Laboratoire

S4E APP3

Créé par : Rémi Pelletier
Mis à jour par : D Gaucher, P Gendron, J-B Michaud et J-P Gouin
Version E2022 : J-P Gouin

Exercice 1a

À partir du code de départ dans le fichier main.c, compléter la fonction fct_C() qui doit avoir les spécifications suivantes :

Entrée
Le tableau de l'état des interrupteurs Le numéro de la DEL présentement allumée
Fonction
Une boucle qui compte le nombre d'interrupteur à « on ». Vérification si on doit faire un changement ou non de la DEL à allumer.
Sortie
Le numéro de la DEL à allumer La valeur 10 s'il s'agit de la bonne DEL déjà allumée La valeur -1 s'il y a un erreur.

```c
#include <xc.h>
#include "config.h"

int fct_C(unsigned char *valueSW);
extern int fct_S(int *valueSW);
/*La bonne pratique est d'aouter un _t à la fin d'un type non standard
 pour la lisibilite du programme. C'est ce que fait stdint.h*/

void main() {
    LED_Init();
    SWT_Init();
    unsigned char valueSW [8];
    unsigned char valueLED [8];
    unsigned char noLED;
    int no;

    // Boucle principale
    while(1) {
        // boucle pour lire SW
        for(no=0; no<8; no++)  {
            valueSW[no] = SWT_GetValue(no); //Lecture de la SW
            valueLED[no] = 0; //Remet à 0 les LED
        }
        //(numéro 1a)
        noLED = fct_C(valueSW);   // Appelle de la fonction en C

        //(numéro 1b)
        //noLED = fct_S((int) *valueSW);     // Appelle de la fonction en assembleur

        valueLED[noLED] = 1; // modifie le tableau de SW
        for(no=0; no<8; no++)  {
            LED_SetValue(no,valueLED[no]);   //Écriture de la SW
        }
    }
}

int fct_C(unsigned char *valueSW){
    int noLED;

    noLED = 2;      //met à 2 le numéro de la LED à allumer.

    return noLED;
}
```

Exercice 1b

Modifier votre code en complétant le code en assembleur dans le fichier fct_S.s remplacera la fonction fct_C() et qui sera appelée par le par le code en C.

Attention : Vous devez prévoir l'utilisation vos registres dans la fonction en assembleur et utiliser les bons registres d'entrée et de sortie de la fonction (par référence ou par valeur).

Entrée
Le tableau de l'état des interrupteurs Le nombre de DEL Le numéro de la DEL présentement allumée
Fonction
Validation du nombre de DEL et d'interrupteurs. Une boucle qui compte le nombre d'interrupteur à « on ». Vérification si on doit faire un changement ou non de la DEL à allumer.
Sortie
Le numéro de la DEL à allumer La valeur 10 s'il s'agit de la bonne DEL déjà allumée La valeur -1 s'il y a un erreur.

Exercice 1b

Registers		
0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

Notions à ne pas oublier

Fonctions : Passage de paramètres par référence ou par valeur?

Mots clés en C :

- extern (variables globales -> projet multifichiers)
- Volatile (variable globale conservée pendant une interruption)

Convention des registres $v0-$v1 et $a0-$a3

Stack et sauvegarde de $s0-$s7

- La fonction appelée doit conserver l'état des registres $s0-$s7 : protection du contexte. Le contenu des registres à préserver est conservé sur la pile (stack), puis restauré avant de retourner à l'appelant.
- ISR doit tout sauvegarder!

Exercice 2

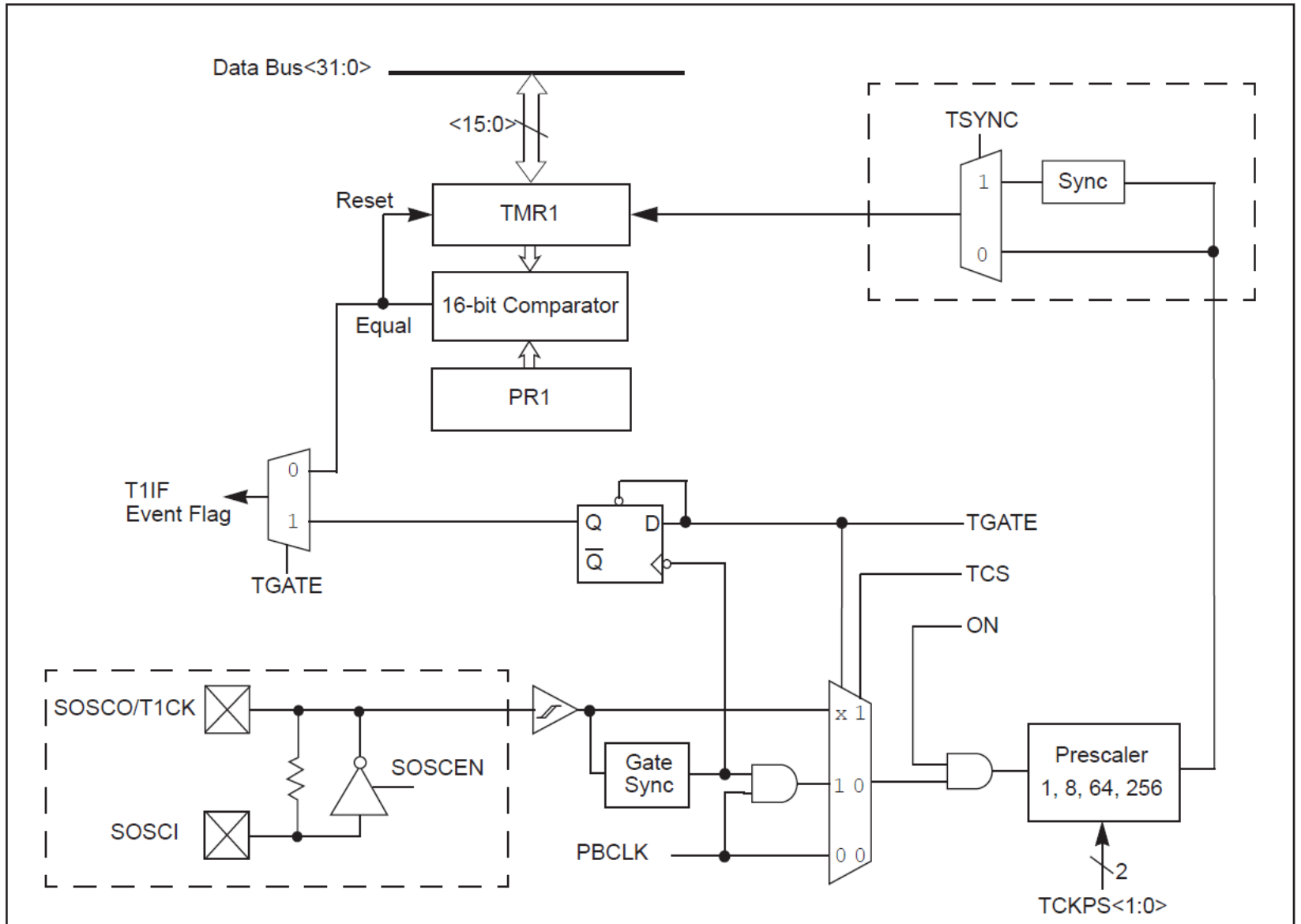
Modifier un code de départ pour arriver à une interruption au millisecondes et y inclure un compteur. Réviser le fonctionnement d'un timer.

