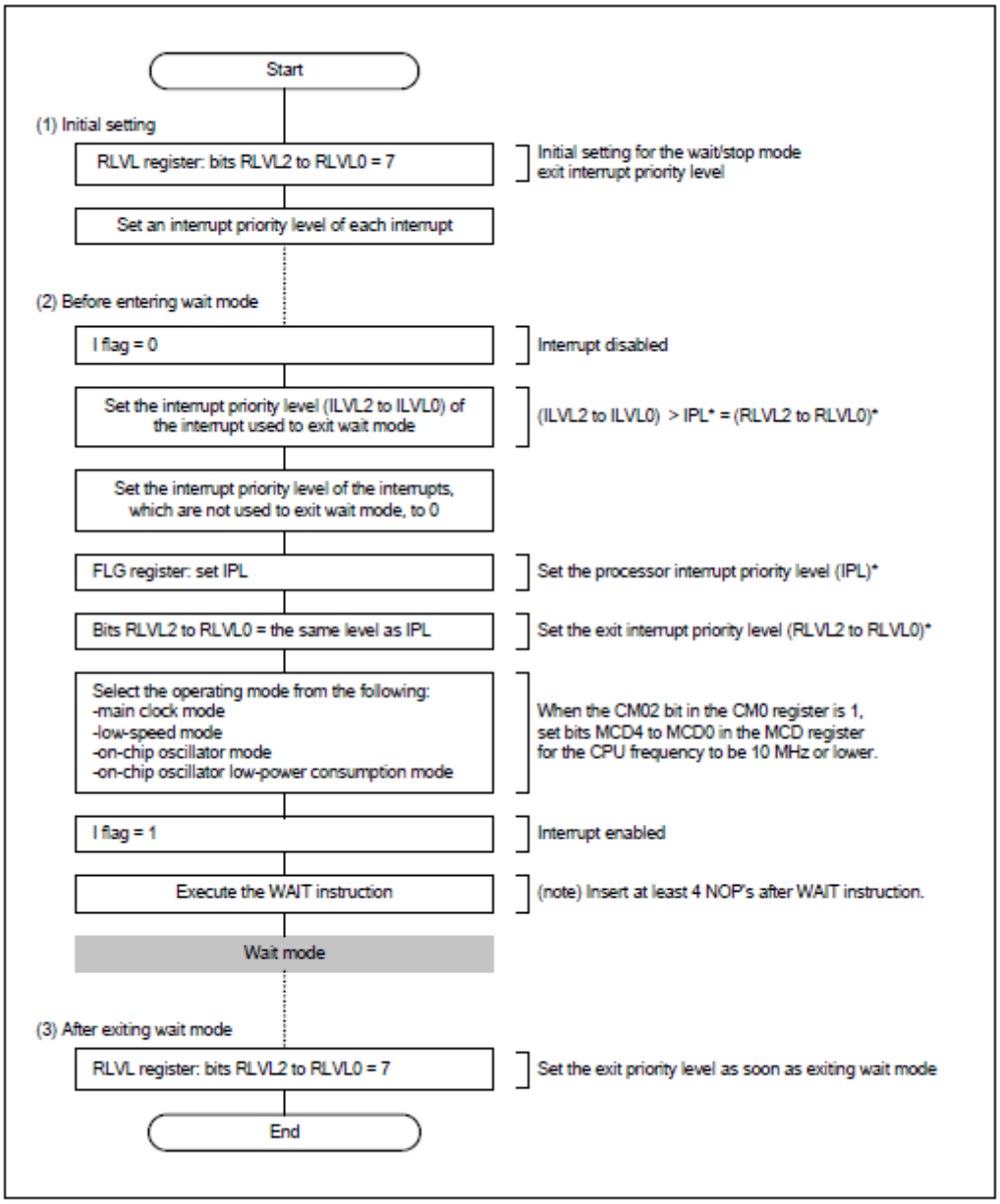


Figure 9.14 Procedure to Enter Wait Mode

1.5 Mode stop

Le M32C87 possède un mode appelé mode «stop» qui permet de couper la fréquence du CPU , d'arrêter le watchdog timer et d'arrêter les horloges des périphériques. Lorsque ce mode est lancé, les instructions du programme ne s'exécutent plus et le système passe dans un état d'attente. Les fréquences utilisées par les périphériques sont inactives. Le CPU se réveille lors d'un reset matériel ou par une interruption $\overline{INT}$ qui s'active lors d'un changement d'état sur une broche d'interruption. Une interruption des timers peut aussi réveiller le CPU mais ces timers doivent fonctionner à l'aide d'une autre horloge que celle du quartz principale. Cette horloge doit être connectée aux broches XCIN et XCOUT et doit être inférieure ou égale à 100 Hz. Le niveau de priorité des interruptions doit également être supérieur au niveau défini dans le registre RLVL pour pouvoir réveiller le microcontrôleur comme dans le mode wait. Le passage en mode stop s'effectue en passant le bit cm10 du registre cm1 à l'état haut. Voici la démarche à suivre pour passer en mode stop correctement.

