TP5 : Pilotage des LEDs à partir d'un port série RS232 en assembleur et en C

Objectif du TP

Réaliser un programme en langage C pour piloter 4 LEDs en utilisant l'interface série RS232.

Réalisation

Pour ce faire, il faut réaliser les programmes suivants :

- Réaliser une procédure en assembleur permettant de récupérer un caractère ASCII issu du port série RS232 : **unsigned char ReceptionCar(void)**.

- Réaliser une procédure en assembleur permettant d'envoyer un caractère ASCII vers le port série RS232 : **void EnvoiCar(unsigned char car)**.

- Réaliser un programme en C pour récupérer une chaîne de caractères sur l'entrée port série RS232 et la renvoyer sur la sortie, en utilisant les deux procédures précédentes. Au passage, les 4 bits de poids faible du code ASCII de chaque caractère reçu sont visualisés sur les LEDs. Par exemple, utiliser '@ABC...MNO'.

Configuration du port série RS232

Le port série RS232 est configuré de la manière suivantes (voir programme principal main.c) :

- Mode normal et 8 bits par caractère sans parité,
- 1 bit stop,
- Vitesse de communication 115200 bits par seconde.

Test et validation du programme

1- Lancer le programme Terminal.exe.

2- Configurer le port série du PC conformément à celui du KIT de développement (Figure 1).

3- Cliquer sur « **connect** ».

4- Connecter le port série RS232 du PC au port série (USART0) du kit de développement.

Pour envoyer un caractère vers le port série, il faut positionner le curseur dans la fenêtre d'envoi et taper un caractère au clavier puis cliquer sur « **send** ».

NB : Il faut réutiliser les programmes en assembleur du TP4 : Allumer/éteindre les diodes.