a) Ajouter la fonction LED_ToogleValue() pour changer l'état de la DEL 0 précisément à chaque seconde.

b) Ajouter une fonction en C, puis en assembleur pour générer une onde carrée de 1kHz sur le PMOD B patte 1, puis valider la fréquence avec l'Analog Discovery.
Conseil : Utiliser les fonctions de pmod.c

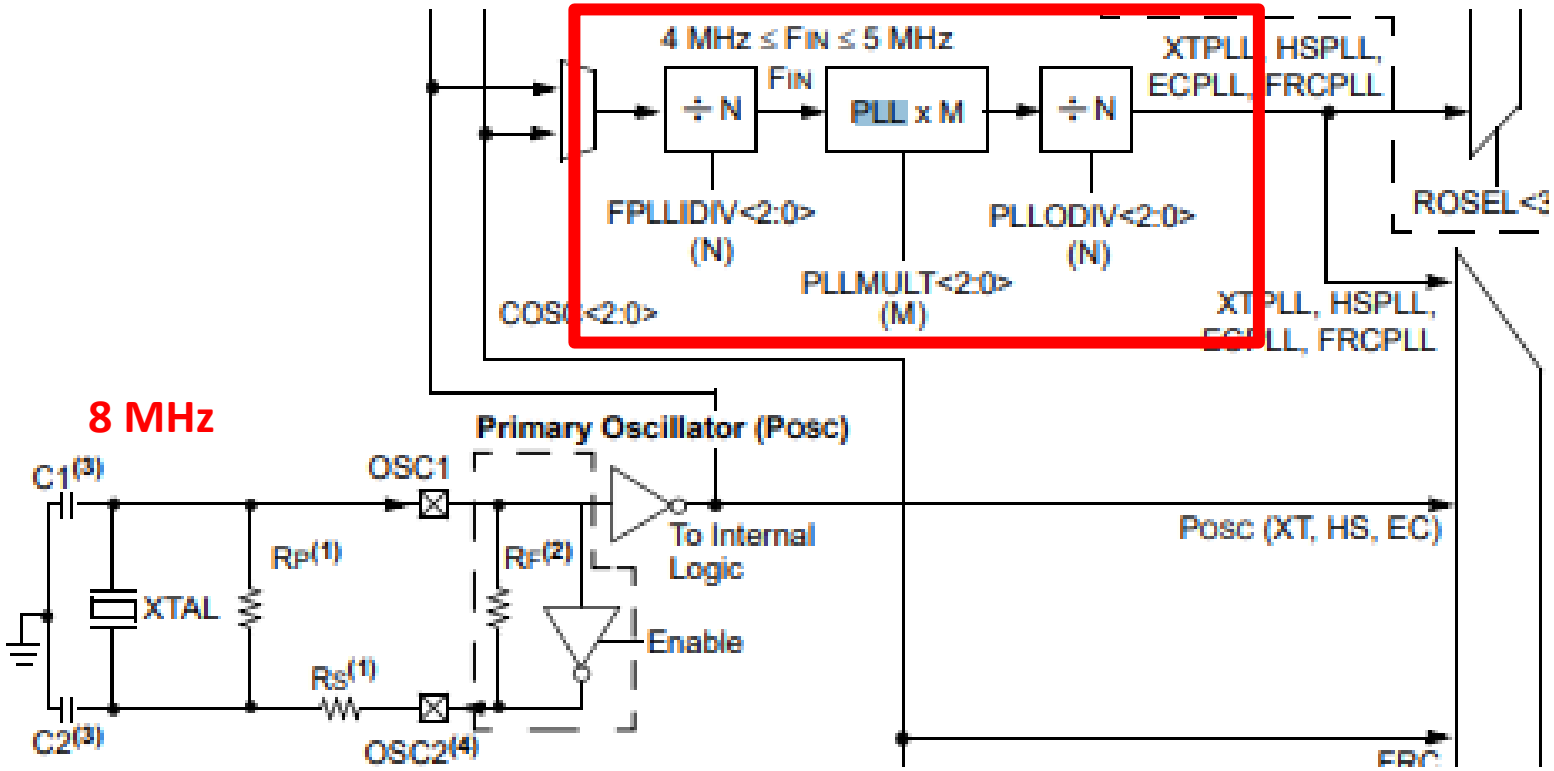
c) En utilisant les fonctions de la librairie adc.h, modifier le code en C pour que la période en milliseconde de l'onde carrée soit égale à la valeur du potentiomètre, puis valider la fréquence avec l'Analog Discovery.

FIGURE 13-1: TIMER1 BLOCK DIAGRAM



Note: The default state of the SOSCEN (OSCCON<1>) bit during a device Reset is controlled by the FSOSCEN bit in Configuration Word, DEVCFG1.

Horloges


4 MHz ≤ FIN ≤ 5 MHz
÷ N
FIN
PLL x M
÷ N
XTPLL, HSPLL,
ECPLL, FRCPLL
FPLLIDIV<2:0>
(N)
PLLMULT<2:0>
(M)
PLLODIV<2:0>
(N)
ROSEL<3
XTPLL, HSPLL,
ECPLL, FRCPLL
8 MHz
Primary Oscillator (Posc)
C1(3)
OSC1
To Internal
Logic
Posc (XT, HS, EC)
RP(1)
RF(2)
XTAL
Enable
RS(1)
C2(3)
OSC2(4)

Où est la configuration?

