les écoles supérieures

ETML

MINF
Mise en œuvre des microcontrôleurs PIC32MX

Chapitre 7

Utilisation de l'USART

🛠 T.P. PIC32MX

Christian HUBER (CHR)
Serge CASTOLDI (SCA)
Version 1.82 novembre 2021

CONTENU DU CHAPITRE 7

7. UTILISATION DE L'USART

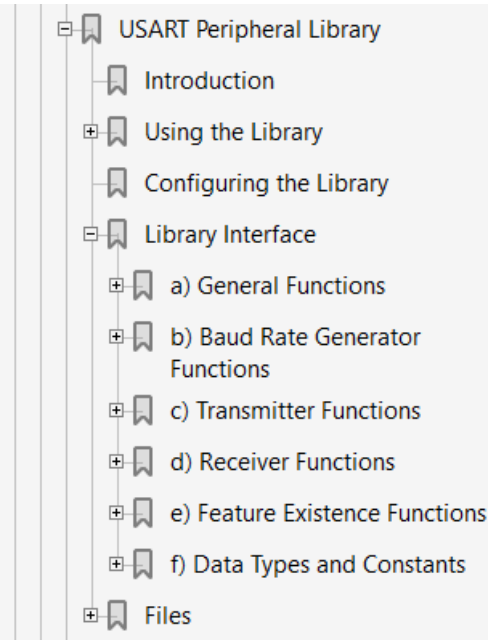
Dans ce chapitre, nous allons étudier comment utiliser l'USART du PIC32MX dans le but de réaliser une communication RS232 avec le kit PIC32MX795F512L.

Les documents de référence sont :

- La documentation "PIC32 Family Reference Manual" :
  Section 21 : UART
- Pour les détails spécifiques au PIC32MX795F512L, il faut se référer au document "PIC32MX5XX/6XX/7XX Family Data Sheet" :
  Section 19 : UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART)
- La documentation d'Harmony, qui se trouve dans <Répertoire Harmony>\v<n>\doc :
  Section MPLAB Harmony Framework Reference > Peripheral Libraries Help,
  sous-section USART Peripheral Library

7.1. UTILISATION DE LA PLIB_USART

Voici l'organisation de la documentation de l'USART Peripheral Library dans la documentation Harmony :



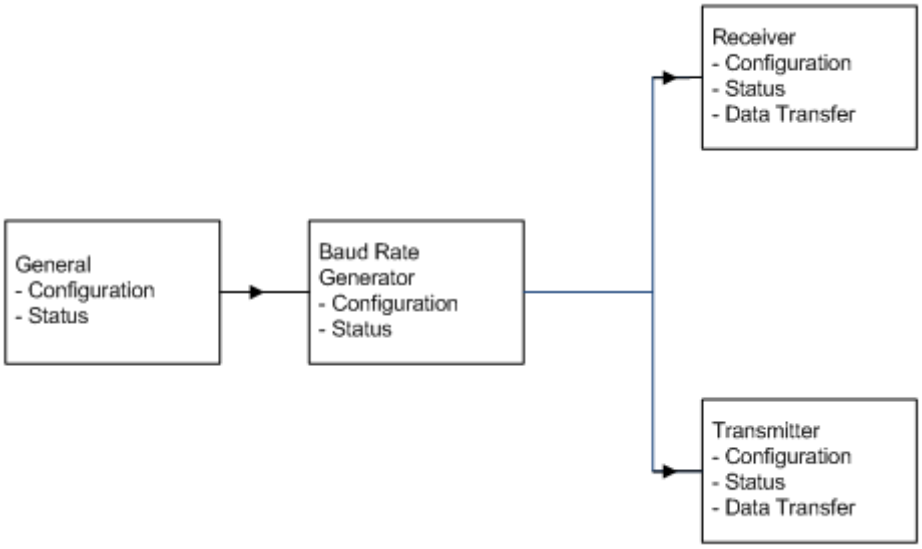
On s'aperçoit que les fonctions sont catégorisées selon leur utilité :

Library Overview

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the USART module.

Library Interface Section	Description
General Functions	Provides setup and configuration interface routines for • IrDA • Hardware Flow Control • Operation in power saving modes. • and other general setup
Baud Rate Generator Functions	Provides setup and configuration interface routines together with status routines for Baud Rate Generator (BRG).
Transmitter Functions	Provides setup, data transfer, error and the status interface routines for the transmitter.
Receiver Functions	Provides setup, data transfer, error and the status interface routines for the receiver.
Feature Existence Functions	Provides interface routines to determine existence of features.

Le modèle d'utilisation est le suivant :



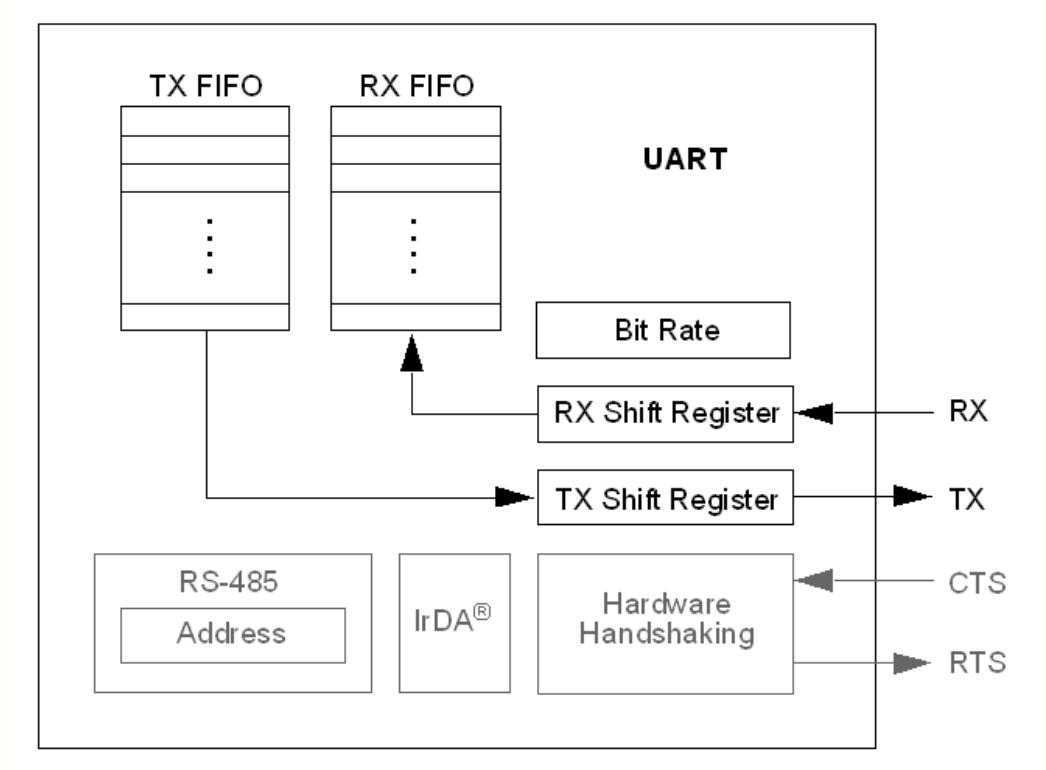
Selon toute logique, les opérations sont donc, dans l'ordre :

1. Configuration générale de l'USART.
2. Configuration du baudrate.
3. Configuration, puis utilisation des parties réception et émission.

7.2. CONFIGURATION DE L'USART

Pour bien configurer l'USART, il faut comprendre son architecture.

