```
//###########################################################################
//
// FILE:    Example_2833xSci_Echoback.c
//
// TITLE:   DSP2833x Device SCI Echoback.

// DESCRIPTION:
//
//
//    This test recieves and echo-backs data through the SCI-A port.
//
//    1) Configure hyperterminal:
//       Use the included hyperterminal configuration file SCI_96.ht.
//       To load this configuration in hyperterminal: file->open
//       and then select the SCI_96.ht file.
//    2) Check the COM port.
//       The configuration file is currently setup for COM1.
//       If this is not correct, disconnect Call->Disconnect
//       Open the File-Properties dialog and select the correct COM port.
//    3) Connect hyperterminal Call->Call
//       and then start the 2833x SCI echoback program execution.
//    4) The program will print out a greeting and then ask you to
//       enter a character which it will echo back to hyperterminal.
//
//    As is, the program configures SCI-A for 9600 baud with
//    SYSCLKOUT = 150MHz and LSPCLK = 37.5 MHz
//    SYSCLKOUT = 100MHz and LSPCLK = 25.0 Mhz
//
//
//    Watch Variables:
//       LoopCount for the number of characters sent
//       ErrorCount
//
//
//###########################################################################
// $TI Release: 2833x/2823x Header Files and Peripheral Examples V133 $
// $Release Date: June 8, 2012 $
//###########################################################################

#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File

// Prototype statements for functions found within this file.
void scia_echoback_init(void);
void scia_fifo_init(void);
void scia_xmit(int a);
void scia_msg(char *msg);

// Global counts used in this example
Uint16 LoopCount;
Uint16 ErrorCount;

void main(void)
{

    Uint16 ReceivedChar;
    char *msg;
```

```
// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP2833x_SysCtrl.c file.
   InitSysCtrl();

// Step 2. Initalize GPIO:
// This example function is found in the DSP2833x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
   // InitGpio(); Skipped for this example

// For this example, only init the pins for the SCI-A port.
// This function is found in the DSP2833x_Sci.c file.
   InitSciaGpio();

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;

// Initialize PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
   InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
   IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in DSP2833x_DefaultIsr.c.
// This function is found in DSP2833x_PieVect.c.
   InitPieVectTable();

// Step 4. Initialize all the Device Peripherals:
// This function is found in DSP2833x_InitPeripherals.c
// InitPeripherals(); // Not required for this example

// Step 5. User specific code:

    LoopCount = 0;
    ErrorCount = 0;

    scia_fifo_init();      // Initialize the SCI FIFO
    scia_echoback_init();  // Initalize SCI for echoback

    msg = "\r\n\n\nHello World!\0";
    scia_msg(msg);

    msg = "\r\nYou will enter a character, and the DSP will echo it back! \n\0";
    scia_msg(msg);

    for(;;)
    {
       msg = "\r\nEnter a character: \0";
```

```
        scia_msg(msg);

        // Wait for inc character
        while(SciaRegs.SCIFFRX.bit.RXFFST !=1) { } // wait for XRDY =1 for empty state

        // Get character
        ReceivedChar = SciaRegs.SCIRXBUF.all;

        // Echo character back
        msg = "  You sent: \0";
        scia_msg(msg);
        scia_xmit(ReceivedChar);

        LoopCount++;
    }

}


// Test 1,SCIA  DLB, 8-bit word, baud rate 0x000F, default, 1 STOP bit, no parity
void scia_echoback_init()
{
    // Note: Clocks were turned on to the SCIA peripheral
    // in the InitSysCtrl() function

    SciaRegs.SCICCR.all =0x0007;   // 1 stop bit,  No loopback
                                   // No parity,8 char bits,
                                   // async mode, idle-line protocol
    SciaRegs.SCICTL1.all =0x0003;  // enable TX, RX, internal SCICLK,
                                   // Disable RX ERR, SLEEP, TXWAKE
    SciaRegs.SCICTL2.all =0x0003;
    SciaRegs.SCICTL2.bit.TXINTENA =1;
    SciaRegs.SCICTL2.bit.RXBKINTENA =1;
    #if (CPU_FRQ_150MHZ)
          SciaRegs.SCIHBAUD    =0x0001;  // 9600 baud @LSPCLK = 37.5MHz.
          SciaRegs.SCILBAUD    =0x00E7;
    #endif
    #if (CPU_FRQ_100MHZ)
      SciaRegs.SCIHBAUD    =0x0001;  // 9600 baud @LSPCLK = 20MHz.
      SciaRegs.SCILBAUD    =0x0044;
    #endif
    SciaRegs.SCICTL1.all =0x0023;  // Relinquish SCI from Reset
}

// Transmit a character from the SCI
void scia_xmit(int a)
{
    while (SciaRegs.SCIFFTX.bit.TXFFST != 0) {}
    SciaRegs.SCITXBUF=a;

}

void scia_msg(char * msg)
{
    int i;
    i = 0;
    while(msg[i] != '\0')
```

```
    {
        scia_xmit(msg[i]);
        i++;
    }
}

// Initalize the SCI FIFO
void scia_fifo_init()
{
    SciaRegs.SCIFFTX.all=0xE040;
    SciaRegs.SCIFFRX.all=0x204f;
    SciaRegs.SCIFFCT.all=0x0;

}




//===========================================================================
// No more.
//===========================================================================
```