Configuré dans le fichier ‘config_bits.c’

Pour l’APP, il est recommandé de laisser la configuration par défaut

```
#pragma config FSRSSEL = PRIORITY_7 // Shadow Register Set Priority Select
//(SRS Priority 7)
#pragma config PMDL1WAY = OFF // Peripheral Module Disable
// Configuration (Allow multiple
// reconfigurations)
#pragma config IOL1WAY = OFF // Peripheral Pin Select Configuration
// (Allow multiple reconfigurations)
// DEVCFG2
#pragma config FPLLIDIV = DIV_2 // PLL Input Divider (2x Divider)
#pragma config FPLLMUL = MUL_24 // PLL Multiplier (20x Multiplier)
#pragma config FPLLODIV = DIV_1 // System PLL Output Clock Divider
// (PLL Divide by 1)
// DEVCFG1
#pragma config FNOSC = PRIPLL // Oscillator Selection Bits (Primary
// Osc w/PLL (XT+,HS+,EC+PLL))
#pragma config FSOSCEN = OFF // Secondary Oscillator Enable
// (Disabled)
#pragma config IESO = OFF // Internal/External Switch Over
// (Disabled)
```

Configuration du timer1

```
void initialize_300us_interrupt(void)  {
  T1CONbits.TCKPS = 3;              //    1:256 prescaler value
  T1CONbits.TGATE = 0;              //    not gated input (the default)
  T1CONbits.TCS = 0;               //    PCBLK input (the default)
  PR1 = (int)(((float)(TMR_TIME * PB_FRQ) / 256) + 0.5);    //set period register, generates one interrupt every ?

  TMR1 = 0;                       //    initialize count to 0
  IPC1bits.T1IP = 2;               //    INT step 4: priority
  IPC1bits.T1IS = 0;               //    subpriority
  IFS0bits.T1IF = 0;               //    clear interrupt flag
  IEC0bits.T1IE = 1;               //    enable interrupt
  T1CONbits.ON = 1;                //    turn on Timer1
}
```

Exercice 2 a)

Modifier un code de départ pour arriver à une interruption au millisecondes et y inclure un compteur. Réviser le fonctionnement d'un timer.

a) Ajouter la fonction LED_ToogleValue() pour changer l'état de la DEL 0 précisément à chaque seconde.

b) Ajouter une fonction en C, puis en assembleur pour générer une onde carrée de 1kHz sur le PMOD B patte 1, puis valider la fréquence avec l'Analog Discovery.
Conseil : Utiliser les fonctions de pmod.c

c) En utilisant les fonctions de la librairie adc.h, modifier le code en C pour que la période en milliseconde de l'onde carrée soit égale à la valeur du potentiomètre, puis valider la fréquence avec l'Analog Discovery.

```
void __ISR(_TIMER_1_VECTOR, IPL2AUTO) Timer1ISR(void)
{
  Flag_1s = 1;          //indique à la boucle principale qu'on doit traiter
  IFS0bits.T1IF = 0;    // clear interrupt flag
}


Main()
{
…
 while(1) {
       if(Flag_1s) {
          Flag_1s = 0;       // Reset the flag to capture the next event
          if (++count >= 1000) {
             count = 0;
         LED_ToggleValue(0);
          LCD_seconde(++seconde);
       }
      }
}
```

Exercice 2 b)

Modifier un code de départ pour arriver à une interruption au millisecondes et y inclure un compteur. Réviser le fonctionnement d'un timer.

a) Ajouter la fonction LED_ToogleValue() pour changer l'état de la DEL 0 précisément à chaque seconde.

b) Ajouter une fonction en C, puis en assembleur pour générer une onde carrée de 1kHz sur le PMOD B patte 1, puis valider la fréquence avec l'Analog Discovery.
Conseil : Utiliser les fonctions de pmod.c

c) En utilisant les fonctions de la librairie adc.h, modifier le code en C pour que la période en milliseconde de l'onde carrée soit égale à la valeur du potentiomètre, puis valider la fréquence avec l'Analog Discovery.

Résumé des I/O

There is seven I/O port named A–G and each containing 16 bits.

Each I/O Port has the following control registers:

- TRISx, LATx, PORTx, ANSELx, CNPUx, CNPDx and ODCx.

Setting a TRIS bit to 0 makes the corresponding pin an output or 1 makes the pin an input.

The LAT register is used to write to the I/O Port.

- Writing to the LAT register sets any pins configured as outputs. Reading from the LAT register returns the last value written.

The PORT register is used to read from the I/O Port. Reading from the PORT register returns the current state of all the pins in the I/O Port.

- Writing to the PORT register may not produce the expected result, therefore writing to LAT register is recommended.

To summarize: write using LAT, read using PORT.

Exercice 2 b)

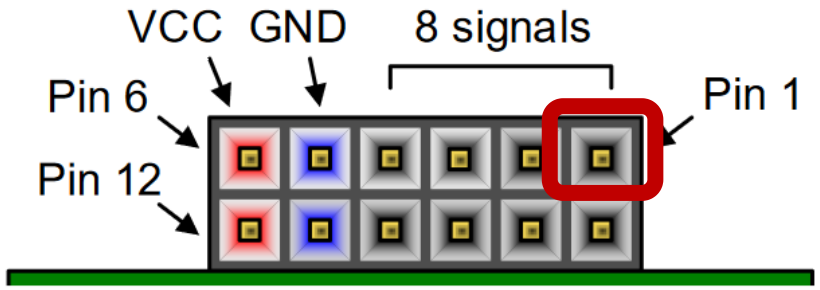
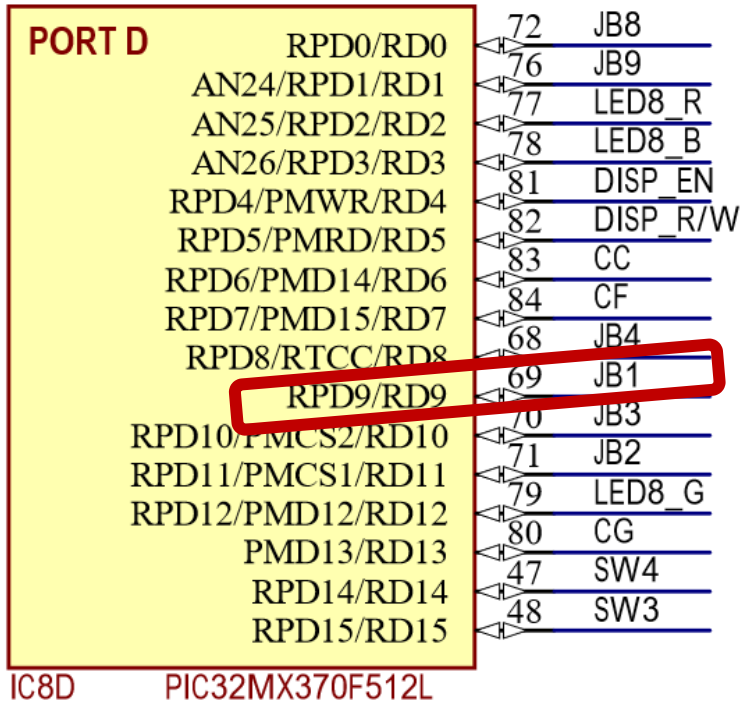


Figure 20.2. Pmod connectors: front view, as loaded on PCB.