7.2.1. SCHÉMA BLOC DE PRINCIPE DE L'UART



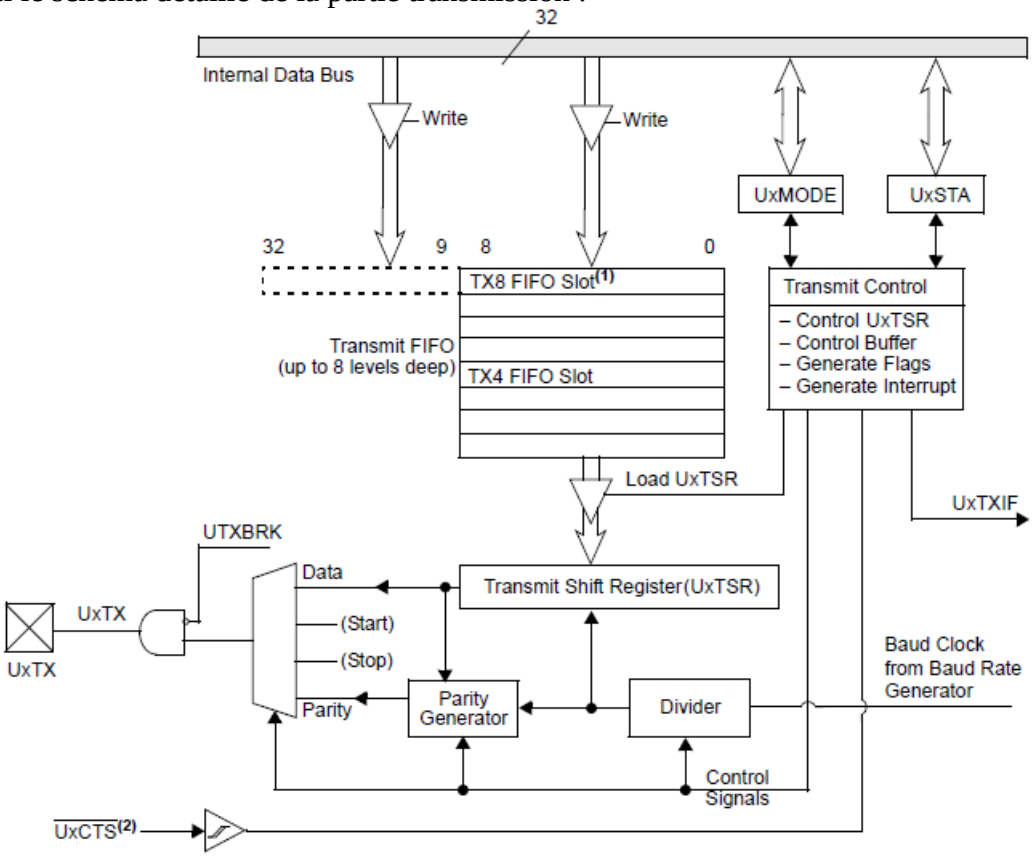
7.2.1. LISTE DES UART À DISPOSITION (PIC32MX795F512L)

On dispose de 6 UARTs. Les UARTS 1, 2 et 3 disposent de signaux de handshake (/CTS et /RTS) gérés matériellement.

Pin Name	Pin Number(1)			Pin Type	Buffer Type	Description
	64-Pin QFN/TQFP	100-Pin TQFP	121-Pin XBGA			
U1CTS	43	47	L9	I	ST	UART1 clear to send
U1RTS	49	48	K9	O	—	UART1 ready to send
U1RX	50	52	K11	I	ST	UART1 receive
U1TX	51	53	J10	O	—	UART1 transmit
U3CTS	8	14	F3	I	ST	UART3 clear to send
U3RTS	4	10	E3	O	—	UART3 ready to send
U3RX	5	11	F4	I	ST	UART3 receive
U3TX	6	12	F2	O	—	UART3 transmit
U2CTS	21	40	K6	I	ST	UART2 clear to send
U2RTS	29	39	L6	O	—	UART2 ready to send
U2RX	31	49	L10	I	ST	UART2 receive
U2TX	32	50	L11	O	—	UART2 transmit
U4RX	43	47	L9	I	ST	UART4 receive
U4TX	49	48	K9	O	—	UART4 transmit
U6RX	8	14	F3	I	ST	UART6 receive
U6TX	4	10	E3	O	—	UART6 transmit
U5RX	21	40	K6	I	ST	UART5 receive
U5TX	29	39	L6	O	—	UART5 transmit

7.2.2. SCHÉMA DÉTAILLÉ DE LA TRANSMISSION

Voici le schéma détaillé de la partie transmission :

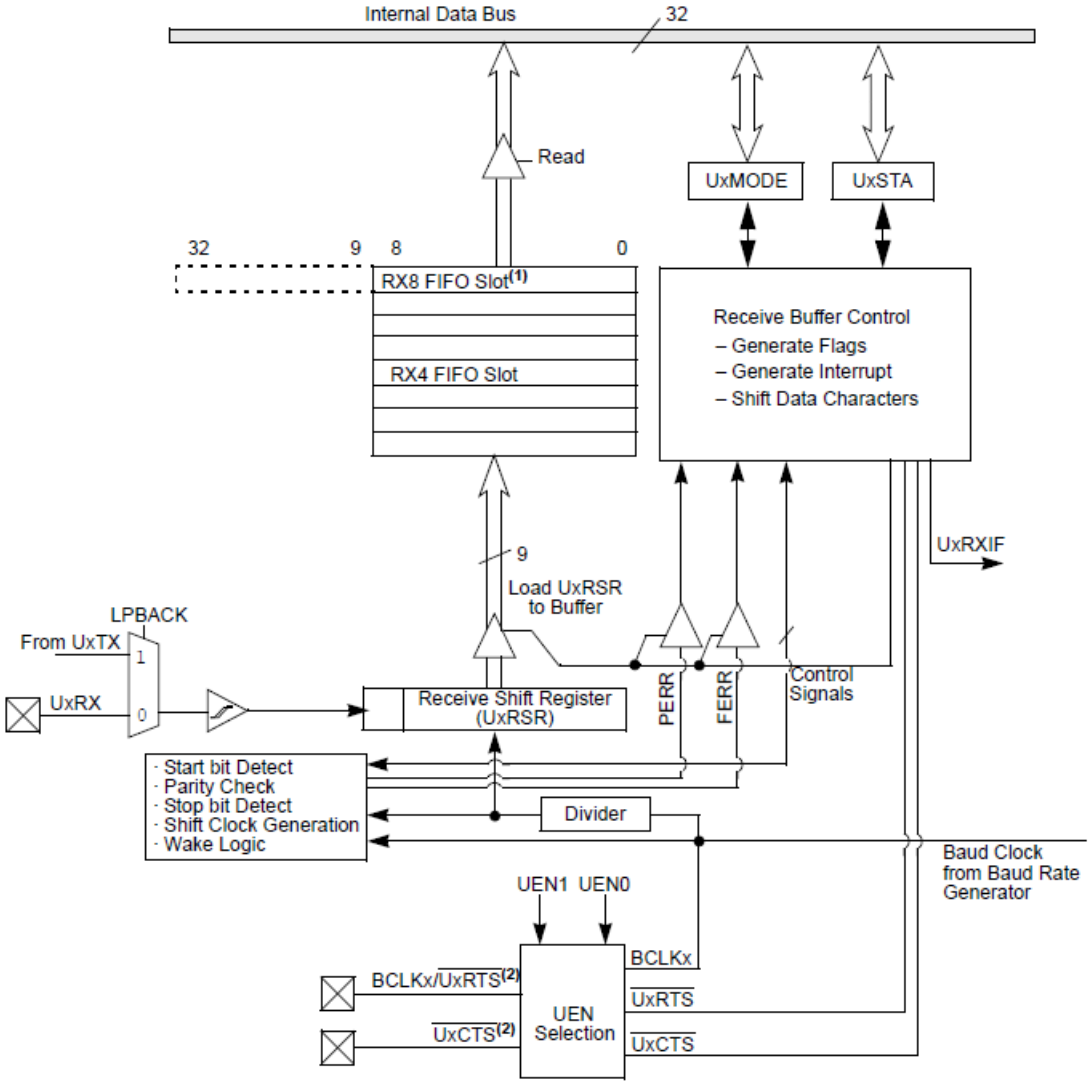


Note 1: Refer to the specific device data sheet for availability of 8-level-deep FIFO.
2: Refer to the specific device data sheet for availability of $\overline{\text{UxCTS}}$ pin.

Le PIC32MX795F512L possède un FIFO matériel de 8 éléments.
On constate également la présence de la broche d'entrée /UxCTS qui permet l'autorisation/blocage de l'émission par hardware.

7.2.3. SCHÉMA DÉTAILLÉ DE LA RÉCEPTION

Voici le schéma détaillé de la réception :



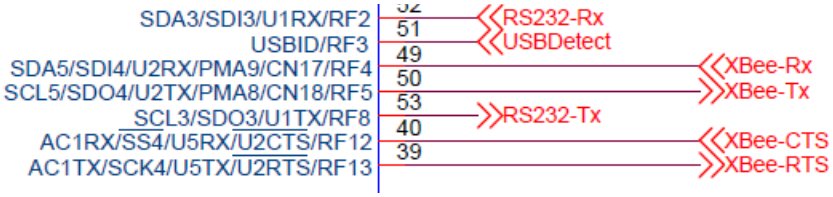
Note 1: Refer to the specific device data sheet for availability of 8-level-deep FIFO.
2: Refer to the specific device data sheet for availability of UxRTS and UxCTS pins.

Le PIC32MX795F512L possède un FIFO matériel de 8 éléments.
On constate la présence des broches /UxRTS et /UxCTS qui permettent le handshaking hardware.

