Unit / Module Description:	RS485 on the SMT148FX
Unit / Module Number:	SMT148FX
Document Issue Number:	1.0
Issue Date:	08/11/2010
Original Author:	F.S

Application Note for SMT148FX

RS485 Example

Sundance Multiprocessor Technology Ltd, Chiltern House, Waterside, Chesham, Bucks. HP5 1PS.

Revision History

Issue	Changes Made	Date	Initials
1.0	First release	08/11/10	FS

Table of Contents

Table of Figures

1 Introduction

This document explains one way to implement the RS485 on the SMT148FX. It is based on an EDK project with a MicroBlaze processor that sends and receives data through the RS485.

On the SMT148FX, each of the 16 RS485 signal pairs is driven by an SN75HVD12. They are arranged into two groups of 8-bits each and have a single control signal which selects the group to be a transmitter or receiver.

Connector RS1 carries these signals.

Function	pin	pin	Function
S0-	1	2	S0+
S1-	3	4	S1+
S2-	5	6	S2+
S3-	7	8	S3+
S4-	9	10	S4+
S5-	11	12	S5+
S6-	13	14	S6+
S7-	15	16	S7+
GND	17	18	GND
GND	19	20	GND

Function	pin	pin	Function
GND	21	22	GND
GND	23	24	GND
S8-	25	26	S8+
S9-	27	28	S9+
S10-	29	30	S10+
S11-	31	32	S11+
S12-	33	34	S12+
S13-	35	36	S13+
S14-	37	38	S14+
S15-	39	40	S15+

Figure 1 : RS485 Connector (RS1)