Pmod pin	Schematic Label	PIC32 pin
PMODB_1	JB1	RPD9/RD9
PMODB_2	JB2	RPD11/PMCS1/RD11
PMODB_3	JB3	RPD10/PMCS2/RD10
PMODB_4	JB4	RPD8/RTCC/RD8
PMODB_7	JB7	SOSCO/RPC14/T1CK/RC14
PMODB_8	JB8	RPD0/RD0
PMODB_9	JB9	AN24/RPD1/RD1
PMODB_10	JB10	SOSCI/RPC13/RC13

```
/**------------------------------------------------------------ */
/** PMODS_InitPin
**
**  Parameters:
**      unsigned char bPmod      - the PMOD where the pin is located
**                                 0 - PMODA
**                                 1 - PMODB
**      unsigned char bPos       - the pin position in the Pmod (allowed values 1-4, 7-10)
**                                 1  - JA1 (if bPmod = 0), JB1 (if bPmod = 1)
**                                 2  - JA2 (if bPmod = 0), JB2 (if bPmod = 1)
**                                 3  - JA3 (if bPmod = 0), JB3 (if bPmod = 1)
**                                 4  - JA4 (if bPmod = 0), JB4 (if bPmod = 1)
**                                 7  - JA7 (if bPmod = 0), JB7 (if bPmod = 1)
**                                 8  - JA8 (if bPmod = 0), JB8 (if bPmod = 1)
**                                 9  - JA9 (if bPmod = 0), JB9 (if bPmod = 1)
**                                10  - JA10(if bPmod = 0), JB10(if bPmod = 1)
**      unsigned char bDir       - the pin direction
**                                 0  - Output
**                                 1  - Input
**      unsigned char pull-up    - the pull-up property of the pin
**                                 0  - No pull-up
**                                 1  - Pull-up
**      unsigned char pull-down  - the pull-down property of the pin
**                                 0  - No pull-down
**                                 1  - Pull-down
**
**  Return Value:
**
**
**  Description:
**      This function configures the pins located in the PMODA and PMODB connectors
**      to be used as digital input / output, also allowing pullup and pulldown properties to be specified.
**      This function uses pin related definitions from config.h file.
**      If the bPmod and bPos do not specify a valid pin, nothing happens.
*/
```

Exercice 2 b)

```
Main()
{
PMODS_InitPin(1,1,0,0,0); // initialisation du JB1 (RD9))
unsigned char pmodValue = 0;
…
 while(1) {
       if(Flag_1s) {
          Flag_1s = 0;       // Reset the flag to capture the next event
                 // LATDbits.LATD9 ^= 1;      // ^ = XOR sur la patte D9 (JB1)
                  pmodValue = PMODS_GetValue(1, 1);              // Lire D9 (JB1)
                 pmodValue ^= 1;                                 // XOR
                  PMODS_SetValue(1, 1, pmodValue);               // Écrire D9 (JB1)
                  if (++count >= 1000) {
                count = 0;
                    LED_ToggleValue(0);
                    LCD_seconde(++seconde);
                  }
      }
}
```

Exercice 2 b)

```
Main()
{
PMODS_InitPin(1,1,0,0,0); // initialisation du JB1 (RD9))
…
 while(1) {
       if(Flag_1s) {
          Flag_1s = 0;       // Reset the flag to capture the next event
             pmod_s();             // Appelle de la function en assembleur
           if (++count >= 1000) {
             count = 0;
               LED_ToggleValue(0);
               LCD_seconde(++seconde);
             }
      }
}
```

Exercice 2 b)

Code en Assembleur à ajouter :

```
li $t0, 0x0200          # Création du masque sur la pin RD9
                        # 0x0200 = 0b 0000 0010 0000 0000
lw $t1, PORTD($0)       # Lecture le port D
xor $t2, $t1, $t0       # Masquage
sw $t2, LATD($0)        # Écriture le port D
```

Exercice 2 c)

Modifier un code de départ pour arriver à une interruption au millisecondes et y inclure un compteur. Réviser le fonctionnement d'un timer.

a) Ajouter la fonction LED_ToogleValue() pour changer l'état de la DEL 0 précisément à chaque seconde.

b) Ajouter une fonction en C, puis en assembleur pour générer une onde carrée de 1kHz sur le PMOD B patte 1, puis valider la fréquence avec l'Analog Discovery.
Conseil : Utiliser les fonctions de pmod.c

c) En utilisant les fonctions de la librairie adc.h, modifier le code en C pour que la période en milliseconde de l'onde carrée soit égale à la valeur du potentiomètre, puis valider la fréquence avec l'Analog Discovery.

Exercice 3

a) Réviser le fonctionnement d'un UART.

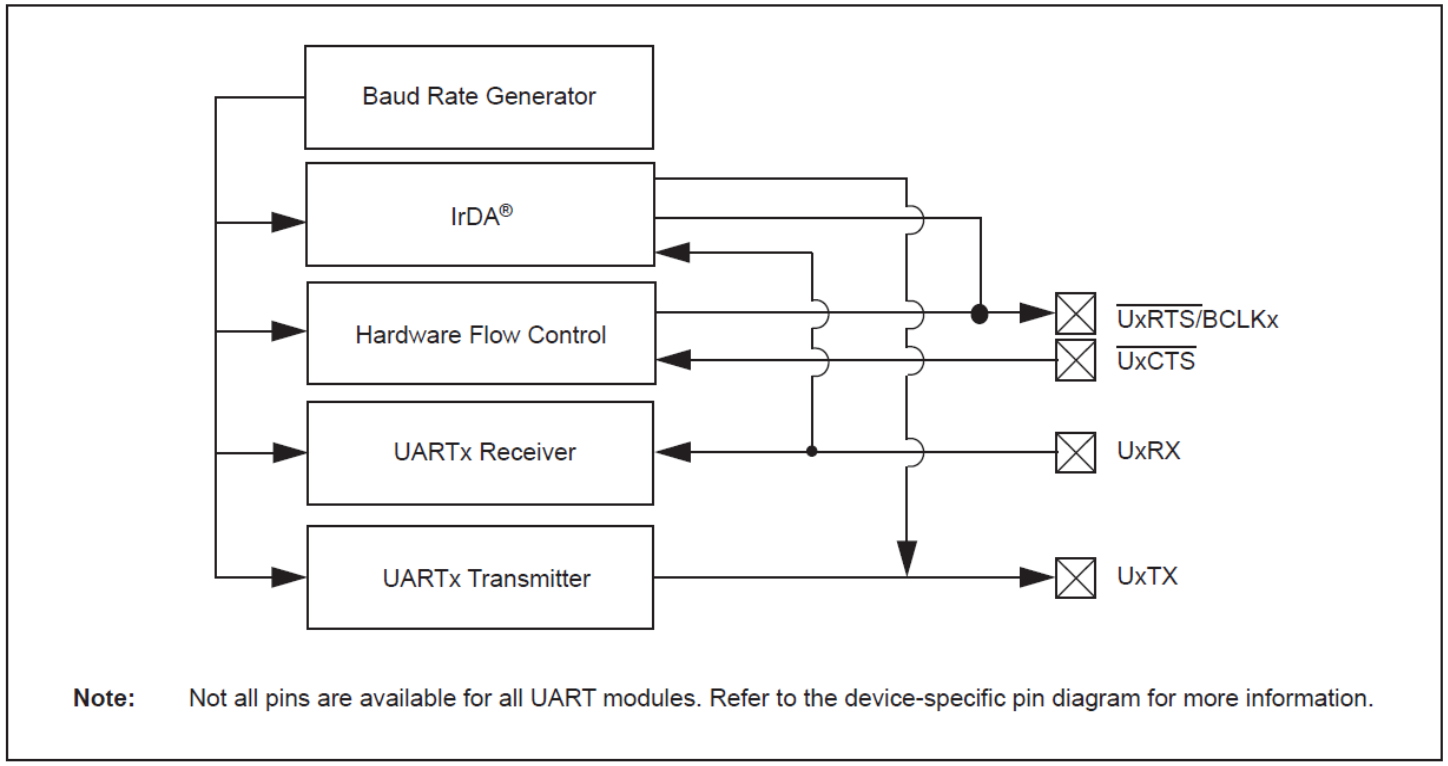
b) Extraire le diagramme de séquence de l'exemple UART qui utilise la librairie de Digilent.