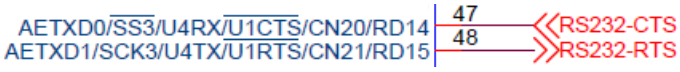
7.2.4. RÉALISATION DE LA LIAISON RS232 SUR LE KIT

7.2.4.1. CABLAGE SUR LE PIC32MX795F512L

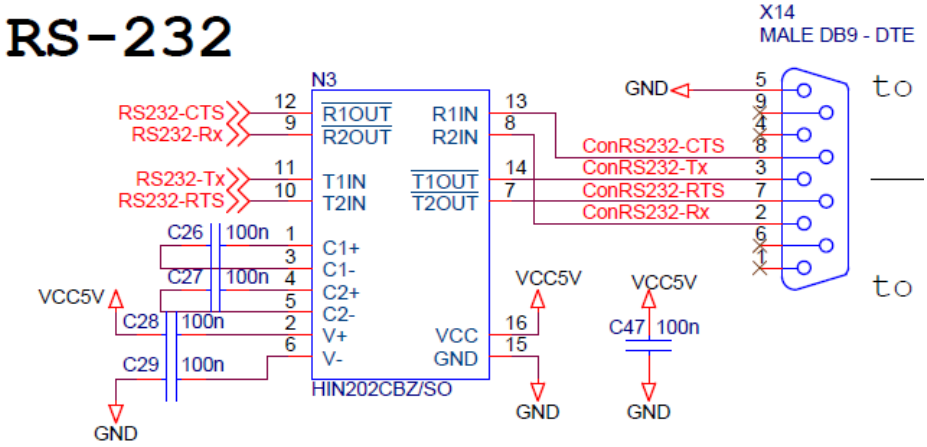
Comme on peut l'observer sur l'extrait du câblage du PIC32MX795F512L, c'est l'UART 1 qui est câblé sur le kit :



Les signaux /CTS et /RTS sont câblés sur les broches permettant un traitement automatique (par le PIC) du handshaking.

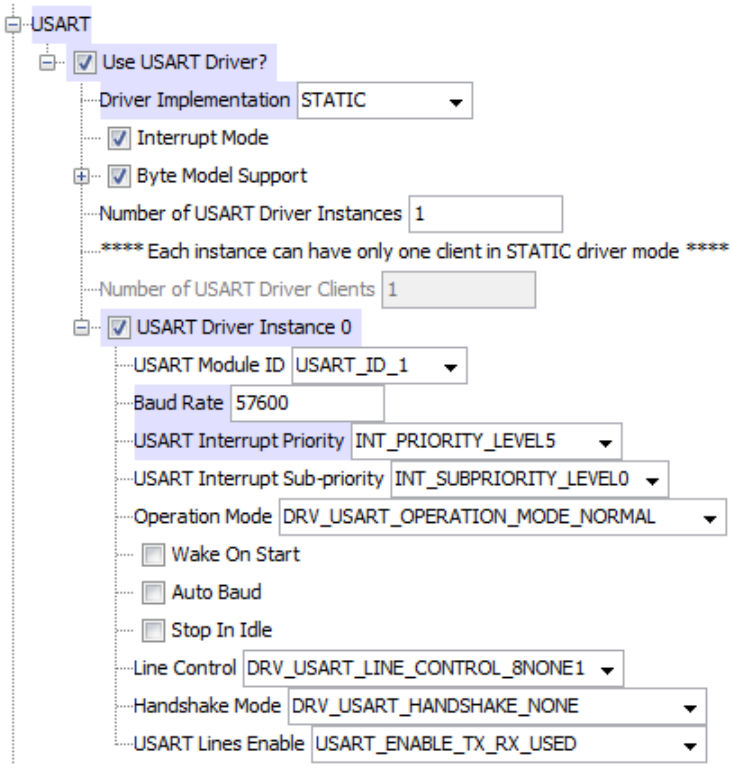


7.2.4.2. ADAPTATION AUX NIVEAUX RS232



7.3. INITIALISATION DE L'USART AVEC MHC

Pour découvrir les fonctions de configuration, nous utilisons le Microchip Harmony Configurator et nous introduisons un driver USART avec les choix suivants :



Cette configuration a été effectuée avec les versions de logiciels suivants :

- MPLABX 4.15
- Harmony 2.05

Ceci correspond à utiliser l'USART 1 avec un baudrate de 57600 [Bd].
On ne peut pas choisir le comportement des interruptions de réception et d'émission. Il faudra le faire manuellement.
Les éléments pour lesquels le choix n'est pas évident sont développés ci-après.

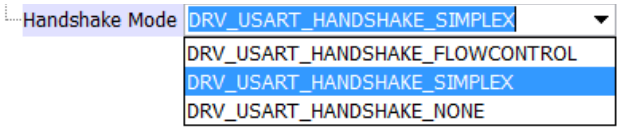
7.3.1. HANDSHAKE MODE

Pour le Handshake mode, on dispose de 2 possibilités :

Members	Description
USART_HANDSHAKE_MODE_FLOW_CONTROL	Enables flow control
USART_HANDSHAKE_MODE_SIMPLEX	Enables simplex mode communication, no flow control

Choix du mode SIMPLEX pour ne pas activer l'automatisme de gestion et permettre une gestion par logiciel pour découvrir le principe.

Remarque : Au niveau du MHC, le menu déroulant a 3 éléments :



Si on choisit NONE, il n'y a pas génération de la fonction de configuration :

Par contre si on choisit SIMPLEX on obtient la génération de la fonction PLIB_USART_HandshakeModeSelect.

```
PLIB_USART_HandshakeModeSelect(USART_ID_1,
                               DRV_USART_HANDSHAKE_SIMPLEX);
```

7.3.2. OPERATION MODE

Pour l'operation mode, on dispose de 4 possibilités :

Members	Description
DRV_USART_OPERATION_MODE_IRDA	USART works in IRDA mode
DRV_USART_OPERATION_MODE_NORMAL	This is the normal point to point communication mode where the USART communicates directly with another USART by connecting it's Transmit signal to the external USART's Receiver signal and vice versa. An external transceiver may be connected to obtain RS-232 signal levels. This type of connection is typically full duplex.
DRV_USART_OPERATION_MODE_ADDRESSED	This is a multi-point bus mode where the USART can communicate with many other USARTS on a bus using an address-based protocol such as RS-485. This mode is typically half duplex and the physical layer may require a transceiver. In this mode every USART on the bus is assigned an address and the number of data bits is 9 bits
DRV_USART_OPERATION_MODE_LOOPBACK	Loopback mode internally connects the Transmit signal to the Receiver signal, looping data transmission back into this USART's own input. It is useful primarily as a test mode.

Utilisation du mode normal.

Au niveau du code généré, on obtient :

```
/* Initialize the USART based on configuration settings */
PLIB_USART_InitializeModeGeneral(USART_ID_1,
            false,    /* Auto baud*/
            false,    /* LoopBack mode*/
            false,    /* Auto wakeup on start*/
            false,    /* IRDA mode*/
            false);   /* Stop In Idle mode*/
```

7.3.3. LINE CONTROL MODE

Pour le Line Control Mode on dispose de 8 possibilités :

Members	Description
USART_8N1	8 Data Bits, No Parity,one Stop Bit
USART_8E1	8 Data Bits, Even Parity, 1 stop bit
USART_8O1	8 Data Bits, odd Parity, 1 stop bit
USART_8N2	8 Data Bits, No Parity,two Stop Bits
USART_8E2	8 Data Bits, Even Parity, 2 stop bits
USART_8O2	8 Data Bits, odd Parity, 2 stop bits
USART_9N1	9 Data Bits, No Parity, 1 stop bit
USART_9N2	9 Data Bits, No Parity, 2 stop bits

Notre choix est 8N1 pour 8 bits data, pas de parité et un seul bit de stop.

Au niveau du code généré, on obtient :

```
/* Set the line control mode */
PLIB_USART_LineControlModeSelect(USART_ID_1,
                                 DRV_USART_LINE_CONTROL_8NONE1);
```

7.3.4. CONFIGURATION DES MODES D'INTERRUPTION

☹ Le MHC ne permet pas un réglage des interruptions de l'USART en fonction du niveau des buffers. Toutefois, le PIC32MX795F512L offre cette possibilité.

On obtient par défaut la configuration suivante avec l'appel de la fonction PLIB_USART_InitializeOperation :

```
PLIB_USART_InitializeOperation(USART_ID_1,
            USART_RECEIVE_FIFO_ONE_CHAR,
            USART_TRANSMIT_FIFO_EMPTY,
            USART_ENABLE_TX_RX_USED);
```

On aura une interruption de réception à chaque caractère reçu, et une interruption d'émission dès que le buffer d'envoi est vide. En cas de nécessité, ceci pourra être modifié manuellement (voir ci-dessous).

