
# Compte-rendu TP4
# MI43

Bertrand RIX
Alexis URVOY

Printemps 2011

# Introduction

Le but de la partie 4 de ce tp était de mettre en place un pilote utilisant des buffers pour envoyer et recevoir des données par la liaison série de la carte LPC (UART). Pour effectuer ceci nous disposions de l'OS FreeRTOS, très léger mais suffisant pour l'application.

En introduction à l'utilisation de FreeRTOS, il nous était demandé de créer une simple tâche allumant les LEDs de manière synchronisées (un chenillard de LEDs, plus un clignotement). Cela consistait simplement en la création d'une nouvelle tâche et la synchronisation de celle-ci et de la tâche de clignotement déjà écrite, avec deux sémaphores. Nous ne nous attarderons pas ici sur cette partie d'introduction et passerons directement à la partie concernant les buffers.

# I - Principe de la bufferisation

Pour l'envoi de données, il s'agit de placer dans un buffer la donnée à envoyer, dans la fonction fputc, et de générer une interruption qui va se charger d'envoyer la donnée. Ainsi plusieurs données peuvent être placé les unes à la suite des autres dans le buffer, en attente d'être envoyées sur la liaison série par la routine d'interruption. La difficulté ici était de savoir de quelle façon générer l'interruption.

En fait l'interruption sur l'envoi des données intervient au moment où on place la donnée à envoyer dans le registre U2THR (qui place la donnée sur le port série) alors qu'on aurait plutôt voulu que celle-ci intervienne au moment où on place une nouvelle donnée dans le buffer d'envoi. De ce fait nous n'avons pas pu utiliser les interruptions pour l'envoi. La donnée est donc simplement placée dans un buffer par fputc, et envoyée sur la liaison série par la fonction sendchar :

```
/*
 * Fonction permettant de placer une donnée dans le buffer d'envoi
 *
 * @param :
 *    int ch : le code ASCII du caractère à envoyer sur la liaison
 *    FILE *f : non utilisé
 */
int fputc(int ch, FILE *f) {

	// on place la donnée dans le buffer
	bufferPut[posBufferPut++] = ch;

	// la fonction sendchar va lire le buffer et envoyé la donnée
	//on passe quand même un paramètre a la fonction , mais celui-ci
	//n'est pas prit en compte
	sendchar(0);

	return 0;
}
```

```
/*
 * Fonction permettant de placer une donnée sur la liaison série
 *
 * @param :
 *    int ch : non utilisé
 *
 */
int sendchar (int ch) {
      /* Write character to Serial Port */

      while (!(U2LSR & 0x20));

      //insérer notre donnée dans le buffer
      //attendre tant que buffer plein
      //générer une interruption

      //PosReadBufferPut permet de connaitre la place de la dernière donnée
à envoyer
      //qui peut etre différente de la dernière donnée mis en buffer (dans
le
      //cas où on utilise les interruptions)
      return (U2THR = bufferPut[posReadBufferPut++]);
}
```

Dans le cas de la réception de données sur le port série, l'interruption est générée à la lecture du registre U2RBR (registre contenant la dernière donnée reçus sur la ligne). Ici la fonction fgetc va appeler getkey, qui va lire notre donnée arrivant et la placer dans le buffer de réception via la fonction d'interruption. Fgetc va ensuite retourner la dernière donnée non lu présente dans le buffer.

Le problème étant qu'ici aussi l'interruption se déclenche uniquement quand on essai de lire la donnée présente sur le port série. Il aurait fallu être capable de la faire intervenir à chaque fois qu'une nouvelle donnée était disponible sur le port série, de manière à sauvegarder ces données dans le buffer à chaque fois qu'elle se présentait. A ce moment là, fgetc aurait simplement eu besoin de lire notre buffer. Nous n'avons cependant pas réussi à trouver dans la documentation la manière de gérer les interruptions de cette façon.

```
/*
 * Fonction permettant de lire une donnée sur la liaison série
 *
 * @param :
 *    FILE *f : non utilisé
 */
int fgetc(FILE *f)  {

      // lit la donnée sur le port série et écrit celle ci dans le buffer
      // de reception
      getkey();

      //retourne la dernière donnée a lire présente dans le buffer
      return bufferRec[posReadBufferRec++];
  }
```

```
/* Read character from Serial Port */
int getkey (void) {

      //Tant qu'il n'y a pas de donnée, on attends
      while (!(U2LSR & 0x01));

      //pour utilisation des interruptions
      //char recu sera placé dans le buffer par la routine d'interruption
      charRecu = U2RBR;
      return 0;
}
```

## II - Mise en place des interruptions

On utilise ici UART2, on active donc les interruptions comme suit :

```
portENTER_CRITICAL()
  {
      /* Setup the VIC for the UART. */
      VICIntSelect &= ~(1<<28 );
      VICIntEnable |= (1<<28);
      VICVectAddr28 = ( unsigned long ) vUART_ISREntry;
      VICVectPriority28 = 1;
      /* Enable UART2 interrupts. */
      U2IER |= 3;

  }
```

On donne notre fonction d'interruption au Vic comme d'habitude.
Enfin voila notre fonction d'interruption, qui ne sert ici qu'à la réception des données par la liaison série :

```
void vUART_ISRHandler( void )
{

      if(   U2LSR & 0x01 )
      {//interruption a cause d'une réception (activée a la lecture de la
donnée)

            //on place notre donnée reçue dans le buffer
            bufferRec[posBufferRec] = charRecu;
            posBufferRec++;
      }

      if(U2LSR & 0x20){
      //interruption a cause d'un envoie (activée a l'écriture de la donnée
sur le registre U2THR)

      }
      /* Clear the ISR in the VIC. */
      VICVectAddr = 0;
}
```