2 RS485 Example

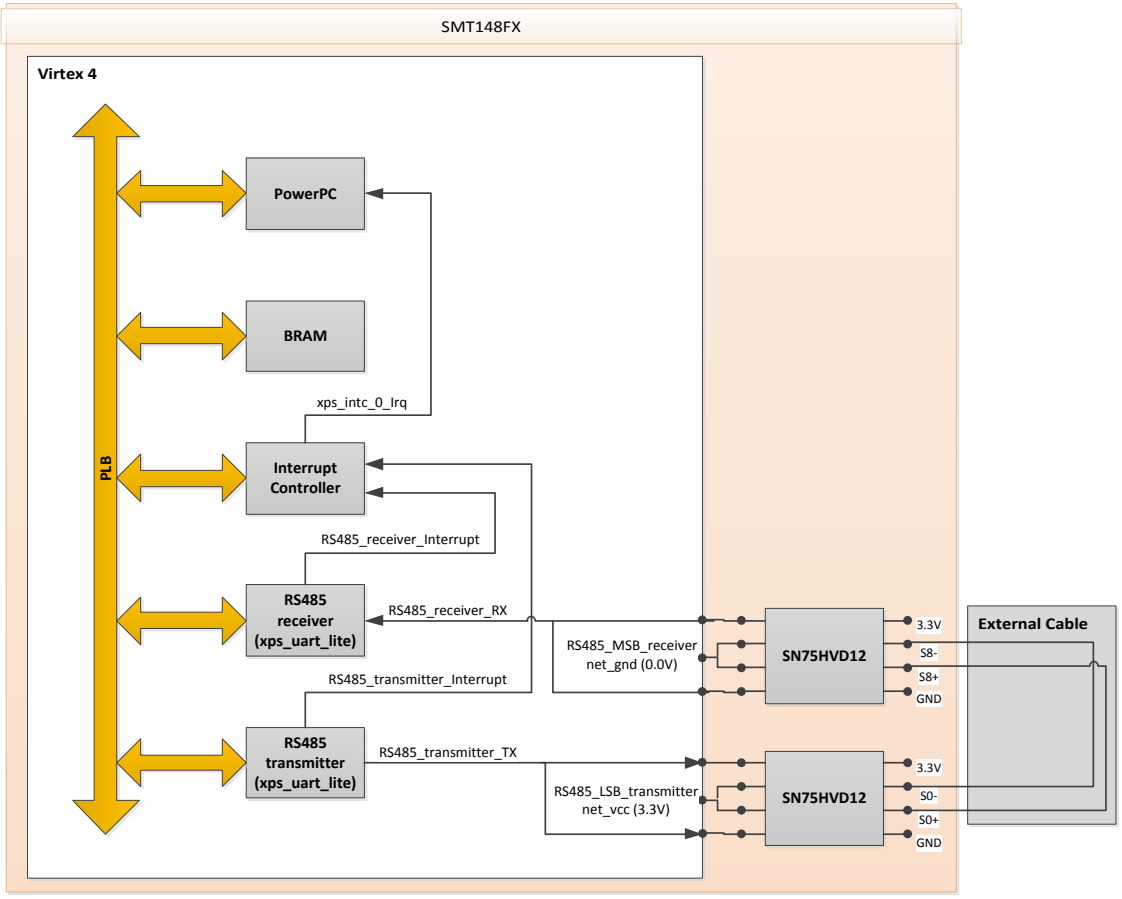

Figure 2 : RS485 Example Diagram

For this example the RS485 LSB signals are set as transmitter with the SN75HVD12 enable pins connected to VCC and the MSB are set as receiver. Only the signal pairs S0 and S8 will be driven, it provides a 4 wires full duplex RS485.

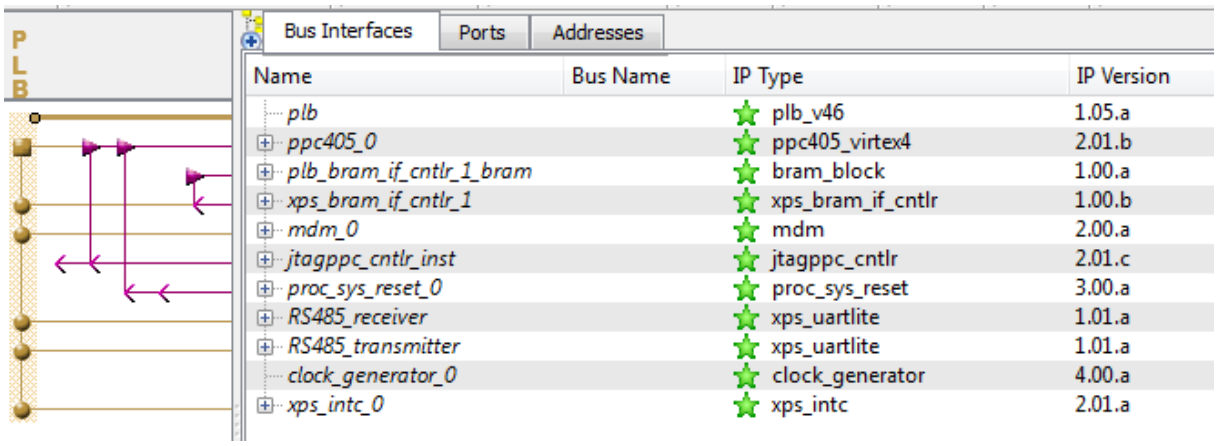

Figure 3 : EDK project bus interfaces

The example is using two uartlite IP from the Xilinx library to transmit and receive data. For the transmitter the TX pin is set as external and the RX pin for the receiver.

Two external pins need to be added to set the LSB and MSB SN75HVD12 control signals, to receive or transmit.

Use the following MHS file and UCF file in your EDK project, those file have been generated for EDK12.3, you might need minor modifications for other versions.