7.3.5. LA FONCTION PLIB_USART_INITIALIZEOPERATION

Cette fonction configure trois aspects :

- Le mode de l'interruption de réception
- Le mode de l'interruption d'émission
- Le mode d'utilisation de l'USART

Voici le prototype de cette fonction :

```
void PLIB_USART_InitializeOperation(USART_MODULE_ID index,
    USART_RECEIVE_INTR_MODE receiveInterruptMode,
    USART_TRANSMIT_INTR_MODE transmitInterruptMode,
    USART_OPERATION_MODE operationMode);
```

7.3.5.1. USART_RECEIVE_INTR_MODE

Permet de configurer le comportement de l'interruption de réception en relation avec la situation du FIFO hardware de réception.

Nous disposons de 3 possibilités :

Members	Description
USART_RECEIVE_FIFO_HALF_FULL	Interrupt when receive buffer is half full
USART_RECEIVE_FIFO_3B4FULL	Interrupt when receive buffer is 3/4 full
USART_RECEIVE_FIFO_ONE_CHAR	Interrupt when a character is received

Comme la taille du FIFO hardware de réception est de 8, avec HALF_FULL l'interruption se produit lorsque l'on a reçu 4 caractères. Avec 3B4FULL, l'interruption se produit lorsque l'on a reçu 6 caractères et avec ONE_CHAR l'interruption se produit dès que l'on a reçu un caractère.

- Les choix 3B4FULL et HALF_FULL peuvent être judicieux en cas de réception continuelle de trames. Le CPU n'est alors pas interrompu à chaque caractère.
- Le mode ONE_CHAR exploite peu le FIFO hardware, mais permet de traiter au fur et à mesure de l'arrivée des caractères.

7.3.5.2. USART_TRANSMIT_INTR_MODE

Permet de configurer le comportement de l'interruption d'émission en relation avec la situation du FIFO hardware d'émission.

Nous disposons de 3 possibilités :

Members	Description
USART_TRANSMIT_FIFO_EMPTY	Interrupt when the transmit buffer becomes empty
USART_TRANSMIT_FIFO_IDLE	Interrupt when all characters are transmitted
USART_TRANSMIT_FIFO_NOT_FULL	Interrupt when at least one location is empty in the transmit buffer

- Avec FIFO_EMPTY on obtient l'interruption lorsque le FIFO d'émission est vide, c'est notre choix.
- La situation FIFO_IDLE correspond à attendre que le FIFO soit vide et que le dernier caractère ait été transmis.
- Avec la situation FIFO_NOT_FULL, on obtient l'interruption dès qu'il y a une place dans le FIFO d'émission. Cela signifie que l'interruption se produit après chaque caractère transmis.

☝ C'est dans le traitement de l'interruption que l'on étudiera le principe d'utilisation.

7.3.5.3. USART_OPERATION_MODE

Permet de configurer l'utilisation des éléments de l'USART.

Nous disposons de 4 possibilités :

Members	Description
USART_ENABLE_TX_RX_BCLK_USED	TX, RX and BCLK pins are used by USART module
USART_ENABLE_TX_RX_CTS_RTS_USED	TX, RX, CTS and RTS pins are used by USART module
USART_ENABLE_TX_RX_RTS_USED	TX, RX and RTS pins are used by USART module
USART_ENABLE_TX_RX_USED	TX and RX pins are used by USART module

Il y a une relation avec le choix du mode de handshake :
Avec HANDSHAKE_SIMPLEX, on obtient :

```
PLIB_USART_InitializeOperation(USART_ID_1,
            USART_RECEIVE_FIFO_ONE_CHAR,
            USART_TRANSMIT_FIFO_IDLE,
            USART_ENABLE_TX_RX_USED);
```

OU en fonction des essais successifs :

```
            USART_ENABLE_TX_RX_CTS_RTS_USED);
```

Avec HANDSHAKE_FLOWCONTROL, on obtient :

```
PLIB_USART_InitializeOperation(USART_ID_1,
            USART_RECEIVE_FIFO_ONE_CHAR,
            USART_TRANSMIT_FIFO_IDLE,
            USART_ENABLE_TX_RX_CTS_RTS_USED);
```

Avec HANDSHAKE_NONE, on obtient la même situation que simplex:

```
PLIB_USART_InitializeOperation(USART_ID_1,
            USART_RECEIVE_FIFO_ONE_CHAR,
            USART_TRANSMIT_FIFO_IDLE,
            USART_ENABLE_TX_RX_USED);
```

Par contre, la fonction PLIB_USART_HandshakeModeSelect n'est pas appelée.

 Il peut arriver, lors du passage d'un mode à l'autre, qu'on finisse par obtenir USART_ENABLE_TX_RX_CTS_RTS_USED avec le mode SIMPLEX, ce qui crée un conflit.
Il faut donc utiliser NONE si on ne veut pas utiliser automatiquement RTS et CTS.

7.3.6. CODE C GÉNÉRÉ PAR LE MHC

La fonction **DRV_USART0_Initialize()** est appelée par la fonction SYS_Initialize() du fichier system_init.c. Elle est implémentée dans le fichier drv_usart_static.c.
Situation avec HANDSHAKE_NONE :

```
SYS_MODULE_OBJ DRV_USART0_Initialize(void)
{
    uint32_t clockSource;

    /* Disable the USART module to configure it*/
    PLIB_USART_Disable (USART_ID_1);

 /* Initialize the USART based on configuration settings */
    PLIB_USART_InitializeModeGeneral(USART_ID_1,
            false,  /*Auto baud*/
            false,  /*LoopBack mode*/
            false,  /*Auto wakeup on start*/
            false,  /*IRDA mode*/
            false);  /*Stop In Idle mode*/

    /* Set the line control mode */
    PLIB_USART_LineControlModeSelect(USART_ID_1,
                          DRV_USART_LINE_CONTROL_8NONE1);

    /* We set the receive interrupt mode to receive an
interrupt whenever FIFO is not empty */
    PLIB_USART_InitializeOperation(USART_ID_1,
            USART_RECEIVE_FIFO_ONE_CHAR,
            USART_TRANSMIT_FIFO_IDLE,
            USART_ENABLE_TX_RX_USED);

    /* Get the USART clock source value*/
    clockSource =
     SYS_CLK_PeripheralFrequencyGet (CLK_BUS_PERIPHERAL_1);

    /* Set the baud rate and enable the USART */
    PLIB_USART_BaudSetAndEnable(USART_ID_1, clockSource,
                   57600);  /*Desired Baud rate value*/

    /* Clear the interrupts to be on the safer side*/
    SYS_INT_SourceStatusClear(INT_SOURCE_USART_1_TRANSMIT);
    SYS_INT_SourceStatusClear(INT_SOURCE_USART_1_RECEIVE);
    SYS_INT_SourceStatusClear(INT_SOURCE_USART_1_ERROR);

    /* Enable the error interrupt source */
    SYS_INT_SourceEnable(INT_SOURCE_USART_1_ERROR);

    /* Enable the Receive interrupt source */
    SYS_INT_SourceEnable(INT_SOURCE_USART_1_RECEIVE);

    /* Return the driver instance value*/
    return (SYS_MODULE_OBJ)DRV_USART_INDEX_0;
}
```

On constate la correspondance entre les éléments configurés et les fonctions de configurations de la PLIB_USART.

Au niveau de la configuration de l'interruption, on constate qu'il y a 3 sources d'interruption pour le même vecteur.

☝ Il sera donc nécessaire de traiter dans la même interruption la réception et la transmission, ainsi que la situation d'erreur.

7.3.7. MODIFICATIONS NÉCESSAIRES

Pour obtenir une configuration adaptée au traitement des interruptions de l'USART, il faut effectuer la modification suivante au niveau du fichier drv_usart_static.c.

Situation après génération :

```
PLIB_USART_InitializeOperation(USART_ID_1,
                               USART_RECEIVE_FIFO_ONE_CHAR,
                               USART_TRANSMIT_FIFO_IDLE,
                               USART_ENABLE_TX_RX_USED);
```

Situation après modification :

```
PLIB_USART_InitializeOperation(USART_ID_1,
                               USART_RECEIVE_FIFO_ONE_CHAR,
                               USART_TRANSMIT_FIFO_EMPTY,
                               USART_ENABLE_TX_RX_USED);
```

☝ On conserve la configuration par défaut USART_RECEIVE_FIFO_ONE_CHAR.