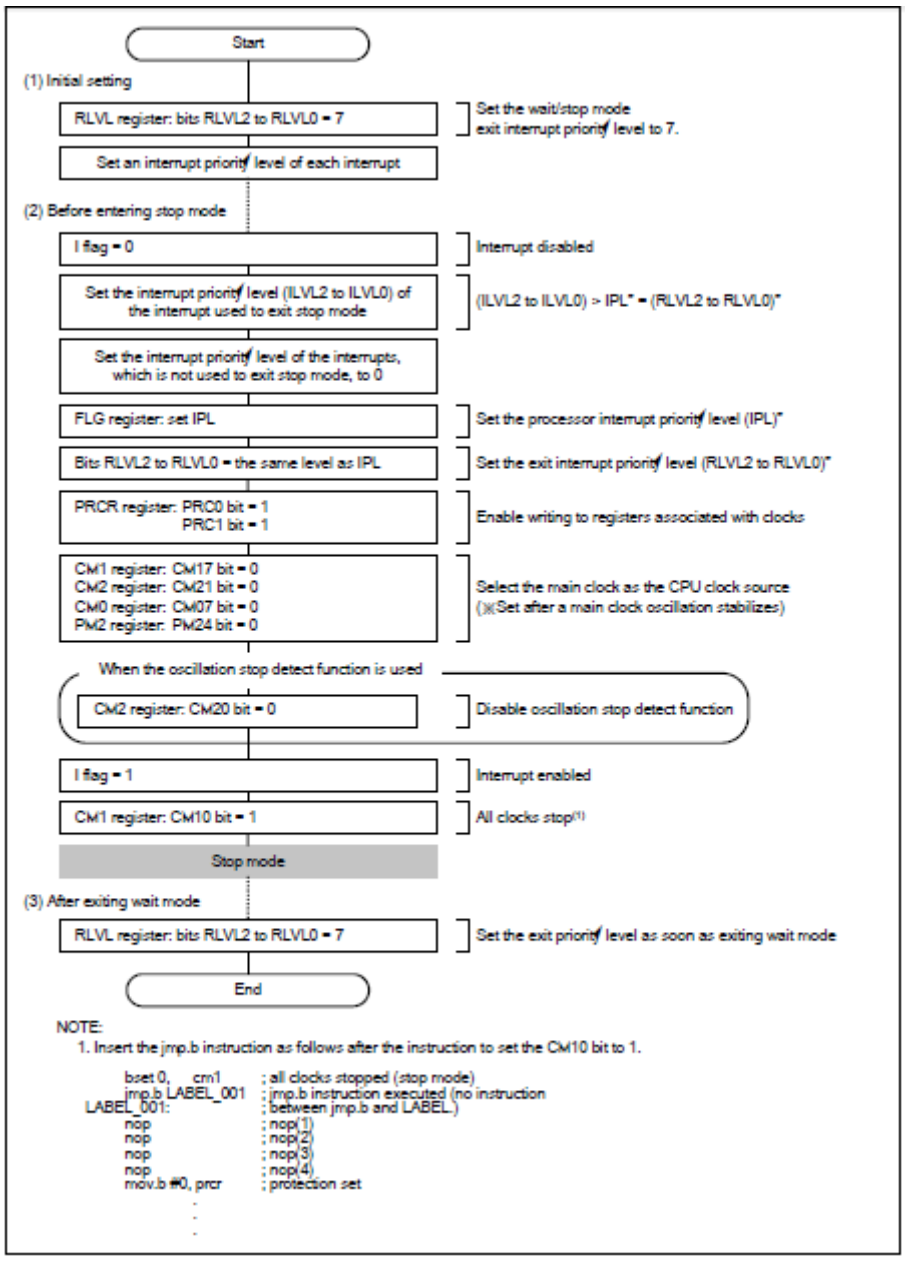


Figure 9.15 Procedure to Enter Stop Mode

2) Cas pratique

Pour analyser les performances de ces techniques, une manipulation a été effectuée sur une carte de test. Les résultats obtenus suite à cette manipulation seront ensuite analysés dans cette partie.

2.1 Présentation de la manipulation

La manipulation consiste à utiliser toutes les techniques de mise en basse consommation du microcontrôleur M32C87 sur un programme simple.

Le programme qui a été choisi est très simple, il consiste à générer un chenillard sur le port 9. Sur les 8 bits du port, 7 bits sont à 0 et un seul bit est à 1. Toutes les secondes, le bit qui est à 1 est décalé à gauche sur le port 9. Lorsque c'est le bit 7 qui est à 1, une seconde plus tard, ce sera le bit 0 qui sera actif.

Ce programme utilise les timers B et A pour compter une seconde. Le timer A génère une interruption lorsque la seconde s'est écoulée.

Sur une itération de la boucle principale de ce programme, un décalage du chenillard est effectué puis il y a une attente d'une seconde grâce au timer. Le code de ce programme est présent en annexe.

Les techniques de mise en basse consommation au niveau logiciel ont été testées à savoir la mise en sortie des broches inutilisées, l'utilisation du mode wait et l'utilisation du mode stop.

Les techniques de mise en basse consommation au niveau matériel ont également été testées à savoir la réduction de la fréquence du quartz (20MHz, 10 MHz, 1MHz) et la réduction de la tension d'alimentation (5V, 4V, 3V).

Cette manipulation a été effectuée sur une carte de test ne comportant que le microcontrôleur, le quartz et le montage de connexion du debugger. La tension d'alimentation est fournie par une alimentation externe réglable, il n'y a donc aucun composant externe qui peut fausser les résultats, le schéma de la carte est présent en annexe.

