```c
/*******************************************************************************
 System Interrupts File

  File Name:
    system_interrupt.c

  Summary:
    Raw ISR definitions.

  Description:
    This file contains a definitions of the raw ISRs required to support the
    interrupt sub-system.

  Summary:
    This file contains source code for the interrupt vector functions in the
    system.

  Description:
    This file contains source code for the interrupt vector functions in the
    system.  It implements the system and part specific vector "stub" functions
    from which the individual "Tasks" functions are called for any modules
    executing interrupt-driven in the MPLAB Harmony system.

  Remarks:
    This file requires access to the systemObjects global data structure that
    contains the object handles to all MPLAB Harmony module objects executing
    interrupt-driven in the system.  These handles are passed into the individual
    module "Tasks" functions to identify the instance of the module to maintain.
 *******************************************************************************/

// DOM-IGNORE-BEGIN
/*******************************************************************************
Copyright (c) 2011-2014 released Microchip Technology Inc.  All rights reserved.

Microchip licenses to you the right to use, modify, copy and distribute
Software only when embedded on a Microchip microcontroller or digital signal
controller that is integrated into your product or third party product
(pursuant to the sublicense terms in the accompanying license agreement).

You should refer to the license agreement accompanying this Software for
additional information regarding your rights and obligations.

SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF
MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.
IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER
CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR
OTHER LEGAL EQUITABLE THEORY ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES
INCLUDING BUT NOT LIMITED TO ANY INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR
CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF
SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES
(INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
 *******************************************************************************/
// DOM-IGNORE-END

// *****************************************************************************
// *****************************************************************************
// Section: Included Files
// *****************************************************************************
// *****************************************************************************

#include "system/common/sys_common.h"
#include "app.h"
#include "system_definitions.h"

// *****************************************************************************
// *****************************************************************************
// Section: System Interrupt Vector Functions
// *****************************************************************************
// *****************************************************************************
void __ISR(_UART_1_VECTOR, ipl7AUTO) _IntHandlerDrvUsartInstance0(void)
{

    USART_ERROR usartStatus;
    bool        isTxBuffFull;
    static bool        isStatusBeg = false;
    int8_t      charReceived;
    int8_t      charToSend;
    int8_t      charTrash;
    int8_t      TXsize;

    //------------------------------------------------------------------------// 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)){

        // Parity error or overrun
        usartStatus = PLIB_USART_ErrorsGet(USART_ID_1);

        if ((usartStatus & (USART_ERROR_PARITY | USART_ERROR_FRAMING |
                USART_ERROR_RECEIVER_OVERRUN)) == 0){

            // All char received are transferred to the FIFO
            // 1 if ONE_CHAR, 4 if HALF_FULL and 6 3B4FULL
            while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_1)){

                charReceived = PLIB_USART_ReceiverByteReceive(USART_ID_1);

                putCharInFifo(&usartFifoRx, charReceived);

                // Beginning of a status
                if(charReceived == '<' && appData.isBluetoothInCommandMode == false) isStatusBeg = true;

                // Ending of a status
                if(appData.isBluethoothModuleInit && charReceived == '>' &&
                        appData.isBluetoothInCommandMode == false && isStatusBeg == true){

                    isStatusBeg = false;
                    USART1_Callback_Function();
                }

                //
                if(isStatusBeg == false && appData.isBluetoothInCommandMode ==
                        false) getCharFromFifo(&usartFifoRx, &charTrash);
            }
            // Buffer is empty, clear interrupt flag
            PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_RECEIVE);

        }else{
            // Deleting errors
            // Reading errors clears them except for overrun
            if((usartStatus & USART_ERROR_RECEIVER_OVERRUN) ==
                    USART_ERROR_RECEIVER_OVERRUN){

                PLIB_USART_ReceiverOverrunErrorClear(USART_ID_1);
            }
        }
    }


    //------------------------------------------------------------------------// 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(&usartFifoTx);
        // i_cts = input(RS232_CTS);

        isTxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_1);

        if (/*(i_cts == 0) && */(TXsize > 0) && (isTxBuffFull == false)){
            do{
                getCharFromFifo(&usartFifoTx, &charToSend);
                if(charToSend != '\0') PLIB_USART_TransmitterByteSend(USART_ID_1, charToSend);
                /*i_cts = RS232_CTS;*/
                TXsize = getReadSize (&usartFifoTx);
                isTxBuffFull = PLIB_USART_TransmitterBufferIsFull(USART_ID_1);

            }while(/*(i_cts == 0) && */( TXsize > 0 ) && isTxBuffFull == false);
        }

        // Disables TX interrupt (to avoid unnecessary interruptions if there's
        // nothing left to transmit)
        if(TXsize == 0){

            PLIB_INT_SourceDisable(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
        }
        // Clears the TX interrupt Flag
        PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_USART_1_TRANSMIT);
    }
}


////-------------------------------------------------------------------------// TIMER0 ID1 <--- Disabled
//void __ISR(_TIMER_1_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance0(void){
//
//    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_1);
//    TIMER0_Callback_Function();
//}


//-------------------------------------------------------------------------// TIMER1 ID2
void __ISR(_TIMER_2_VECTOR, ipl2AUTO) IntHandlerDrvTmrInstance1(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_2);
    TIMER1_Callback_Function();
}


//-------------------------------------------------------------------------// TIMER2 ID5
void __ISR(_TIMER_5_VECTOR, ipl0AUTO) IntHandlerDrvTmrInstance2(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_5);
}


//-------------------------------------------------------------------------// TIMER3 ID3
void __ISR(_TIMER_3_VECTOR, ipl0AUTO) IntHandlerDrvTmrInstance3(void)
{
    PLIB_INT_SourceFlagClear(INT_ID_0,INT_SOURCE_TIMER_3);
}


 /*******************************************************************************
 End of File
*/
```