Liste des modes pour TX interrupt :

```
typedef enum {
    USART_TRANSMIT_FIFO_NOT_FULL = 0x00,
    USART_TRANSMIT_FIFO_IDLE = 0x01,
    USART_TRANSMIT_FIFO_EMPTY = 0x02
} USART_TRANSMIT_INTR_MODE;
```

Liste des modes pour RX interrupt :

```
typedef enum {
    USART_RECEIVE_FIFO_3B4FULL = 0x02,
    USART_RECEIVE_FIFO_HALF_FULL = 0x01,
    USART_RECEIVE_FIFO_ONE_CHAR = 0x00
} USART_RECEIVE_INTR_MODE;
```

7.4. FONCTIONS DE L'EMETTEUR

On dispose d'un certain nombre de fonctions liées à l'émission.

	Name	Description
	PLIB_USART_Transmitter9BitsSend	Data to be transmitted in the byte mode with the 9th bit.
	PLIB_USART_TransmitterBreakSend	Transmits the break character.
	PLIB_USART_TransmitterBreakSendIsComplete	Returns the status of the break transmission
	PLIB_USART_TransmitterBufferIsFull	Gets the transmit buffer full status.
	PLIB_USART_TransmitterByteSend	Data to be transmitted in the Byte mode.
	PLIB_USART_TransmitterDisable	Disables the specific USART module transmitter.
	PLIB_USART_TransmitterEnable	Enables the specific USART module transmitter.
	PLIB_USART_TransmitterIdleIsLowDisable	Disables the Transmit Idle Low state.
	PLIB_USART_TransmitterIdleIsLowEnable	Enables the Transmit Idle Low state.
	PLIB_USART_TransmitterInterruptModeSelect	Sets the USART transmitter interrupt mode.
	PLIB_USART_TransmitterIsEmpty	Gets the transmit shift register empty status.
	PLIB_USART_TransmitterAddressGet	Returns the address of the USART TX register

Voici la description des fonctions les plus utiles.

7.4.1. LA FONCTION PLIB_USART_TRANSMITTERBYTESEND

La fonction **PLIB_USART_TransmitterByteSend** permet de transmettre un octet.

```
void PLIB_USART_TransmitterByteSend(USART_MODULE_ID index, int8_t data);
```

7.4.2. LA FONCTION PLIB_USART_TRANSMITTERBUFFERISFULL

La fonction **PLIB_USART_TransmitterBufferIsFull** permet de connaitre la situation du tampon d'émission.

```
bool PLIB_USART_TransmitterBufferIsFull(USART_MODULE_ID index);
```

Si true, le buffer est plein. Avec false le buffer n'est pas plein, il y a au moins la place pour un caractère.

7.5. FONCTIONS DU RECEPTEUR

On dispose d'un certain nombre de fonctions liées à la réception.

	Name	Description
	PLIB_USART_ReceiverAddressAutoDetectDisable	Disables the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressAutoDetectEnable	Setup the automatic Address Detect mode.
	PLIB_USART_ReceiverAddressDetectDisable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressDetectEnable	Enables the Address Detect mode.
	PLIB_USART_ReceiverAddressIsReceived	Checks and return if the data received is an address.
	PLIB_USART_ReceiverByteReceive	Data to be received in the Byte mode.
	PLIB_USART_ReceiverDataIsAvailable	Identifies if the receive data is available for the specified USART module.
	PLIB_USART_ReceiverDisable	Disables the USART receiver.
	PLIB_USART_ReceiverEnable	Enables the USART receiver.
	PLIB_USART_ReceiverFramingErrorHasOccurred	Gets the framing error status.
	PLIB_USART_ReceiverIdleStateLowDisable	Disables receive polarity inversion.
	PLIB_USART_ReceiverIdleStateLowEnable	Enables receive polarity inversion.
	PLIB_USART_ReceiverInterruptModeSelect	Sets the USART receiver FIFO level.
	PLIB_USART_ReceiverIsIdle	Identifies if the receiver is idle.
	PLIB_USART_ReceiverOverrunErrorClear	Clears a USART vverrun error.
	PLIB_USART_ReceiverOverrunHasOccurred	Identifies if there was a receiver overrun error.
	PLIB_USART_ReceiverParityErrorHasOccurred	Gets the parity error status.
	PLIB_USART_ReceiverAddressGet	Returns the address of the USART RX register
	PLIB_USART_Receiver9BitsReceive	Data to be received in the byte mode with the 9th bit.

Voici la description des fonctions les plus utiles.

7.5.1. LA FONCTION PLIB_USART_RECEIVERBYTERECEIVE

La fonction **PLIB_USART_ReceiverByteReceive** permet de lire un byte des données reçu par l'USART et stocké dans le tampon de réception.

```
int8_t PLIB_USART_ReceiverByteReceive(USART_MODULE_ID index);
```

☝ Il faut au préalable s'assurer qu'un byte est disponible.

7.5.2. LA FONCTION PLIB_USART_RECEIVERDATAISAVAILABLE

La fonction **PLIB_USART_ReceiverDataIsAvailable** permet de savoir si une donnée est disponible.

```
bool PLIB_USART_ReceiverDataIsAvailable(USART_MODULE_ID index);
```

7.5.2.1. PLIB_USART_RECEIVERDATAISAVAILABLE, EXEMPLE

Cet exemple repris de la réponse à l'interruption de réception montre comment exploiter le tampon de réception.

```
int8_t c;

while (PLIB_USART_ReceiverDataIsAvailable(USART_ID_1))
{
    c = PLIB_USART_ReceiverByteReceive(USART_ID_1);
    PutCharInFifo ( &descrFifoRX, c);
}
```

7.5.3. FONCTION PLIB_USART_RECEIVEROVERRUNERRORCLEAR

La fonction **PLIB_USART_ReceiverOverrunErrorClear** permet de supprimer une erreur d'overrun. L'appel de cette fonction efface les données reçues.

```
void PLIB_USART_ReceiverOverrunErrorClear(USART_MODULE_ID index);
```

7.6. FONCTIONS GENERALES DE L'USART

Voici les nombreuses fonctions générales (ne s'appliquent pas toutes pour le suivant le modèle de microcontrôleur) :

Name	Description
PLIB_USART_Disable	Disables the specific USART module
PLIB_USART_Enable	Enables the specific USART module.
PLIB_USART_HandshakeModeSelect	Sets the data flow configuration.
PLIB_USART_IrDADisable	Disables the IrDA encoder and decoder.
PLIB_USART_IrDAEnable	Enables the IrDA encoder and decoder.
PLIB_USART_LineControlModeSelect	Sets the data flow configuration.
PLIB_USART_LoopbackDisable	Disables Loopback mode.
PLIB_USART_LoopbackEnable	Enables Loopback mode.
PLIB_USART_OperationModeSelect	Configures the operation mode of the USART module.
PLIB_USART_StopInIdleDisable	Disables the Stop in Idle mode (the USART module continues operation when the device is in Idle mode).
PLIB_USART_StopInIdleEnable	Discontinues operation when the device enters Idle mode.
PLIB_USART_WakeOnStartDisable	Disables the wake-up on start bit detection feature during Sleep mode.
PLIB_USART_WakeOnStartEnable	Enables the wake-up on start bit detection feature during Sleep mode.
PLIB_USART_WakeOnStartIsEnabled	Gets the state of the sync break event completion.
PLIB_USART_ErrorsGet	Return the status of all errors in the specified USART module.
PLIB_USART_InitializeModeGeneral	Enables or disables general features of the USART module.
PLIB_USART_InitializeOperation	Configures the Receive and Transmit FIFO interrupt levels and the hardware lines to be used by the module.
PLIB_USART_AddressGet	Gets the address for the Address Detect mode.
PLIB_USART_AddressMaskGet	Gets the address mask for the Address Detect mode.
PLIB_USART_AddressMaskSet	Sets the address mask for the Address Detect mode.
PLIB_USART_AddressSet	Sets the address for the Address Detect mode.
PLIB_USART_ModuleIsBusy	Returns the USART module's running status.
PLIB_USART_RunInOverflowDisable	Disables the Run in overflow condition mode.
PLIB_USART_RunInOverflowEnable	Enables the USART module to continue to operate when an overflow error condition has occurred.
PLIB_USART_RunInOverflowIsEnabled	Gets the status of the Run in Overflow condition.
PLIB_USART_RunInSleepModeDisable	Turns off the USART module's BRG clock during Sleep mode.
PLIB_USART_RunInSleepModeEnable	Allows the USART module's BRG clock to run when the device enters Sleep mode.
PLIB_USART_RunInSleepModeIsEnabled	Gets the status of Run in Sleep mode.

Voici la description des fonctions les plus utiles.