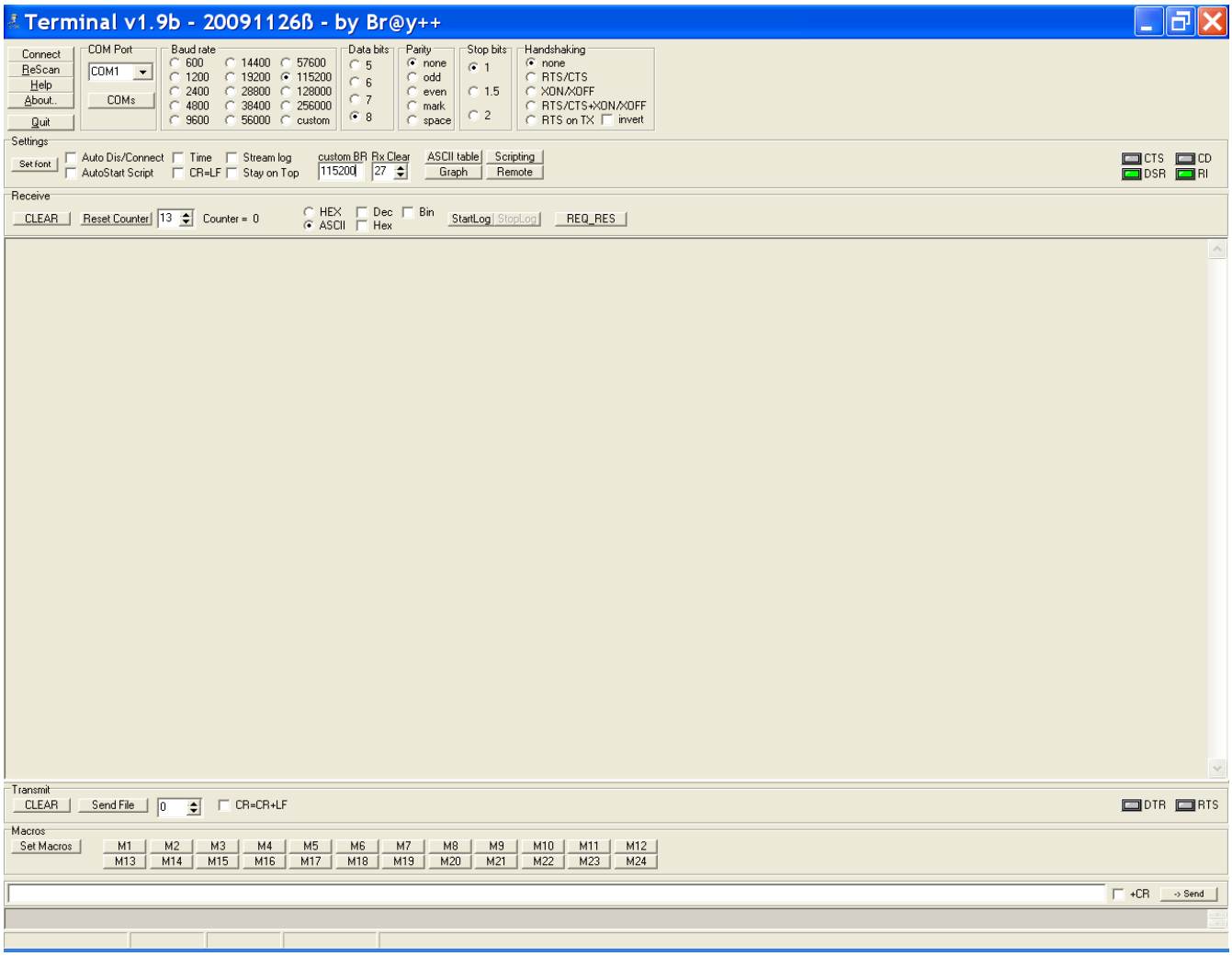

Figure 1 : Interface du programme Terminal.exe

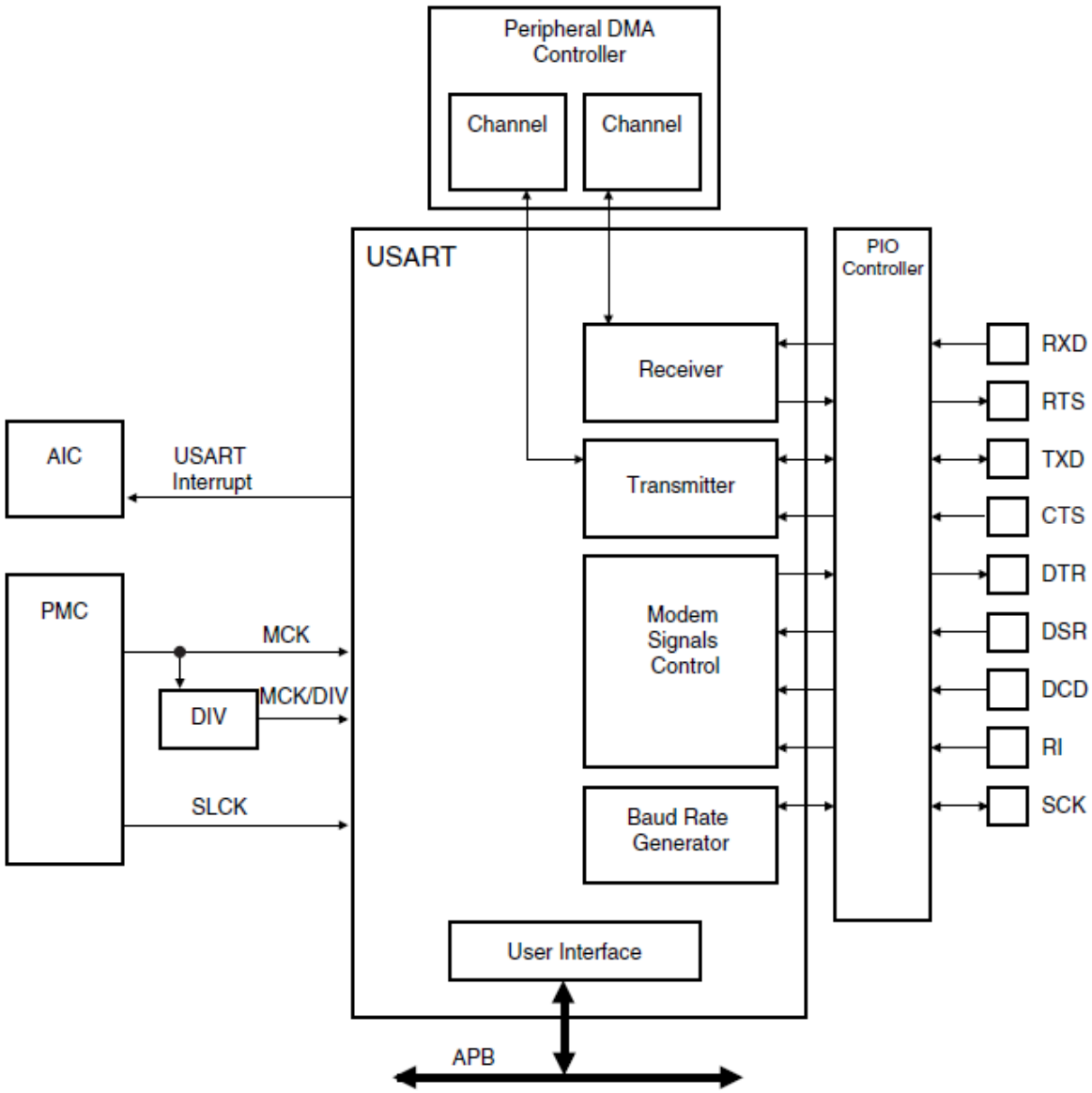

Figure 2 : Schéma synoptique du contrôleur RS232

31.7.6 USART Channel Status Register

Name: US_CSR

Access Type: Read-only

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
CTS	DCD	DSR	RI	CTSIC	DCDIC (1)	DSRIC (1)	RIIC (1)
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
–	–	NACK	RXBUFF	TXBUFE	ITERATION	TXEMPTY	TIMEOUT
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

Figure 3 : Le registre d'état de l'USART

Table 31-12. Memory Map

Offset	Register	Name	Access	Reset
0x0000	Control Register	US_CR	Write-only	–
0x0004	Mode Register	US_MR	Read-write	–
0x0008	Interrupt Enable Register	US_IER	Write-only	–
0x000C	Interrupt Disable Register	US_IDR	Write-only	–
0x0010	Interrupt Mask Register	US_IMR	Read-only	0x0
0x0014	Channel Status Register	US_CSR	Read-only	–
0x0018	Receiver Holding Register	US_RHR	Read-only	0x0
0x001C	Transmitter Holding Register	US_THR	Write-only	–
0x0020	Baud Rate Generator Register	US_BRGR	Read-write	0x0
0x0024	Receiver Time-out Register	US_RTOR	Read-write	0x0
0x0028	Transmitter Timeguard Register	US_TTGR	Read-write	0x0
0x2C - 0x3C	Reserved	–	–	–
0x0040	FI DI Ratio Register	US_FIDI	Read-write	0x174
0x0044	Number of Errors Register	US_NER	Read-only	–
0x0048	Reserved	–	–	–
0x004C	IrDA Filter Register	US_IF	Read-write	0x0
0x5C - 0xFC	Reserved	–	–	–
0x100 - 0x128	Reserved for PDC Registers	–	–	–