Les changement de programme ont été flashés dans le microcontrôleur pour pouvoir déconnecter le débugger et supprimer la consommation qu'il engendre. Un ampèremètre a été placé entre l'alimentation et la carte pour mesurer le courant consommé.

2.2 Résultats obtenus

Résultats pour un quartz de 20 MHz :

Tension	Consommation obtenue			
	Pas de mode, Ports flottants	Pas de mode, Ports en sortie	Mode wait, Ports en sortie	Mode stop, Ports en sortie
5V	26.1 mA	25.2 mA	6.16 mA	2.7 mA
4V	23 mA	22.8 mA	3.7 mA	0.9 mA
3V	21.59 mA	21.58 mA	2.45 mA	0.05 mA

Résultats pour un quartz de 10 MHz :

Tension	Consommation obtenue			
	Pas de mode, Ports flottants	Pas de mode, Ports en sortie	Mode wait, Ports en sortie	Mode stop, Ports en sortie
5V	17.8 mA	17.34 mA	5.48 mA	2.75 mA
4V	14.77 mA	14.48 mA	2.57 mA	0.95 mA
3V	13.6 mA	13.3 mA	1.37 mA	0.04 mA

Résultats pour un quartz de 1 MHz :

Tension	Consommation obtenue			
	Pas de mode, Ports flottants	Pas de mode, Ports en sortie	Mode wait, Ports en sortie	Mode stop, Ports en sortie
5V	9.9 mA	8.2 mA	5.5 mA	2.75 mA
4V	5.3 mA	4.7 mA	1.81 mA	1 mA
3V	3.4 mA	3.33 mA	0.33 mA	0.03 mA

2.3 Analyse critique des résultats

Tout d'abord, d'un point de vue matériel, les réductions de l'alimentation et de la fréquence sont très efficaces lorsqu'aucune techniques logicielle est utilisée puisque l'on passe de 26.1 mA (20 MHz, 5V) à 3.4 mA (1 MHz, 3V) soit une réduction de consommation de 87 %. Lorsque les modes de mise en basse consommation sont utilisés, la réduction de la tension reste toujours efficace, cependant la réduction de la fréquence est moins efficace surtout pour le mode stop, ce qui est cohérent puisque dans ce mode, toutes les fréquences sont stoppées.