7.6.1. LA FONCTION PLIB_USART_DISABLE

La fonction **PLIB_USART_Disable** permet de désactiver l'USART. Cela a pour effet de vider les tampons d'émission et de réception.

```
void PLIB_USART_Disable(USART_MODULE_ID index);
```

7.6.2. LA FONCTION PLIB_USART_ENABLE

La fonction **PLIB_USART_Enable** permet d'activer l'USART. Cela a pour effet que les broches RX et TX sont gérées par l'USART.

```
void PLIB_USART_Enable(USART_MODULE_ID index);
```

7.6.3. LA FONCTION PLIB_USART_ERRORGET

La fonction **PLIB_USART_ErrorGet** permet d'obtenir la situation des bits d'erreurs.

```
USART_ERROR PLIB_USART_ErrorsGet(USART_MODULE_ID index);
```

Le type USART_ERROR est défini ainsi :

```
typedef enum {
    USART_ERROR_NONE = 0x00,
    USART_ERROR_RECEIVER_OVERRUN = 0x01,
    USART_ERROR_FRAMING = 0x02,
    USART_ERROR_PARITY = 0x04
} USART_ERROR;
```

7.6.4. EXEMPLE GESTION DES ERREURS

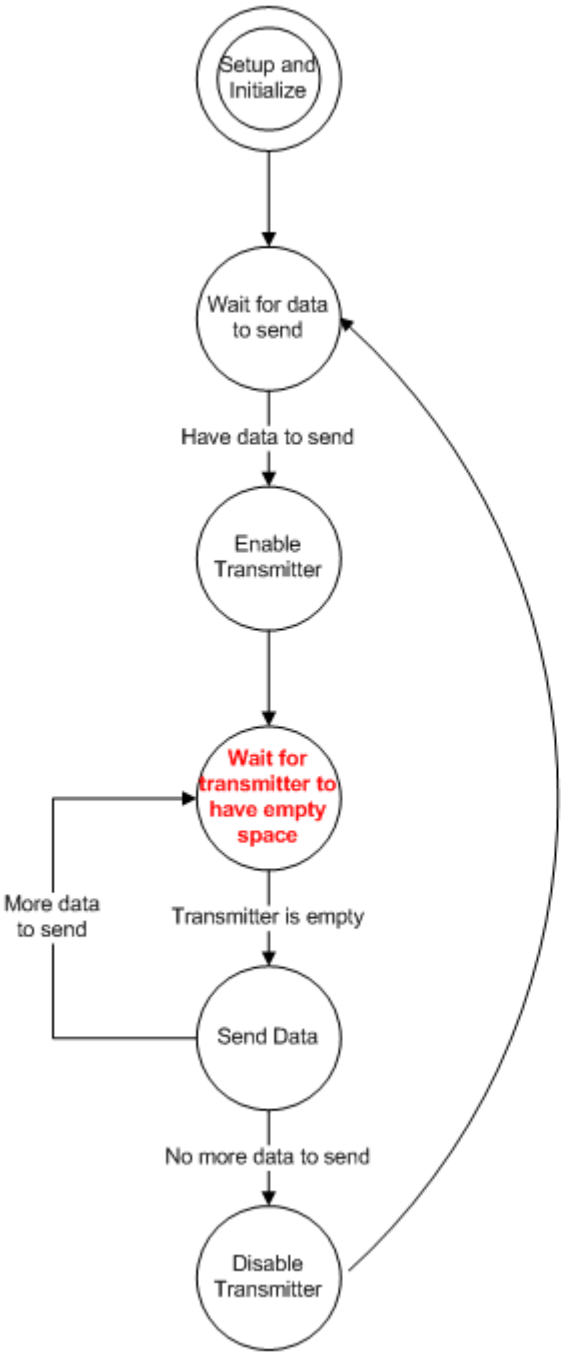
L'exemple repris de la réponse à l'interruption de réception montre le traitement des erreurs.

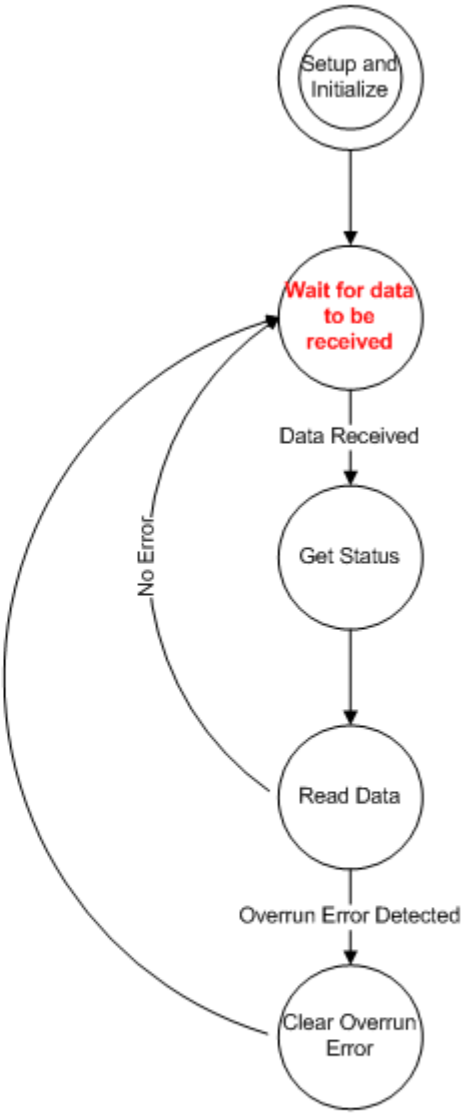
```
USART_ERROR UsartStatus = PLIB_USART_ErrorsGet(USART_ID_1);

if ( (UsartStatus & (USART_ERROR_PARITY |
                     USART_ERROR_FRAMING |
                     USART_ERROR_RECEIVER_OVERRUN)) == 0) {
     // OK traitement de réception possible
} else {
    // Suppression des erreurs
    // La lecture des erreurs les efface sauf pour overrun
    if ( (UsartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
                              USART_ERROR_RECEIVER_OVERRUN) {
         PLIB_USART_ReceiverOverrunErrorClear(USART_ID_1);
    }
}
```

7.7. REALISATION ISR DE L'USART

7.7.1. DIAGRAMME DE PRINCIPE

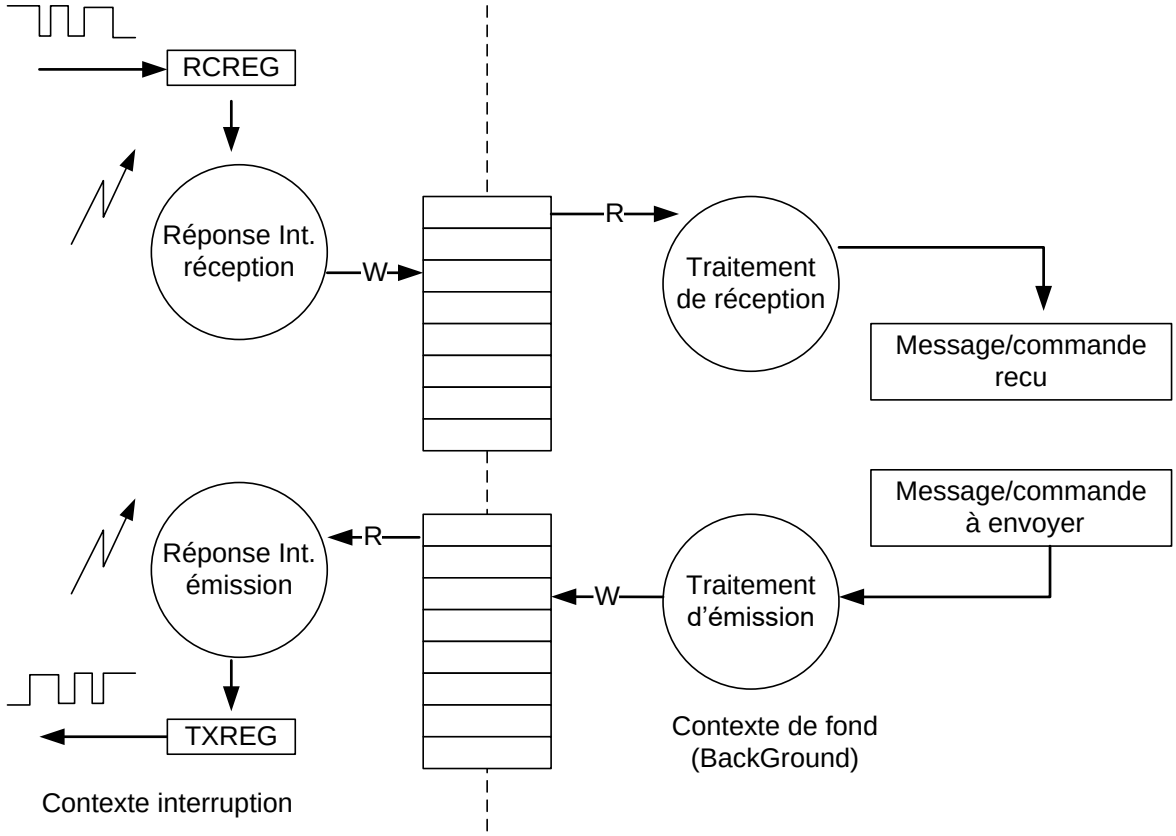
Dans la documentation Harmony, on trouve les 2 diagrammes d'état suivants :





7.7.2. CONCEPT ÉMISSION - RÉCEPTION AVEC FIFO

Le diagramme ci-dessous illustre le principe du traitement de la réception et de l'émission sous interruption en utilisant des FIFO software.