c) Est-ce que la fréquence de PBCLK choisie à l'exercice 4 convient au baud rate demandé? Quelle serait la valeur du registre BRG dans ce cas ?

d) Modifier le code pour reconnaître une séquence prédéfinie au début d'une ligne de texte reçue par l'application.

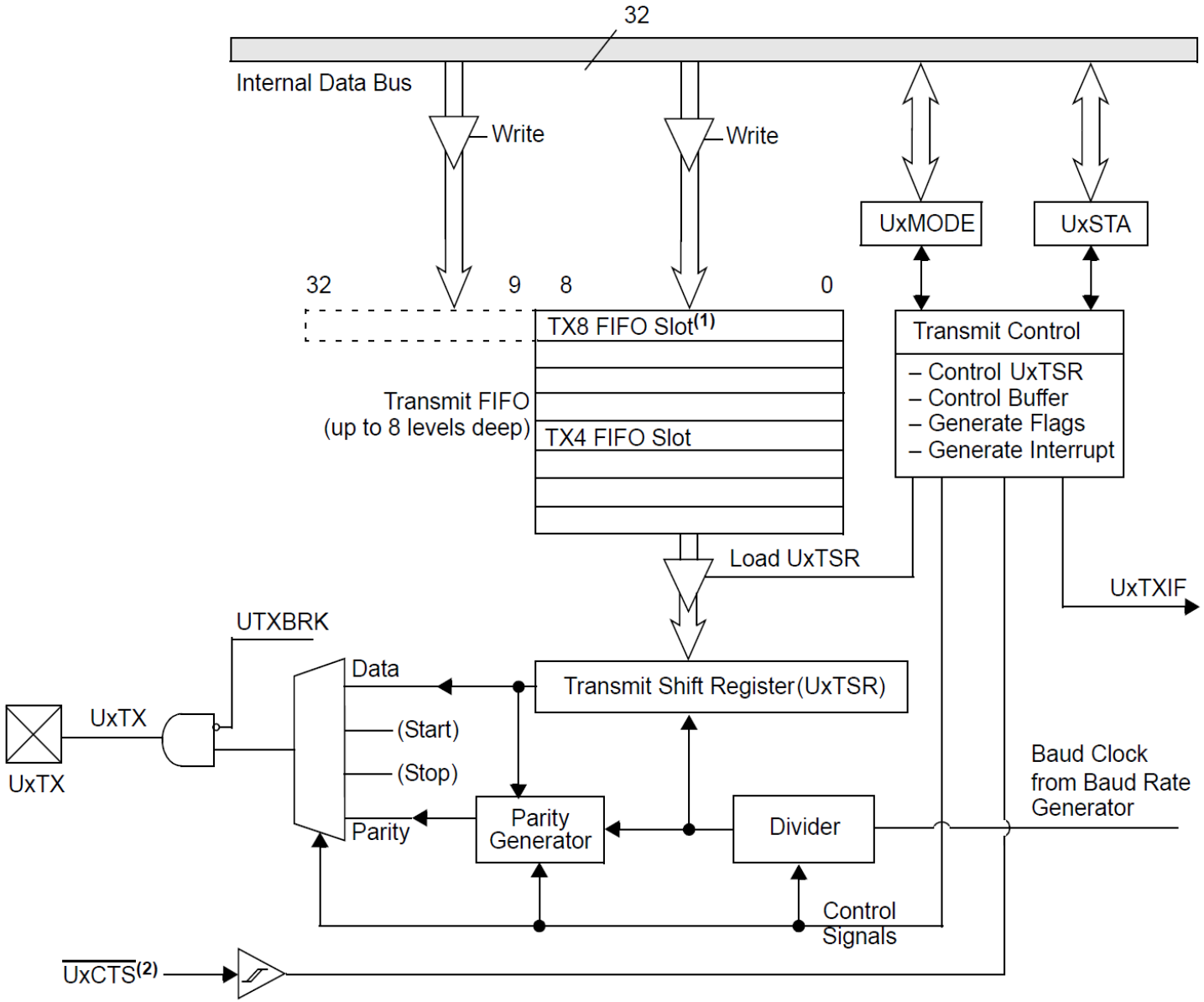
Exercice 3

FIGURE 20-1: UART SIMPLIFIED BLOCK DIAGRAM



Note: Not all pins are available for all UART modules. Refer to the device-specific pin diagram for more information.