Figure 4 : Les registres internes de l'USART0

Le fichier main.c à compléter.

```
/* Modified by KM HOU ISIMA 2011*/
#include <board.h>
#include <pio/pio.h>
#include <usart/usart.h>
#include <pmc/pmc.h>
#include <utility/trace.h>
#include <stdio.h>
//------------------------------------------------------------------------------
//         Local definitions
//------------------------------------------------------------------------------
#ifndef AT91C_ID_TC0
#if defined(AT91C_ID_TC012)
    #define AT91C_ID_TC0 AT91C_ID_TC012
#elif defined(AT91C_ID_TC)
    #define AT91C_ID_TC0 AT91C_ID_TC
#else
    #error Pb define ID_TC
#endif
#endif
```

```
/// Maximum Bytes Per Second (BPS) rate that will be forced using CTS
#define MAX_BPS             500
//------------------------------------------------------------------------------
//         Local variables
//------------------------------------------------------------------------------
/// Pins to configure for the application.
const Pin pins[] = {
    PINS_DBGU,
    PIN_USART0_RXD,    PIN_USART0_TXD,    PIN_USART0_CTS,    PIN_USART0_RTS
};
/// Number of bytes received between two timer ticks.
volatile unsigned int bytesReceived = 0;
/// Receive buffer.
unsigned char pBuffer[BUFFER_SIZE];
/// String buffer.
char pString[24];
/// USART0 in hardware handshaking mode, asynchronous, 8 bits, 1 stop bit,
/// no parity, 115200 bauds and enables its transmitter and receiver.
void ConfigureUsart0(void)
{    unsigned int mode = AT91C_US_USMODE_HWHSH   | AT91C_US_CLKS_CLOCK
                       | AT91C_US_CHRL_8_BITS   | AT91C_US_PAR_NONE
                       | AT91C_US_NBSTOP_1_BIT | AT91C_US_CHMODE_NORMAL;
    // Enable the peripheral clock in the PMC
    PMC_EnablePeripheral(AT91C_ID_US0);
    // Configure the USART in the desired mode @115200 bauds
    USART_Configure(AT91C_BASE_US0, mode, 115200, BOARD_MCK);
    // Enable receiver & transmitter
    USART_SetTransmitterEnabled(AT91C_BASE_US0, 1);
    USART_SetReceiverEnabled(AT91C_BASE_US0, 1);
}
int main(void)
{ unsigned char car='9';
// section code d'initialisation obligatoire
// Configure pins
      PIO_Configure(pins, PIO_LISTSIZE(pins));
// Configure USART0 and display startup trace
      ConfigureUsart0();
```

```
// Ecrire votre programme ici
}
/* fichier assembleur Auteur : KM HOU,    Date : 20/05/2011,      ISIMA
// ========== Register definition for US0 peripheral ==========
#define AT91C_US0_TTGR ((AT91_REG *) 0xFFFC0028) // Transmitter Time-guard
#define AT91C_US0_BRGR ((AT91_REG *) 0xFFFC0020) // Baud Rate Generator
#define AT91C_US0_RHR  ((AT91_REG *) 0xFFFC0018) // Receiver Holding
#define AT91C_US0_IMR  ((AT91_REG *) 0xFFFC0010) // Interrupt Mask
#define AT91C_US0_NER  ((AT91_REG *) 0xFFFC0044) // Nb Errors
#define AT91C_US0_RTOR ((AT91_REG *) 0xFFFC0024) // Receiver Time-out
#define AT91C_US0_FIDI ((AT91_REG *) 0xFFFC0040) // FI_DI_Ratio
#define AT91C_US0_CR   ((AT91_REG *) 0xFFFC0000) // Control
#define AT91C_US0_IER  ((AT91_REG *) 0xFFFC0008) // Interrupt Enable
#define AT91C_US0_IF   ((AT91_REG *) 0xFFFC004C) // IRDA_FILTER
#define AT91C_US0_MR   ((AT91_REG *) 0xFFFC0004) // Mode
#define AT91C_US0_IDR  ((AT91_REG *) 0xFFFC000C) // Interrupt Disable
#define AT91C_US0_CSR  ((AT91_REG *) 0xFFFC0014) // Channel Status
#define AT91C_US0_THR  ((AT91_REG *) 0xFFFC001C) // Transmitter Holding
*/
```

Définitions des paramètres et exemples

```
Adresse_base_Uart0      EQU 0xFFFC0000 ;
Offset_RegStatus        EQU 0x14 ; 20
Offset_RegRxD           EQU 0x18 ; 24
Offset_RegTxD           EQU 0x1C ; 28
TxRDY                   EQU 0x2
RxRDY                   EQU 0x1
```

Exemple en assembleur pour tester si le registre de transmission est disponible :

```
; R1 contient le l'adresse de base de l'USART0
      LDR R2,[R1,#0x14] ; lecture du registre d'état
      ANDS R2,R2,#02 ; test si Tx est disponible
```

Exemple en assembleur pour tester si le registre de réception contient un caractère reçu :

```
; R1 contient le l'adresse de base de l'USART0
      LDR R2,[R1,#0x14] ; lecture du registre d'état
      ANDS R2,R2,#01 ; test si Rx est disponible
```

NB : Ces tests sont indispensables avant l'envoi et la lecture d'un caractère reçu.