7.7.3. ISR GÉNÉRÉE PAR LE MHC

Dans la réponse à l'interruption USART (partagée entre émission, réception et erreur), Harmony génère 3 appels à des fonctions qui font partie du driver :

```
void __ISR(_UART_1_VECTOR, ipl5AUTO)
                          _IntHandlerDrvUsartInstance0(void)
{
    DRV_USART_TasksTransmit(sysObj.drvUsart0);
    DRV_USART_TasksReceive(sysObj.drvUsart0);
    DRV_USART_TasksError(sysObj.drvUsart0);
}
```

7.7.3.1. LA FONCTION DRV_USART_TASKSTRANSMIT

Comme on peut le constater, cette fonction teste s'il s'agit d'une interruption de transmission, mais à part la mise à zéro du flag elle ne fait rien ! C'est à l'utilisateur d'implémenter son traitement.

```
void DRV_USART0_TasksTransmit(void)
{
    /* This is the USART Driver Transmit tasks routine.
       In this function, the driver checks if a transmit
       interrupt is active and performs respective action*/

    /* Reading the transmit interrupt flag */
    if(SYS_INT_SourceStatusGet
                            (INT_SOURCE_USART_1_TRANSMIT))
    {
        /* Disable the interrupt,
           to avoid calling ISR continuously*/
        SYS_INT_SourceDisable(INT_SOURCE_USART_1_TRANSMIT);

        /* Clear up the interrupt flag */
        SYS_INT_SourceStatusClear
                            (INT_SOURCE_USART_1_TRANSMIT);
    }
}
```

7.7.3.2. LA FONCTION DRV_USART_TASKSRECEIVE

Comme on peut le constater, cette fonction teste s'il s'agit d'une interruption de réception, mais à part la mise à zéro du flag elle ne fait rien ! C'est à l'utilisateur d'implémenter son traitement.

```
void DRV_USART0_TasksReceive(void)
{
    /* This is the USART Driver Receive tasks routine.
       If the receive interrupt flag is set,
       the tasks routines are executed.
     */

    /* Reading the receive interrupt flag */
    if(SYS_INT_SourceStatusGet(INT_SOURCE_USART_1_RECEIVE))
    {

        /* Clear up the interrupt flag */
        SYS_INT_SourceStatusClear
                            (INT_SOURCE_USART_1_RECEIVE);
    }
}
```

7.7.3.3. LA FONCTION DRV_USART_TASKSERROR

Cette fonction teste s'il s'agit d'une interruption d'erreur, elle effectue un traitement partiel des erreurs.

```
void DRV_USART0_TasksError(void)
{
    /* This is the USART Driver Error tasks routine.
       In this function, the driver checks if an error
       interrupt has occurred.
       If so the error condition is cleared.  */

    /* Reading the error interrupt flag */
    if(SYS_INT_SourceStatusGet(INT_SOURCE_USART_1_ERROR))
    {
        /* This means an error has occurred */
        if(PLIB_USART_ReceiverOverrunHasOccurred
                                            (USART_ID_1))
        {
            PLIB_USART_ReceiverOverrunErrorClear
                                            (USART_ID_1);
        }

        /* Clear up the error interrupt flag */
        SYS_INT_SourceStatusClear
                                (INT_SOURCE_USART_1_ERROR);
    }
}
```

☝ Nous n'utiliserons pas ces fonctions mais nous reprendrons certains éléments dans l'élaboration d'une routine de réponse unique.

7.7.4. ISR USART, RÉALISATION PRATIQUE

Voici un exemple pratique de réalisation de la routine de réponse à l'interruption de l'UART. Pour bien comprendre les traitements, il faut rappeler que nous avons configuré la relation entre les FIFOS hardware et les interruptions de la manière suivante :

```
PLIB_USART_TransmitterInterruptModeSelect(USART_ID_1,
                                  USART_TRANSMIT_FIFO_EMPTY);
PLIB_USART_ReceiverInterruptModeSelect(USART_ID_1,
                                USART_RECEIVE_FIFO_ONE_CHAR);
```

☝ La réponse est prévue pour supporter un changement du **ReceiverInterruptModeSelect** à HALF_FULL ou 3B4FULL, d'où une boucle dans la réception.

```
void __ISR(_UART_1_VECTOR, ipl5AUTO)
                            _IntHandlerDrvUsartInstance0(void)
{
    uint8_t freeSize, TXsize;
    int8_t c;
    int8_t i_cts = 0;
    BOOL  TxBuffFull;

    USART_ERROR  UsartStatus;

    // Marque début interruption avec Led3
    LED3_W = 1;

    // Is this an RX interrupt ?
    if ( PLIB_INT_SourceFlagGet(INT_ID_0,
                            INT_SOURCE_USART_1_RECEIVE) &&
         PLIB_INT_SourceIsEnabled(INT_ID_0,
                             INT_SOURCE_USART_1_RECEIVE) ) {

        // Teste si erreur parité, framing ou overrun
        UsartStatus = PLIB_USART_ErrorsGet(USART_ID_1);

        //pas d'erreur ?
        if ( (UsartStatus & (USART_ERROR_PARITY |
                             USART_ERROR_FRAMING |
                    USART_ERROR_RECEIVER_OVERRUN)) == 0) {
            // transfert dans le FIFO software
            // de tous les char reçus
            while (PLIB_USART_ReceiverDataIsAvailable
                                                (USART_ID_1))
            {
                c = PLIB_USART_ReceiverByteReceive
                                                (USART_ID_1);
                PutCharInFifo ( &descrFifoRX, c);
            }
```

Voir note 1 ci-dessous

```
            LED4_W = !LED4_R; // Toggle Led4
            // buffer is empty, clear interrupt flag
            PLIB_INT_SourceFlagClear(INT_ID_0,
                           INT_SOURCE_USART_1_RECEIVE);
        } else {
            // Suppression des erreurs
            // La lecture des erreurs les efface
            // sauf pour overrun
              if ( (UsartStatus &
                       USART_ERROR_RECEIVER_OVERRUN) ==
                       USART_ERROR_RECEIVER_OVERRUN) {
                   PLIB_USART_ReceiverOverrunErrorClear
                                              (USART_ID_1);
              }
        }

         freeSize = GetWriteSpace ( &descrFifoRX);
         // A cause du cas un int pour 6 char ¾ de 8 = 6
         if (freeSize <= 6 )            {
            // Contrôle de flux : demande stop émission
            RS232_RTS = 1;
            }
        }
    } // end if RX

    // Is this an TX interrupt ?
    if ( PLIB_INT_SourceFlagGet(INT_ID_0,
                        INT_SOURCE_USART_1_TRANSMIT) &&
                 PLIB_INT_SourceIsEnabled(INT_ID_0,
                        INT_SOURCE_USART_1_TRANSMIT) ) {

         TXsize = GetReadSize (&descrFifoTX);
         // i_cts = input(RS232_CTS);
         // On vérifie 3 conditions :
         //    Si CTS = 0 (autorisation d'émettre)
         //    Si il y a un caractère à émettre
         //    Si le txreg est bien disponible
         i_cts = RS232_CTS;
         // Il est possible de déposer un caractère
         // tant que le tampon n'est pas plein
         TxBuffFull =
            PLIB_USART_TransmitterBufferIsFull(USART_ID_1);
         if ( (i_cts == 0) && ( TXsize > 0 ) &&
                                  TxBuffFull == false ) {

             do {
                 GetCharFromFifo(&descrFifoTX, &c);
                 PLIB_USART_TransmitterByteSend
                                          (USART_ID_1, c);
                 i_cts = RS232_CTS;
                 TXsize = GetReadSize (&descrFifoTX);
```

```
                    TxBuffFull =
                            PLIB_USART_TransmitterBufferIsFull
                                                (USART_ID_1);

                } while ( (i_cts == 0) && ( TXsize > 0 ) &&
                                        TxBuffFull==false );
            LED5_W = !LED5_R; // Toggle Led5
            // Clear the TX interrupt Flag
            // (Seulement après TX)
            PLIB_INT_SourceFlagClear(INT_ID_0,
                                INT_SOURCE_USART_1_TRANSMIT);
            if (TXsize == 0) {
                // disable TX interrupt
                // (pour éviter une int inutile)
                PLIB_INT_SourceDisable(INT_ID_0,
                                INT_SOURCE_USART_1_TRANSMIT);
            }
        } else {
           // disable TX interrupt
           PLIB_INT_SourceDisable(INT_ID_0,
                                INT_SOURCE_USART_1_TRANSMIT);
        }
    }

   // Marque fin interruption avec Led3
    LED3_W = 0;
} // end __ISR Usart 1
```

Note 1 :
Dans la partie réception de l'ISR, l'utilisation de la condition PLIB_USART_ReceiverDataIsAvailable() vérifie que des données reçues soient disponibles. Mais aucune vérification de la place disponible dans FIFO software n'est faite avant d'y copier ces données.
Cette manière de procéder peut être acceptable si, comme ici, un contrôle de flux est implémenté, ou si le FIFO est prévu avec suffisamment de place pour stocker le flux de données entrant dans le pire cas.