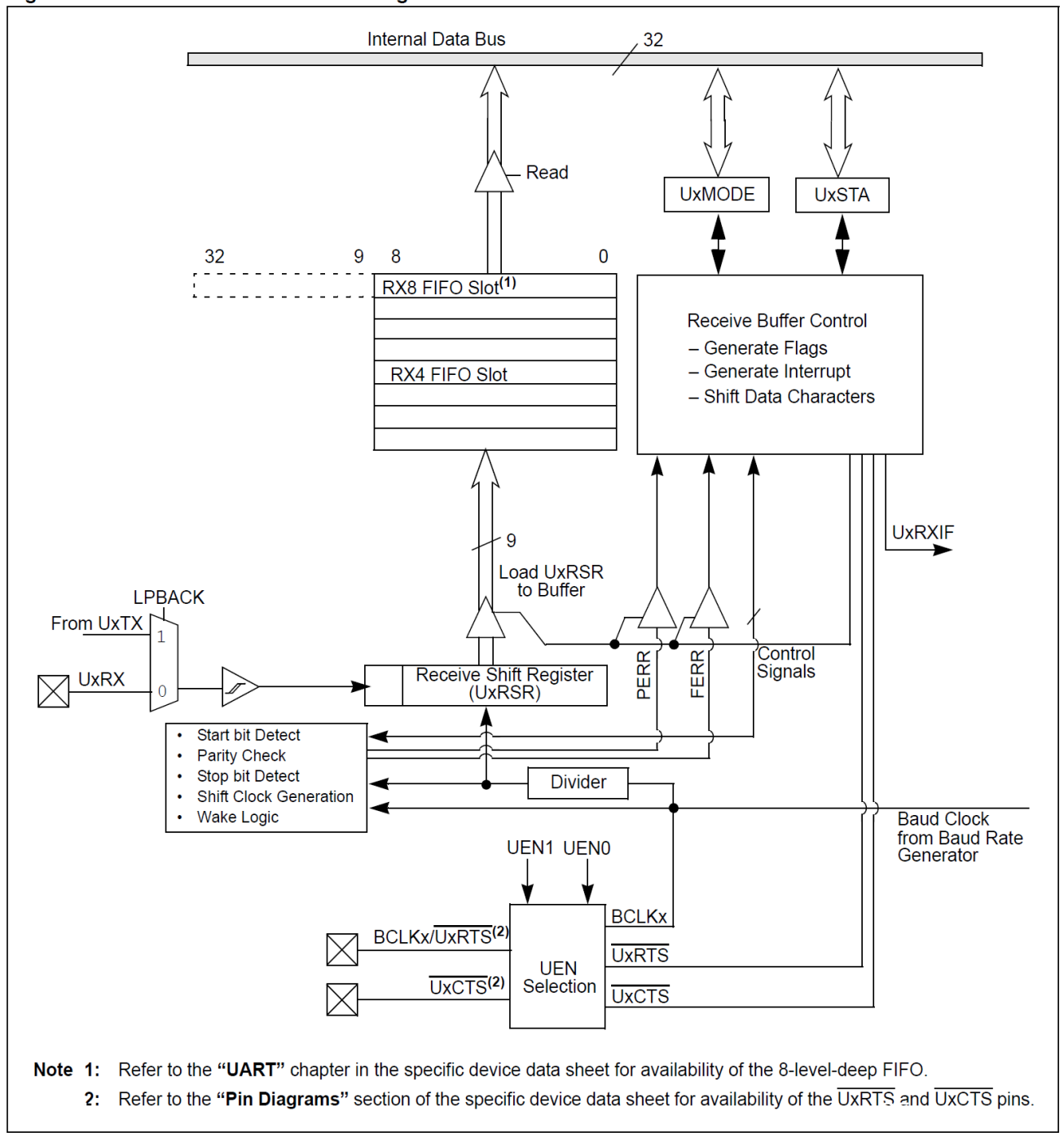
Figure 21-3: UART Transmitter Block Diagram(1)



Note 1: Refer to the **"UART"** chapter in the specific device data sheet for availability of the 8-level-deep FIFO.

2: Refer to the **"Pin Diagrams"** section in the specific device data sheet for availability of the $\overline{\text{UxCTS}}$ pin.

Figure 21-7: UART Receiver Block Diagram(1)



Note 1: Refer to the **"UART"** chapter in the specific device data sheet for availability of the 8-level-deep FIFO.

2: Refer to the **"Pin Diagrams"** section of the specific device data sheet for availability of the $\overline{\text{UxRTS}}$ and $\overline{\text{UxCTS}}$ pins.

Exercice 3

Equation 21-1: UART Baud Rate with BRGH = 0

$$Baud\ Rate = \frac{F_{PB}}{16 \cdot (UxBRG + 1)}$$

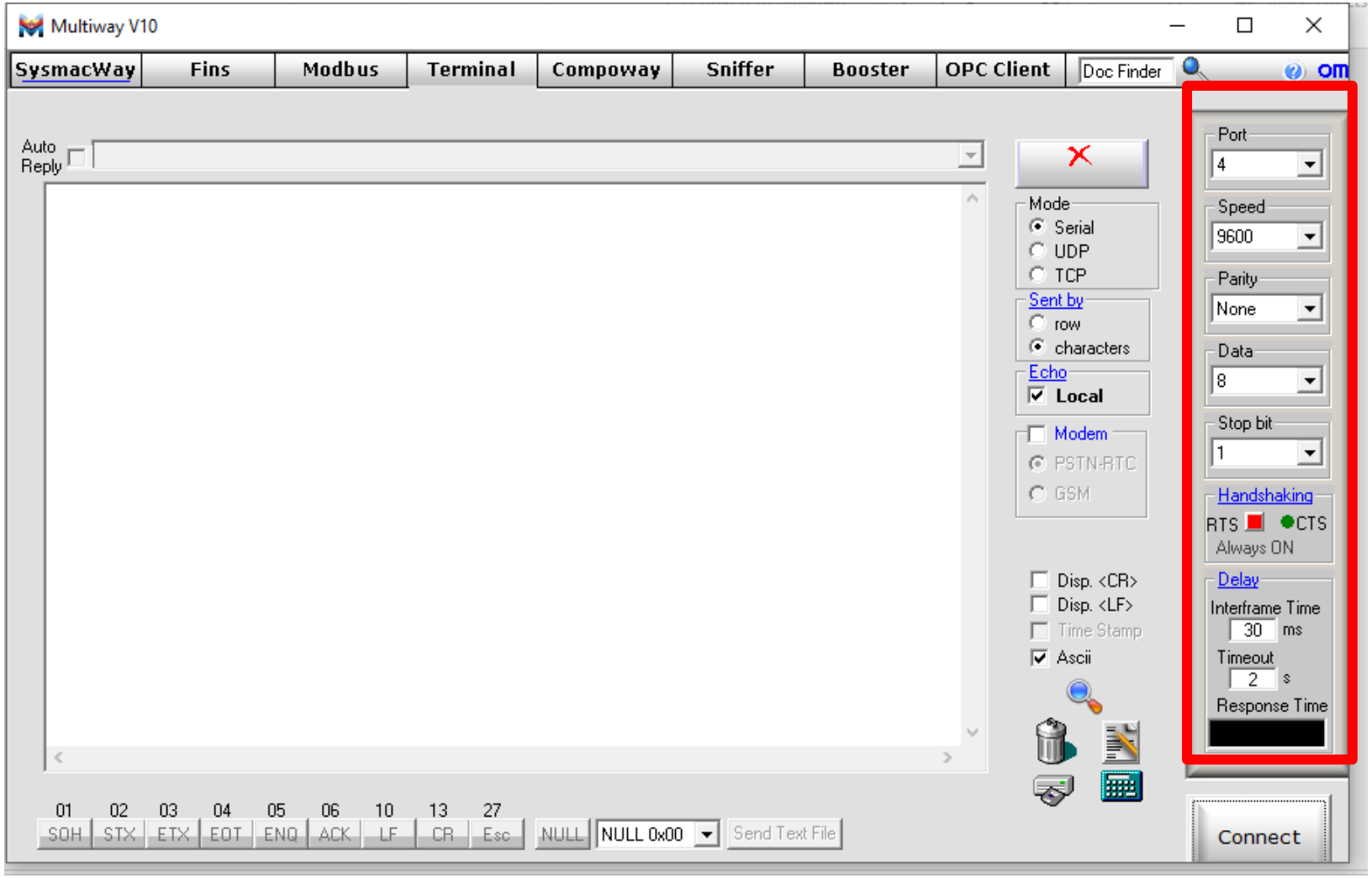
$$UxBRG = \frac{F_{PB}}{16 \cdot Baud\ Rate} - 1$$