2.1 EDK project: MHS file

```
# ##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 12.3 Build EDK_MS3.70d
# Mon Nov 08 11:25:12 2010
# Target Board:  Custom
# Family:    virtex4
# Device:    xc4vfx60
# Package:   ff1152
# Speed Grade:  -10
# Processor number: 1
# Processor 1: ppc405_0
# Processor clock frequency: 300.0
# Bus clock frequency: 100.0
# Debug Interface: FPGA JTAG
# ##############################################################################
 PARAMETER VERSION = 2.1.0

 PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 50000000
 PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 0
 PORT RS485_transmitter_TX_pin = RS485_transmitter_TX, DIR = O
 PORT RS485_receiver_RX_pin = RS485_receiver_RX, DIR = I
 PORT RS485_LSB_transmitter = net_vcc, DIR = O
 PORT RS485_MSB_receiver = net_gnd, DIR = O

BEGIN ppc405_virtex4
 PARAMETER INSTANCE = ppc405_0
 PARAMETER C_FASTEST_PLB_CLOCK = DPLB0
 PARAMETER C_IDCR_BASEADDR = 0b0100000000
 PARAMETER C_IDCR_HIGHADDR = 0b0111111111
 PARAMETER HW_VER = 2.01.b
 BUS_INTERFACE DPLB0 = plb
 BUS_INTERFACE IPLB0 = plb
 BUS_INTERFACE JTAGPPC = ppc405_0_jtagppc_bus
 BUS_INTERFACE RESETPPC = ppc_reset_bus
 PORT CPMC405CLOCK = clk_300_0000MHzDCM0
 PORT EICC405EXTINPUTIRQ = xps_intc_0_Irq
END

BEGIN plb_v46
 PARAMETER INSTANCE = plb
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_NUM_CLK_PLB2OPB_REARB = 100
 PARAMETER HW_VER = 1.05.a
 PORT PLB_Clk = clk_100_0000MHzDCM0
 PORT SYS_Rst = sys_bus_reset
END
```

```
BEGIN xps_bram_if_cntlr
 PARAMETER INSTANCE = xps_bram_if_cntlr_1
 PARAMETER C_SPLB_NATIVE_DWIDTH = 64
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_BASEADDR = 0xffff0000
 PARAMETER C_HIGHADDR = 0xffffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_1_port
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_1_port
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = RS485_transmitter
 PARAMETER C_BAUDRATE = 115200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = plb
 PORT TX = RS485_transmitter_TX
 PORT Interrupt = RS485_transmitter_Interrupt
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = RS485_receiver
 PARAMETER C_BAUDRATE = 115200
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_ODD_PARITY = 0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x84020000
 PARAMETER C_HIGHADDR = 0x8402ffff
 BUS_INTERFACE SPLB = plb
 PORT RX = RS485_receiver_RX
 PORT Interrupt = RS485_receiver_Interrupt
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = plb
 PORT Irq = xps_intc_0_Irq
 PORT Intr = RS485_receiver_Interrupt & RS485_transmitter_Interrupt
END

BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
```

```
 PARAMETER C_CLKIN_FREQ = 50000000
 PARAMETER C_CLKOUT0_FREQ = 100000000
 PARAMETER C_CLKOUT0_PHASE = 0
 PARAMETER C_CLKOUT0_GROUP = DCM0
 PARAMETER C_CLKOUT0_BUF = TRUE
 PARAMETER C_CLKOUT1_FREQ = 125000000
 PARAMETER C_CLKOUT1_PHASE = 0
 PARAMETER C_CLKOUT1_GROUP = NONE
 PARAMETER C_CLKOUT1_BUF = TRUE
 PARAMETER C_CLKOUT2_FREQ = 200000000
 PARAMETER C_CLKOUT2_PHASE = 0
 PARAMETER C_CLKOUT2_GROUP = NONE
 PARAMETER C_CLKOUT2_BUF = TRUE
 PARAMETER C_CLKOUT3_FREQ = 300000000
 PARAMETER C_CLKOUT3_PHASE = 0
 PARAMETER C_CLKOUT3_GROUP = DCM0
 PARAMETER C_CLKOUT3_BUF = TRUE
 PARAMETER C_EXT_RESET_HIGH = 0
 PARAMETER HW_VER = 4.00.a
 PORT CLKIN = dcm_clk_s
 PORT CLKOUT0 = clk_100_0000MHzDCM0
 PORT CLKOUT3 = clk_300_0000MHzDCM0
 PORT RST = sys_rst_s
 PORT LOCKED = Dcm_all_locked
END

BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_cntlr_inst
 PARAMETER HW_VER = 2.01.c
 BUS_INTERFACE JTAGPPC0 = ppc405_0_jtagppc_bus
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER C_EXT_RESET_HIGH = 0
 PARAMETER HW_VER = 3.00.a
 BUS_INTERFACE RESETPPC0 = ppc_reset_bus
 PORT Slowest_sync_clk = clk_100_0000MHzDCM0
 PORT Ext_Reset_In = sys_rst_s
 PORT Dcm_locked = Dcm_all_locked
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Peripheral_Reset = sys_periph_reset
 PORT MB_Debug_Sys_Rst = mdm_0_Debug_SYS_Reset
END

BEGIN mdm
 PARAMETER INSTANCE = mdm_0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_UART = 1
 PARAMETER C_INTERCONNECT = 1
 PARAMETER C_MB_DBG_PORTS = 0
 PARAMETER C_BASEADDR = 0x84400000
 PARAMETER C_HIGHADDR = 0x8440ffff
 BUS_INTERFACE SPLB = plb
 PORT Debug_SYS_Rst = mdm_0_Debug_SYS_Reset
END
```

