Muhammad Haziq KAMARUL AZMAN
Ludovic MIRANDA 30/05/2012

Systèmes à microprocesseurs

Travaux pratiques n°4 : UART

Objectif

Le but de ce TP sera de comprendre comment programmer l'UART de la carte LPC210x pour effectuer une transmission série.

Manipulation

Initialisation de l'UART

Tout d'abord, nous devons configurer la fréquence d'oscillation du montage cclk à un multiple du taux de transmission de l'UART. Pour cela, il faut modifier les registres du PLL.

Nous devons fixer la fréquence d'oscillation du montage cclk à 60MHz sachant que Fosc = 10MHz, et Fcco est compris entre [156 ; 320] MHz. En utilisant les formules trouvées dans le manuel (pg. 47) :

- cclk = M * Fosc
- Fcco = Fosc * M * 2 * P

nous avons trouvé que M = 6 et P = 2 pour pouvoir satisfaire les conditions données.

Nous avons ensuite écrit les valeurs dans le registre PLLCFG :

```
SCB_PLLCFG = 0x00000025; // Configuration de M = 6 et P = 2 pour fixer cclk = 60 MHz et Fcco = 240 MHz (voir pg. 45, 47, 48 du manuel)
```

A chaque fois qu'on effectue un changement, il faut écrire dans le registre PLLFEED pour valider ce dernier. Tout d'abord, on désactive les interruptions puis on écrit les séquences 0xAA et 0x55 successivement :

```
__disable_interrupt(); // Désactivation des interrupts (néssaires pour la mise de séquence d'actionnement (voir pg. 46, 47 du manuel)

SCB_PLLFEED = 0X000000AA; // Séquence d'actionnement pour sauvegarder les configs (voir pg. 46, 47 du manuel)
SCB_PLLFEED = 0x00000055;
```

```
__enable_interrupt(); // Réactivation des interrupts
```

Ensuite, on active le PLL en activant le bit 0 de PLLCON :

```
SCB_PLLCON = 0x00000001; // Activation du PLL (voir pg. 45 du manuel)

__disable_interrupt();

SCB_PLLFEED = 0X000000AA;
SCB_PLLFEED = 0x00000055;

__enable_interrupt();
```

Ensuite, il faut attendre que le PLL se synchronise avec la fréquence d'oscillateur, donc on met en place une boucle :

```
while(!(SCB_PLLSTAT && 0x00000400)) // Attente de la synchronisation du PLL avec la fréquence fixée(voir pg. 46 du manuel)
```

Après que le PLL soit synchronisé avec l'oscillateur, on active le PLL et on le connecte :

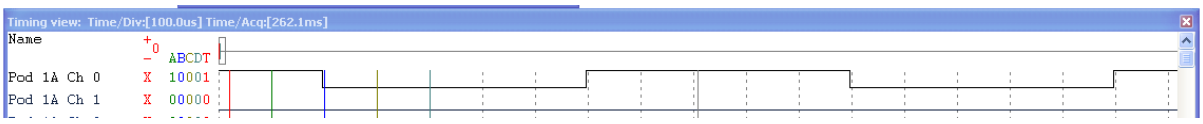
```
SCB_PLLCON = 0x00000003; // Activation du PLL (voir pg. 45 du manuel)

__disable_interrupt();

SCB_PLLFEED = 0X000000AA;
SCB_PLLFEED = 0x00000055;

__enable_interrupt();
```

Ensuite, on synchronise l'horloge de périphérique avec cclk :

```
SCB_VPBDIV = 0x00000001; // Synchronisation de pclk avec cclk (voir pg. 52 du manuel)
```

Pour tester la bonne configuration, nous avons utilisé la fonction *pwm_init()* qui permet de visualiser un signal d'horloge sur la sortie PWM4. Nous avons fixé un rapport de 60000 pour avoir un signal de 1kHz avec un rapport cyclique de 50%. Le code de cette dernière se trouve à la fin de ce compte rendu.

On vérifie que le signal est bien de fréquence 1 kHz.



Configuration de l'UART

Ensuite, il faut configurer l'UART0 en mode transmission à 8 bits de taux 9600 bauds.

Tout d'abord, nous avons configuré les ports pour pouvoir utiliser l'UART0 en transmission et réception :

```
PCB_PINSEL0 = 0x00000005; // Configuration des ports pour permettre la transmission et réception depuis les port P0.0 et P0.1 respectivement (voir pg. 78 du manuel)
```

Ensuite, il faut faire un reset des buffers en Tx et Rx. Il faut aussi mettre le bit 0 du registre FCR à 1 pour pouvoir accéder aux autres bits :

```
UART0_FCR = 0x07; // Reset des buffer de Tx et Rx (voir pg. 90 du manuel)
```

Ensuite, nous avons divisé l'horloge pclk par 390 (soit 0x186) pour avoir une horloge de transmission 16x plus grande que le taux de transmission. Cette valeur a été stockée dans un registre DLM et DLL. Pour y accéder, il faut activer le bit 7 du registre LCR :

```
UART0_LCR = 0x83; // Activation de l'accès aux latch diviseurs (voir pg. 91 du manuel)

UART0_DLM = 0x01; // Configuration du générateur de taux de transmission. Pour cela, nous allons diviser pclk par 16x9600 pour obtenir la facteur de division
UART0_DLL = 0x86; //  (voir pg. 87 du manuel)
```

Finalement, il faut désactiver le bit 7 du registre LCR et mettre l'UART0 en mode 8 bits avec 1 bit d'arrêt et sans bit de parité :

```
UART0_LCR = 0x03; // Configuration en mode 8 bits et désactivation de l'accès aux latch diviseurs (voir pg. 91 du manuel)
```

Muhammad Haziq KAMARUL AZMAN
Ludovic MIRANDA 30/05/2012

Transmission d'une chaîne de caractères

Après avoir configuré l'UART0 pour la transmission, nous avons écrit 3 programmes pour permettre la transmission :

1. *uart0_putc(char c)* qui permet de transmettre un caractère. Pour ce fait, nous avons utilisé le registre THR qui stockera le caractère à transmettre :

```
/* Fonction uart0_putc qui permet de transmettre une caractère
 */

void uart0_putc(char c)
{
    UART0_THR = (int)c; /* On affecte le caractère (en faisant un cast) dans
le registre UART0_THR */
}
```

2. *transmitter0_empty(void)* qui permet de tester si le registre THR est vide ou non. Nous nous sommes servis du registre LSR pour relever le bit d'indication (bit 5) :

```
int transmitter0_empty(void)
{
  return ((UART0_LSR & 0x20)==0x20);
}
```

3. *uart0_puts(char *s)* qui permet de transmettre une chaîne de caractère. Nous avons utilisés les fonctions définies précédemment :

```
void uart0_puts(char *s){
    int i=0; /* Compteur d'indice */

    while(*s) /* Tant que le transmetteur n'est pas vide, on transmet les
caratères une par une */
    {
        if(transmitter0_empty())
        {
            uart0_putc(*s);
            *s++;
        }
    }

    uart0_putc('\0'); /* On met le caratère '\0' pour terminer la
transmission */
}
```