Note: F_{PB} denotes the PBCLK frequency.

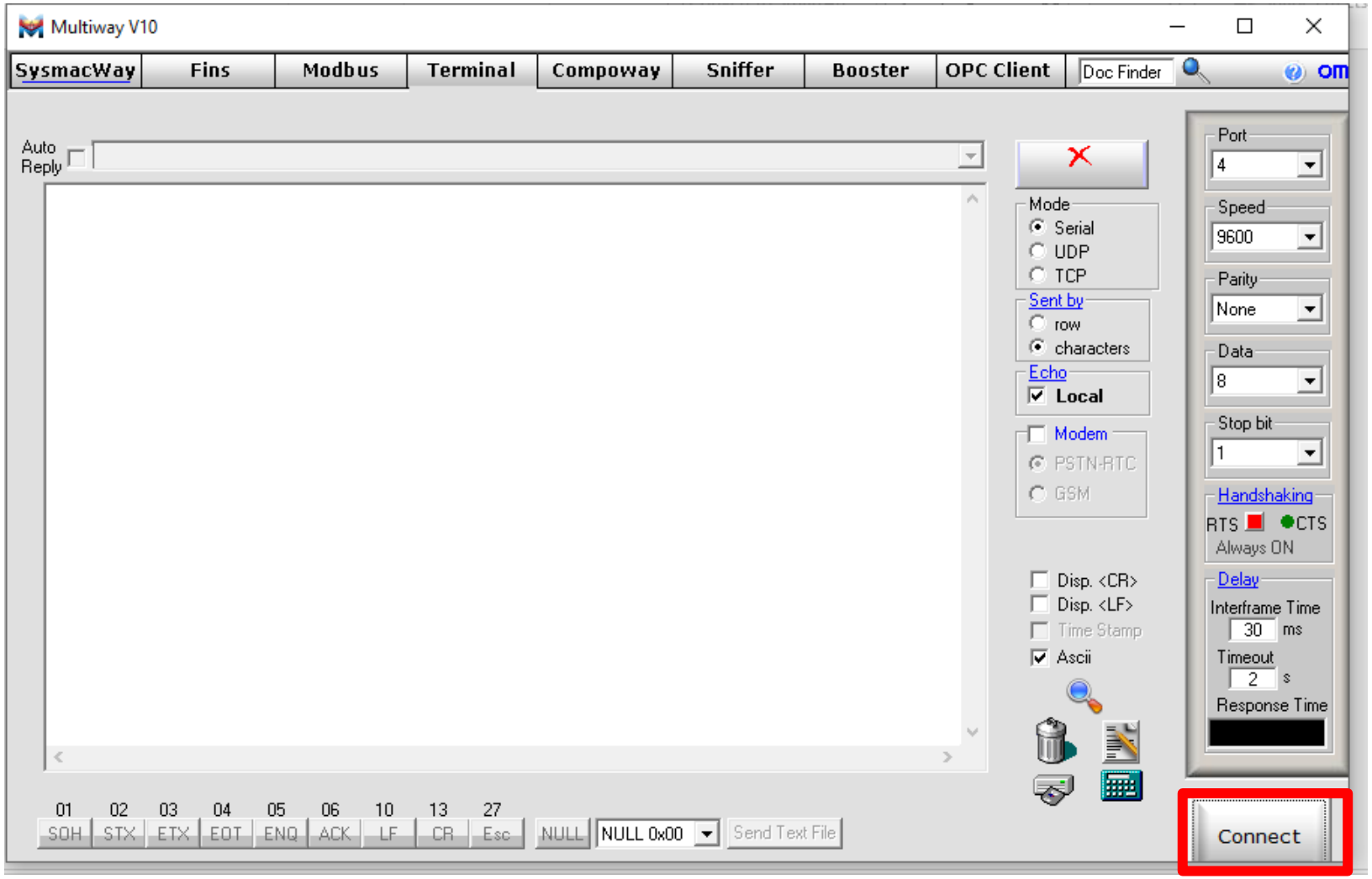
```c
#include <xc.h>
#include "config.h"
#include "uart.h"

#define BAUD_RATE 9600
#define RECEIVE_BUFFER_LEN  cchRxMax

void main() {
    LCD_Init();
    UART_Init(BAUD_RATE);   // Configure UART with interrupt

    macro_enable_interrupts();  // enable interrupts

    char receive_buffer[RECEIVE_BUFFER_LEN];

    while(1) {
        unsigned char len = UART_GetString(receive_buffer, RECEIVE_BUFFER_LEN);
        // Check if a string is available
        if (len > 0){
            LCD_DisplayClear();     // Clear what was previously written.
            DelayAprox10Us(1000);   // Wait a few us after clear before writing again.

            // Write the string on the first line starting at index 0
            LCD_WriteStringAtPos(receive_buffer, 0, 0);
            UART_PutString(receive_buffer);
            UART_PutChar('\n');
            UART_PutChar('\r');
        }
    }
}
```

```
unsigned char UART_GetString( char* pchBuff, int cchBuff )
{
    unsigned char ich;

    // Have we finished receiving a CR+LF terminated string via UART4?
    if(!fRxDone)
    {
        return 0;
    }
    // Does the user buffer have enough space to store the CR+LF terminated string?
    if(cchBuff < ichRx - 1)
    {
        // A buffer underrun occured.
        macro_disable_interrupts;
        fRxDone = 0;
        ichRx = 0;
        macro_enable_interrupts();

        return -2;
    }
    // Was a 0 character CR+LF terminated string received?
    if(2 == ichRx )
    {
        // A zero character length CR+LF terminated string was received.
        macro_disable_interrupts;
        fRxDone = 0;
        ichRx = 0;
        macro_enable_interrupts();
        return -3;
    }

    // copy the received chars to the destination location.
    for(ich = 0; ich < ichRx - 2; ich++)
    {
        *pchBuff = rgchRx[ich];
        pchBuff++;
    }
    *pchBuff = '\0';

    macro_disable_interrupts;
    fRxDone = 0;
    ichRx = 0;
    macro_enable_interrupts();
    return ich;
}
```