Pour ce qui est des techniques logicielles, on observe que configurer les ports

en sortie permet de gagner au maximum 1 mA, ce qui est peu mais très facile à mettre en place.

Le mode wait permet une réduction très importante et le programme reste fonctionnel, la réduction de consommation peut atteindre 78% pour une tension et une fréquence fixe par rapport à un programme non optimisé. Ce mode est donc très intéressant à utiliser pour tous types d'algorithme qui a des phases d'attentes puisque toutes les interruptions peuvent réveiller le microcontrôleur.

Le mode stop permet une très forte réduction de consommation (jusqu'à 89%) et la réduction est indépendante de la fréquence du quartz. Cependant, le programme n'est plus fonctionnel car le CPU reste bloqué lorsque le mode stop est appelé. La fonction d'interruption du timer ne peut pas réveiller le microcontrôleur car l'horloge sur lequel le comptage est effectué est stoppée. Les timers peuvent fonctionner en mode stop seulement avec l'utilisation d'un autre quartz à faible fréquence. Ce mode est donc le plus performant d'un point de vue consommation mais il ne peut pas être utilisé pour tous types d'application.

3) Synthèse

Si on souhaite avoir un système qui consomme peu, il faut le prendre en compte dès la conception de la carte en baissant la tension le plus possible, la réduction de la fréquence est aussi efficace mais cette solution n'est pas conseillée si on souhaite avoir un programme rapide et performant d'un point de vue temporel.

Au niveau des améliorations logicielles, la mise en sortie des ports inutilisés doit de se faire de manière systématique car même si on gagne peu en réduction de consommation, cette solution est simple à mettre en place et n'influe pas sur l'algorithme implanté.

Si le programme possède des phases d'attente, l'utilisation du mode wait est assez efficace car il permet de réduire fortement la consommation tout en gardant les périphériques en état de fonctionnement, le système restera donc réactif à son environnement et pourra se réveiller si un événement intervient.

Pour terminer le mode stop ne peut pas être utilisé sur n'importe quelle application car les périphériques deviennent inactifs lorsque l'on rentre dans ce mode. Cependant, il peut s'avérer être la solution la plus efficace car la réduction de consommation que ce mode engendre est indépendant de la fréquence du quartz, on peut donc obtenir un système très rapide en fonctionnement et qui consomme très peu en veille. Ce mode pourrait être utilisé pour une télécommande par exemple avec des boutons reliés à une broche d'interruption du M32C87. L'interruption générée par l'appui sur un bouton réveillerait le CPU pour traiter la requête rapidement.

Conclusion

Pour conclure le M32C87 offre plusieurs possibilités pour baisser sa consommation. Cependant la mise en basse consommation est une chose qu'il faut prendre en compte dès le début d'un projet car cela influe beaucoup sur la schématique de la carte électronique à concevoir mais aussi sur le programme qui doit être implanté dans le microcontrôleur. Enfin, en cas de forte contrainte sur la consommation d'un système, il est préférable d'utiliser certaines familles de microcontrôleurs plus récentes comme la famille RL78 de Renesas qui offre de meilleures performances sur la consommation, leurs consommations ne dépassent pas 5 mA même sans mode basse consommation et pour des hautes fréquences.