7.7.1. REMARQUE SUR LA RÉALISATION PRATIQUE

En testant le flag d'interruption et le flag d'autorisation, il est possible de déterminer la source de l'interruption (réception, émission ou erreur).

7.7.1.1. TRAITEMENT DE LA RECEPTION

Lors de la réception, on utilise une boucle pour vider le tampon dans le but de supporter les différents modes. Le contenu du tampon est transféré dans le FIFO de réception software.

Donc nous utilisons une boucle qui prend un caractère tant que disponible. Il faut ajouter le traitement d'une éventuelle erreur.

☝ Le flag d'interruption de réception ne doit être mis à zéro que lorsque le tampon de réception est vide.

7.7.1.2. TRAITEMENT DE L'EMISSION

Le traitement d'émission est un peu particulier car l'interruption de transmission se produit lorsqu'il est possible d'émettre. Il est donc nécessaire de bloquer cette interruption s'il n'y a plus rien à transmettre. C'est l'application qui est responsable de d'activer l'interruption lorsqu'il y a un message dans le FIFO software de transmission.

Réalisation d'une boucle qui essaie de remplir au maximum le tampon d'émission.

7.7.2. TEST DU MÉCANISME DE RÉCEPTION

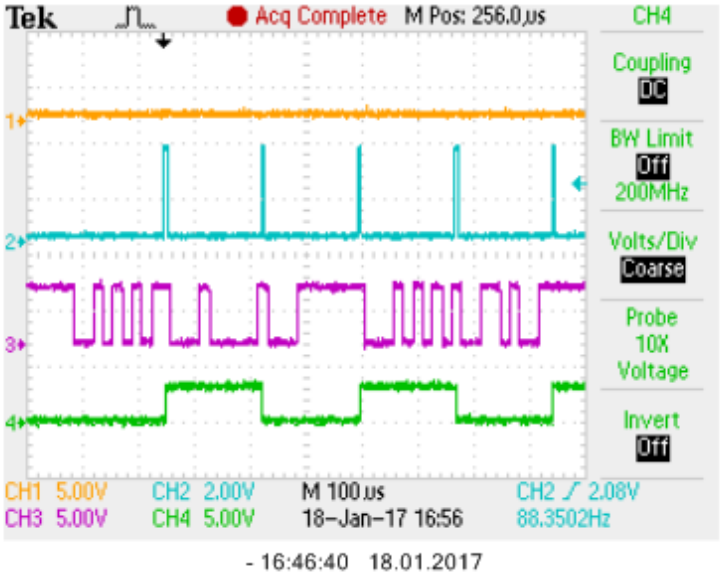
Voici l'observation du comportement de l'interruption de l'USART en relation avec le signal RX :

canal 1 : LED_1 (marque traitement dans l'application)

canal 2 : LED_3 (marque interruption)

canal 3 : broche 52 U1RX

canal 4: LED_6 (inversion dans boucle copie dans FIFO)



On observe qu'il y a 5 interruptions car le message est de 5 caractères. L'interruption se produit après la réception d'un caractère. Le canal 4 nous signale qu'il y a bien copie dans le FIFO à chaque interruption.

7.7.3. TEST DU MÉCANISME D'ÉMISSION

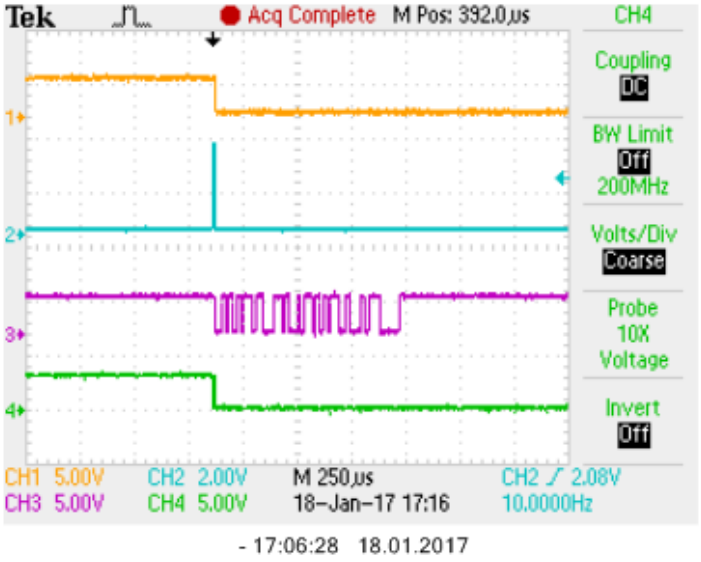
Voici l'observation du comportement de l'interruption de l'USART en relation avec le signal TX :

canal 1 : LED_1 (marque traitement dans l'application)

canal 2 : LED_3 (marque interruption)

canal 3 : broche 53 U1TX

canal 4: LED_7 (inversion dans boucle écriture dans tampon émission)



On constate que l'interruption (unique) a lieu à la fin du traitement de l'application (action SendMessage)

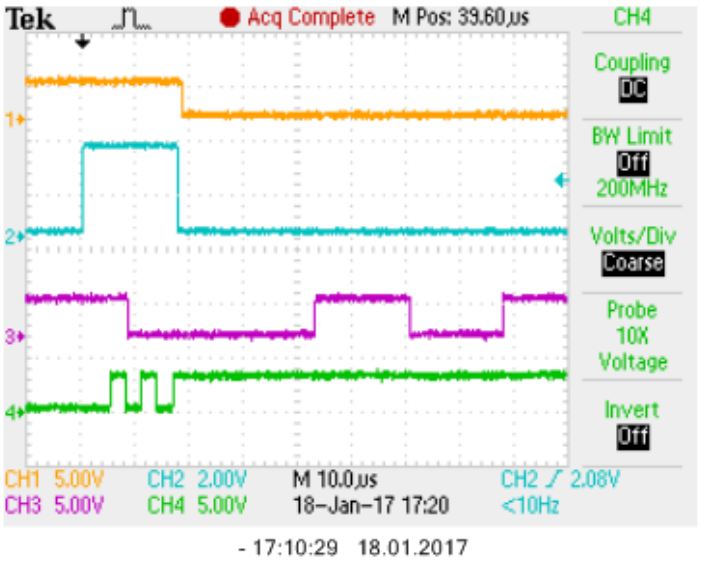
En changeant l'échelle, on peut observer la copie des 5 caractères du message dans le tampon d'émission (5 flancs sur le signal).

canal 1 : LED_1 (marque traitement dans l'application)

canal 2 : LED_3 (marque interruption)

canal 3 : broche 53 U1TX

canal 4: LED_7 (inversion dans boucle écriture dans tampon émission)



On voit ici l'efficacité du mécanisme d'émission : Avec une seule interruption on remplit le tampon d'émission et les 5 caractères sont émis par l'USART.

7.8. HISTORIQUE DES VERSIONS

7.8.1. V1.0 AVRIL 2014

Création du document après une laborieuse mise au point de l'application utilisée pour tester.

7.8.2. V1.5 JANVIER 2015

Adaptation du document aux nouvelles PLIB_USART et PLIB_INT introduite avec Harmony 1.00.

7.8.3. V1.6 JANVIER 2016

Adaptation du document aux détails du code généré lié au MPLABX 3.10 et Harmony 1.06.

7.8.4. V1.7 JANVIER 2017

Adaptation du document aux détails du code généré lié au MPLABX 3.40 et Harmony 1.08. Remarque : forte évolution du driver USART.

7.8.1. V1.8 NOVEMBRE 2017

Reprise et relecture par SCA.

7.8.2. V1.81 JANVIER 2019

Mise à jour figure configuration USART via MHC.

7.8.3. V1.82 NOVEMBRE 2021

Précision code ISR réception.