Multiway

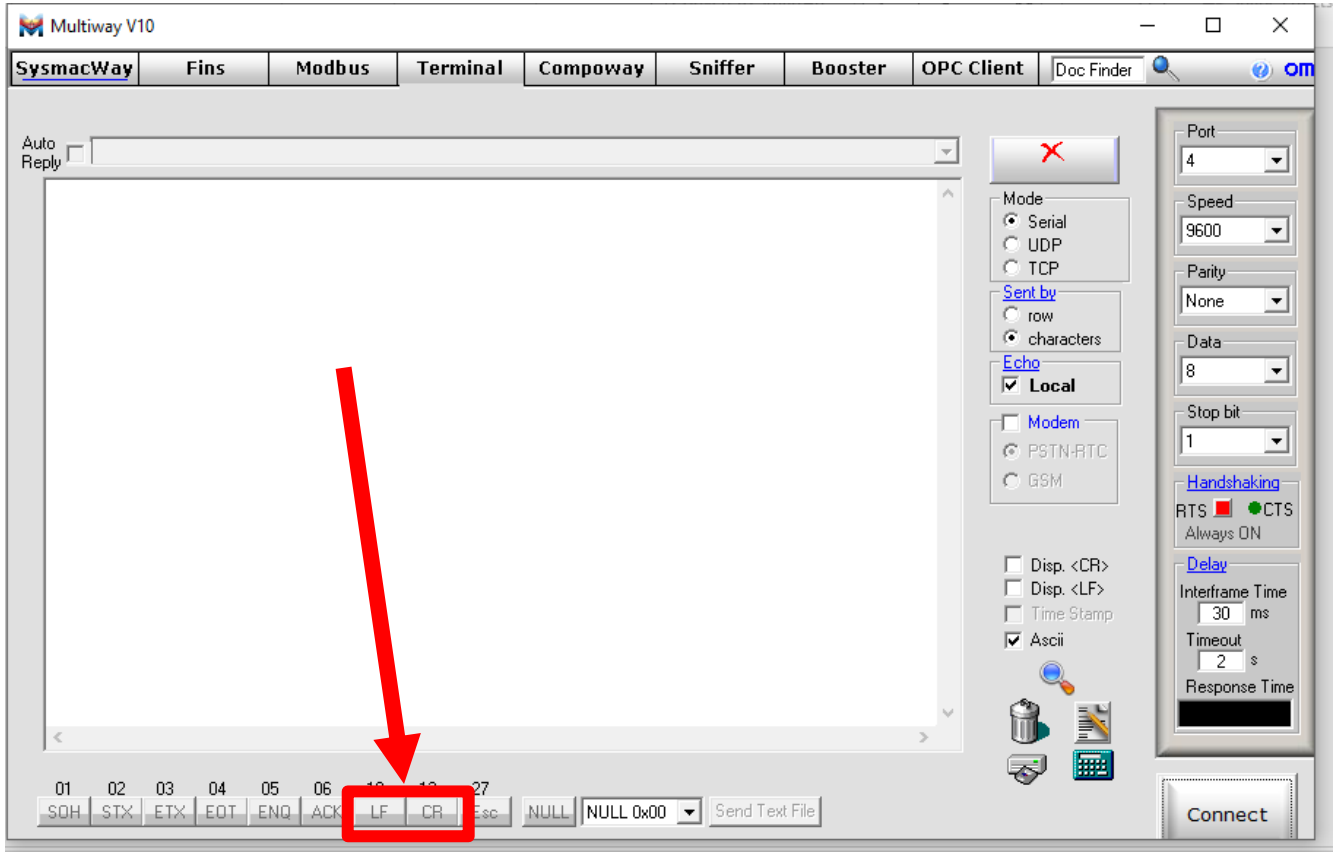

Multiway V10
SysmacWay
Fins
Modbus
Terminal
Compoway
Sniffer
Booster
OPC Client
Doc Finder
Auto Reply
Mode
Serial
UDP
TCP
Sent by
row
characters
Echo
Local
Modem
PSTN-RTC
GSM
Disp. <CR>
Disp. <LF>
Time Stamp
Ascii
Port
4
Speed
9600
Parity
None
Data
8
Stop bit
1
Handshaking
RTS
CTS
Always ON
Delay
Interframe Time
30
ms
Timeout
2
s
Response Time
01 SOH
02 STX
03 ETX
04 EOT
05 ENQ
06 ACK
10 LF
13 CR
27 Esc
NULL
NULL 0x00
Send Text File
Connect

Multiway


Multiway V10
SysmacWay
Fins
Modbus
Terminal
Compoway
Sniffer
Booster
OPC Client
Doc Finder
Auto Reply
Mode
Serial
UDP
TCP
Sent by
row
characters
Echo
Local
Modem
PSTN-RTC
GSM
Disp. <CR>
Disp. <LF>
Time Stamp
Ascii
Port
4
Speed
9600
Parity
None
Data
8
Stop bit
1
Handshaking
RTS
CTS
Always ON
Delay
Interframe Time
30
ms
Timeout
2
s
Response Time
01 SOH
02 STX
03 ETX
04 EOT
05 ENQ
06 ACK
10 LF
13 CR
27 Esc
NULL
NULL 0x00
Send Text File
Connect

Attention

La méthode avec interruption attend un CRLF pour lever l'interruption



Exercices supplémentaires l'UART

Test # 1

Visualisation à l'oscilloscope/logic analyzer de la trame

Test # 2

Utilisation d'un logiciel *Terminal* (port série)
TeraTerm, *multiway et putty* sont disponibles

Test # 3

Utilisation du mode « Debug » dans MPLABX
Visualisation des valeurs des registres (RCREG et TXREG)

Test # 4 **Analog discovery (digilent inc.)**