ANNEXES

Programme de test

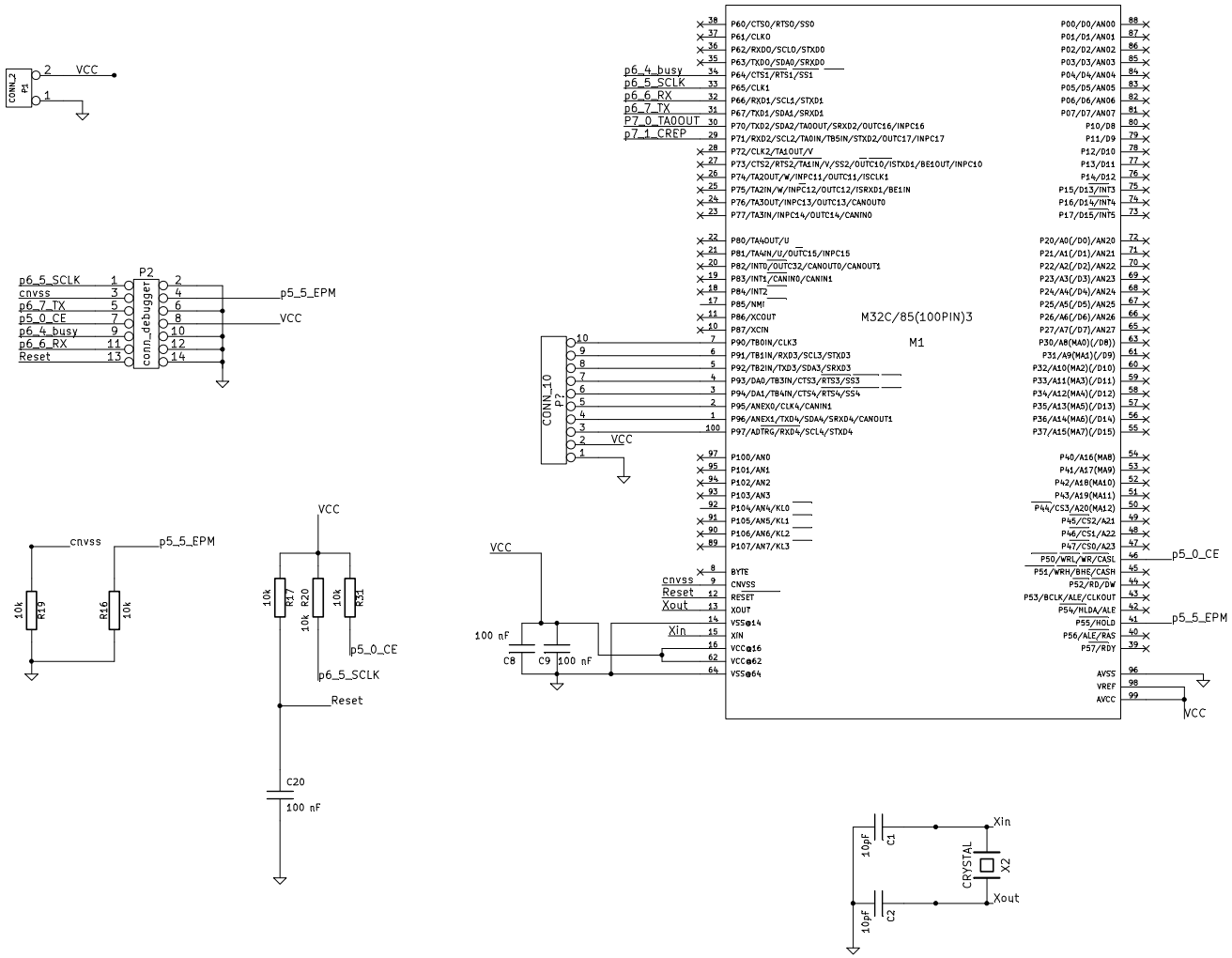
fichier initialisation.c

```
#include "sfr32c87.h"
#include "initialisation.h"

// Initialisation des ports utilises en laissant les ports inutilisés flottant
void init_port(void)
{
    asm("fclr I");
    pd5 |= 0xDE;
    pd6 |= 0x0F;
    pd8 |= 0xDF;
    prc2 = 1;
    pd9 = 0xFF;
    prc2 = 0;
    asm("fset I");
}

// Initialisation des ports utilisés avec mise en sortie des ports inutilisés
void init_port_BC(void)
{
    asm("fset I");
    pd0 = 0xFF;
    p0 = 0x00;
    pd1 = 0xFF;
    p1 = 0x00;
    pd2 = 0xFF;
    p2 = 0x00;
    pd3 = 0xFF;
    p3 = 0x00;
    pd4 = 0xFF;
    p4 = 0x00;
    pd5 |= 0xDE;
    pd6 |= 0x0F;
    pd7 = 0xFF;
    pd8 |= 0xDF;
    prc2 = 1;
    pd9 = 0xFF;
    prc2 = 0;
    pd10=0xFF;
    asm("fset I");
}

void init_clock(void)
{
    prc0 = 1;
    mcd = 0x12; // f = fréquence du Quartz, pas de division
    pm2 = 0;    //Main Clock utilisé
    cm0 = 0;    //Sub clock desactivé Main clock utilisé
    cm1 = 0x20; //Main Clock utilisé pas d'arret (stop mode) PLL desactivé
    cm2 = 0;    //On chip oscilator desactivé
    plc0 = 0x50;    //PLL stoppe
    prc0 = 0;
}
```

```
void init_tb2(void)
{
    tb2s=0;
    tb2mr =00; // tb2 en mode timer avec f = Quartz
    // On initialise le compteur pour compter 1ms
    tb2 = 9999; //Quartz 20 MHz et 10 MHz : 9999, Quartz 1 MHz : 999
    trgsr = 0x55;   //TAi compte sur overflow de tb2
}


void init_ta3(void)
{
    ta3mr = 0x01;   // ta3 en mode event counter mode avec event = overflow de tb2
    ta3ic = 0x04;       //Autorise interruption
    //ta3 cadence la seconde (m+1)*tb2_periode
    ta3=1999;   //Quartz 20 MHz : 1999, Quartz 1 MHz et 10 MHz : 999
}


void start_wait_mode(void)
{
    rlvl = 0x07;        //une interruption superieur a 0 réveille le processeur
    ta3ic = 0x02;       //interuption de timer activé
    asm("fclr I");      //interdit les interruptions
    asm("LDIPL #0");
    rlvl = 0x00;
    prc0 = 1;
    cm02 = 0;   //frequence des périphérique active
    mcd = 0x00;
    prc0 = 0;
    //une interruption superieur a 0 fera sortir le processeur du mode d'attente
    asm("fset I");      //autorise les interruptions
    asm ("wait");       //Execution du mode wait
    asm ("nop");
    asm ("nop");
    asm ("nop");
    asm ("nop");

    asm ("nop");
}

void start_stop_mode(void)
{
    rlvl = 0x07;
    asm("fclr I");    //interdit les interruptions
    ta3ic = 0x02;     //interuption de timer activé
    asm("LDIPL #0");  //bit IPL = rlvl(3..0)
    rlvl = 0x00;      //une interruption superieur a 0 réveille le processeur
    asm("fset I");    //autorise les interruptions
    prc0 = 1;
    cm10 = 1;       //Execution du mode stop
    prc0 =0;
    asm("nop");
    asm("nop");
    asm("nop");
    asm("nop");
}
```