2.2 EDK project: UCF file

```
#RS232 pins, mdm uart is used in this example
#Net fpga_0_RS232_RX_pin LOC="L21" | IOSTANDARD = LVTTL;
#Net fpga_0_RS232_TX_pin LOC="M22" | IOSTANDARD = LVTTL;

Net RS485_LSB_transmitter     LOC="AJ16" | IOSTANDARD = LVTTL; #DIRLSB, LSB RS485 signal pairs are transmitter (DIRLSB connected to net_vcc here)
Net RS485_transmitter_TX_pin LOC="AK31" | IOSTANDARD = LVTTL; #RS485D0

Net RS485_MSB_receiver    LOC="AK16" | IOSTANDARD = LVTTL; #DIRMSB, MSB RS485 signal pairs are receiver (DIRMSB connected to net_gnd here)
Net RS485_receiver_RX_pin LOC="AL29" | IOSTANDARD = LVTTL;              #RS485D8

Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 50000 kHz;
Net fpga_0_clk_1_sys_clk_pin LOC= "L15"| IOSTANDARD=LVTTL;              # BOARD CLOCK
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC= "M23" | IOSTANDARD=LVTTL;            # BOARD RESET Active Low

###### ppc405_0
NET "ppc405_0/C405RSTCHIPRESETREQ" TPTHRU = "ppc405_0_RST_GRP";
NET "ppc405_0/C405RSTCORERESETREQ" TPTHRU = "ppc405_0_RST_GRP";
NET "ppc405_0/C405RSTSYSRESETREQ" TPTHRU = "ppc405_0_RST_GRP";
TIMESPEC "TS_RST_ppc405_0" = FROM CPUS THRU ppc405_0_RST_GRP TO FFS TIG;
```