fichier principal

```
// Declaration des librairies
#include "sfr32c87.h"
#include "initialisation.h"

// Declaration des variables globales
char flag_seconde = 0;

// Timer cadence la seconde en passant un flag a 1
#pragma INTERRUPT 15 interup_ta3
void interup_ta3(void)
{
    //repasse a l'horloge du Quartz
    prc0 = 1;
    mcd = 0x12;
    prc0 = 0;
    flag_seconde = 1;    //Positionne le flag
    ta3s = 0;    //Eteind le compteur
}

void main(void)
{
    // Declaration des variables locales
    char i = 1;
    char tempon;

    // Initialisation
    init_clock();
    init_port();         //<- broches inutilisées laissées flottantes
    //init_port_BC();    <- broches inutilisées configurées en sortie
    init_tb2();
    init_ta3();


    // On autorise les interruptions
    asm ("fset I");

    // On demarre le timer
    tb2s=1;

    while(1)
    {
        p9 = i; //chenillard sur le port 9
        if ( i < 128 ) i<<=1;
        else i = 1;

        ta3s = 1;    //demarre comptage d'une seconde

        while(!flag_seconde);    //<- Attente d'une seconde sans mode de basse conso
        //start_wait_mode();     //<- Attente d'une seconde avec mode wait
        //start_stop_mode();     //<- Attente d'une seconde avec mode stop

        flag_seconde = 0;
    }
}
```



Thiery Sylvain		
POLYTECH – GE		
File: carte_receptrice.sch		
Sheet: /		
Title: carte test application note		
Size: A3	Date: 19 jan 2013	**Rev: 2**
KiCad E.D.A. eeschema (2011-04-29 BZR 2986)-stable		Id: 1/1