2.3 EDK project: RS485_example.c

```
/***************************** Include Files *********************************/

#include "xparameters.h"
#include "xuartlite.h"
#include "xintc.h"
#include "xil_exception.h"
#include <stdio.h>

/************************** Constant Definitions *****************************/

/*
 * The following constants map to the XPAR parameters created in the
 * xparameters.h file. They are defined here such that a user can easily
 * change all the needed parameters in one place.
 */
#define RS485_TRANSMITTER_DEVICE_ID      XPAR_RS485_TRANSMITTER_DEVICE_ID
#define RS485_RECEIVER_DEVICE_ID          XPAR_RS485_RECEIVER_DEVICE_ID
#define INTC_DEVICE_ID                    XPAR_INTC_0_DEVICE_ID
#define RS485_TRANSMITTER_INT_IRQ_ID     XPAR_XPS_INTC_0_RS485_TRANSMITTER_INTERRUPT_INTR
#define RS485_RECEIVER_INT_IRQ_ID         XPAR_XPS_INTC_0_RS485_RECEIVER_INTERRUPT_INTR

/*
 * The following constant controls the length of the buffers to be sent
 * and received with the UartLite device.
 */
#define TEST_BUFFER_SIZE        100
```

```
/************************** Type Definitions *****************************/

/***************** Macros (Inline Functions) Definitions *********************/

/************************** Function Prototypes ******************************/

int UartLiteIntrExample();

int SetupInterruptSystem(XUartLite *TxPtr, XUartLite *RxPtr);

void SendHandler(void *CallBackRef, unsigned int EventData);

void RecvHandler(void *CallBackRef, unsigned int EventData);

/************************** Variable Definitions *****************************/

XUartLite RS485Transmitter;            /* The instance of the UartLite Device */
XUartLite RS485Receiver;           /* The instance of the UartLite Device */

 XIntc InterruptController;     /* The instance of the Interrupt Controller */

/*
 * The following variables are shared between non-interrupt processing and
 * interrupt processing such that they must be global.
 */

/*
 * The following buffers are used in this example to send and receive data
 * with the UartLite.
 */
u8 SendBuffer[TEST_BUFFER_SIZE];
u8 ReceiveBuffer[TEST_BUFFER_SIZE];

/*
 * The following counters are used to determine when the entire buffer has
 * been sent and received.
 */
static volatile int TotalReceivedCount;
static volatile int TotalSentCount;


/*****************************************************************************/
/**
*
* Main function to call the UartLite interrupt example.
*
* @param        None
*
* @return       XST_SUCCESS if successful, XST_FAILURE if unsuccessful
*
* @note         None
*
******************************************************************************/
int main(void)
```

```
{
	int Status;

	print("---Entering main---\r\n");
	/*
	 * Run the UartLite Interrupt example, specify the Device ID that is
	 * generated in xparameters.h.
	 */
	Status = UartLiteIntrExample();
	if (Status != XST_SUCCESS) {
		print("---Example Failed---\r\n");
		return XST_FAILURE;
	}

	print("---Example completed---\r\n");
	return XST_SUCCESS;
}

/*****************************************************************************/
/**
*
* This function does a minimal test on the UartLite device and driver as a
* design example. The purpose of this function is to illustrate
* how to use the XUartLite component.
*
* This function sends data and expects to receive the same data through the
* UartLite. The user must provide a physical loopback such that data which is
* transmitted will be received.
*
* This function uses interrupt driver mode of the UartLite device. The calls
* to the UartLite driver in the handlers should only use the non-blocking
* calls.
*
* @param	DeviceId is the Device ID of the UartLite Device and is the
*		XPAR_<uartlite_instance>_DEVICE_ID value from xparameters.h.
*
* @return	XST_SUCCESS if successful, otherwise XST_FAILURE.
*
* @note
*
* This function contains an infinite loop such that if interrupts are not
* working it may never return.
*
******************************************************************************/
int UartLiteIntrExample()
{
	int Status;
	int Index;

	/*
	 * Initialize the UartLite driver so that it's ready to use.
	 */
	Status = XUartLite_Initialize(&RS485Transmitter, XPAR_RS485_TRANSMITTER_DEVICE_ID);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}
```

```
Status = XUartLite_Initialize(&RS485Receiver, RS485_RECEIVER_DEVICE_ID);
if (Status != XST_SUCCESS) {
		return XST_FAILURE;
}

/*
 * Perform a self-test to ensure that the hardware was built correctly.
 */
Status = XUartLite_SelfTest(&RS485Transmitter);
if (Status != XST_SUCCESS) {
		return XST_FAILURE;
}
Status = XUartLite_SelfTest(&RS485Receiver);
if (Status != XST_SUCCESS) {
		return XST_FAILURE;
}

/*
 * Connect the UartLite to the interrupt subsystem such that interrupts can
 * occur. This function is application specific.
 */
Status = SetupInterruptSystem(&RS485Transmitter, &RS485Receiver);
if (Status != XST_SUCCESS) {
		return XST_FAILURE;
}

/*
 * Setup the handlers for the UartLite that will be called from the
 * interrupt context when data has been sent and received, specify a
 * pointer to the UartLite driver instance as the callback reference so
 * that the handlers are able to access the instance data.
 */
XUartLite_SetSendHandler(&RS485Transmitter, SendHandler, &RS485Transmitter);
XUartLite_SetRecvHandler(&RS485Receiver, RecvHandler, &RS485Receiver);

/*
 * Enable the interrupt of the UartLite so that interrupts will occur.
 */
XUartLite_EnableInterrupt(&RS485Transmitter);
XUartLite_EnableInterrupt(&RS485Receiver);

/*
 * Initialize the send buffer bytes with a pattern to send and the
 * the receive buffer bytes to zero to allow the receive data to be
 * verified.
 */
for (Index = 0; Index < TEST_BUFFER_SIZE; Index++) {
		SendBuffer[Index] = Index;
		ReceiveBuffer[Index] = 0;
}

/*
 * Start receiving data before sending it since there is a loopback.
 */
XUartLite_Recv(&RS485Receiver, ReceiveBuffer, TEST_BUFFER_SIZE);
```

```
	/*
	 * Send the buffer using the UartLite.
	 */
	XUartLite_Send(&RS485Transmitter, SendBuffer, TEST_BUFFER_SIZE);

	/*
	 * Wait for the entire buffer to be received, letting the interrupt
	 * processing work in the background, this function may get locked
	 * up in this loop if the interrupts are not working correctly.
	 */
	while ((TotalReceivedCount != TEST_BUFFER_SIZE) ||
			(TotalSentCount != TEST_BUFFER_SIZE)) {
	}

	/*
	 * Verify the entire receive buffer was successfully received.
	 */
	for (Index = 0; Index < TEST_BUFFER_SIZE; Index++) {
			if (ReceiveBuffer[Index] != SendBuffer[Index]) {
				return XST_FAILURE;
			}
	}

	return XST_SUCCESS;
}

/*****************************************************************************/
/**
*
* This function is the handler which performs processing to send data to the
* UartLite. It is called from an interrupt context such that the amount of
* processing performed should be minimized. It is called when the transmit
* FIFO of the UartLite is empty and more data can be sent through the UartLite.
*
* This handler provides an example of how to handle data for the UartLite,
* but is application specific.
*
* @param	CallBackRef contains a callback reference from the driver.
*		In this case it is the instance pointer for the UartLite driver.
* @param	EventData contains the number of bytes sent or received for sent
*		and receive events.
*
* @return	None.
*
* @note		None.
*
*****************************************************************************/
void SendHandler(void *CallBackRef, unsigned int EventData)
{
	TotalSentCount = EventData;
}

/*****************************************************************************/
/**
*
* This function is the handler which performs processing to receive data from
```

```
* the UartLite. It is called from an interrupt context such that the amount of
* processing performed should be minimized.  It is called data is present in
* the receive FIFO of the UartLite such that the data can be retrieved from
* the UartLite. The size of the data present in the FIFO is not known when
* this function is called.
*
* This handler provides an example of how to handle data for the UartLite,
* but is application specific.
*
* @param        CallBackRef contains a callback reference from the driver, in
*               this case it is the instance pointer for the UartLite driver.
* @param        EventData contains the number of bytes sent or received for sent
*               and receive events.
*
* @return       None.
*
* @note         None.
*
******************************************************************************/
void RecvHandler(void *CallBackRef, unsigned int EventData)
{
        TotalReceivedCount = EventData;
}

/*****************************************************************************/
/**
*
* This function setups the interrupt system such that interrupts can occur
* for the UartLite device. This function is application specific since the
* actual system may or may not have an interrupt controller. The UartLite
* could be directly connected to a processor without an interrupt controller.
* The user should modify this function to fit the application.
*
* @param    UartLitePtr contains a pointer to the instance of the UartLite
*       component which is going to be connected to the interrupt
*       controller.
*
* @return   XST_SUCCESS if successful, otherwise XST_FAILURE.
*
* @note     None.
*
******************************************************************************/
int SetupInterruptSystem(XUartLite *TxPtr, XUartLite *RxPtr)
{

        int Status;


        /*
         * Initialize the interrupt controller driver so that it is ready to
         * use.
         */
        Status = XIntc_Initialize(&InterruptController, INTC_DEVICE_ID);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
```

```
	/*
	 * Connect a device driver handler that will be called when an interrupt
	 * for the device occurs, the device driver handler performs the
	 * specific interrupt processing for the device.
	 */
	Status = XIntc_Connect(&InterruptController, RS485_TRANSMITTER_INT_IRQ_ID,
				(XInterruptHandler)XUartLite_InterruptHandler,
				(void *)TxPtr);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}
	Status = XIntc_Connect(&InterruptController, RS485_RECEIVER_INT_IRQ_ID,
				(XInterruptHandler)XUartLite_InterruptHandler,
				(void *)RxPtr);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	/*
	 * Start the interrupt controller such that interrupts are enabled for
	 * all devices that cause interrupts, specific real mode so that
	 * the UartLite can cause interrupts through the interrupt controller.
	 */
	Status = XIntc_Start(&InterruptController, XIN_REAL_MODE);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	/*
	 * Enable the interrupt for the UartLite device.
	 */
	XIntc_Enable(&InterruptController, RS485_TRANSMITTER_INT_IRQ_ID);
	XIntc_Enable(&InterruptController, RS485_RECEIVER_INT_IRQ_ID);

	/*
	 * Initialize the exception table.
	 */
	Xil_ExceptionInit();

	/*
	 * Register the interrupt controller handler with the exception table.
	 */
	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
				(Xil_ExceptionHandler)XIntc_InterruptHandler,
				&InterruptController);

	/*
	 * Enable exceptions.
	 */
	Xil_ExceptionEnable();

	return XST_SUCCESS;
}
```

This software example is using the interrupt signal generated by the uart to check if the data are transmitted and received. It sends a buffer and waits to receive the data back; a loop-back cable is needed as shown on figure 2. The data are verified, and if the example terminates with no error "---Example completed---"should be display.

The project is displaying the printf through the mdm (set in the mhs file). In an xmd console, when you have loaded the virtex4 bitstream, write:

> connect mdm –uart

> terminal –jtag_uart_server

Then you can run your program and you should have:

XMD% connect mdm -uart

JTAG chain configuration

Device	*ID Code*	*IR Length*	*Part Name*
1	*81eb4093*	*14*	*XC4VFX60*
2	*16d7e093*	*8*	*XC2C512*
3	*01434093*	*6*	*XC3S1500*

Connected to MDM UART Target
XMD% terminal -jtag_uart_server
JTAG-based Terminal Server.
(TCP Port no used is 4321)

4321
XMD% ---Entering main---
---Example completed---

If you want to use the RS232 on the board, add another uartlite IP, the pins locations are commented in the UCF file.