XAPP1054 (v1.0) April 25, 2008

Reference System: MOST NIC Using the XA Automotive ECU Development Kit

Abstract

This application note describes a reference system that tests the operation of the Xilinx Platform Studio (XPS) Media Oriented System Transport (MOST) Network Interface Controller (NIC) cores that are connected to a fiber optic ring on a MOST connectivity daughter card of the XA3S1600E board. The reference system contains two XPS MOST NIC cores along with other peripherals. This document describes how to configure a system containing two XPS MOST NIC cores, the port connections between different cores, and the clocking for the XPS MOST NIC and other XPS peripherals. A basic description of the software application with the reference system is also provided.

This reference system is targeted for the Xilinx XA3S1600E board.

Included Systems

The reference system for the Xilinx XA3S1600E Evaluation Board is included with this application note. The reference system is available at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=109527

Introduction

MOST is a growing standard for automotive multimedia networks that is a low-cost, fiber-optic based protocol that provides connectivity for passenger infotainment networks. The XPS MOST NIC core, in conjunction with the Xilinx Automotive (XA) solution and embedded processing allows designers to leverage the MOST open standard network by providing a higher level of customization in a scalable and flexible design solution. The XPS MOST NIC core is fully validated using independent system testing.

The XPS MOST NIC core provides unique support for real-time access to the synchronous portion of the MOST frame through the LocalLink interface (a Xilinx standardized user interface), allowing designs to take advantage of powerful parallel-processing capabilities of the Xilinx FPGA family. The core is delivered with a 32-bit PLB bus suitable for both standalone and embedded applications. A complete hardware and software solution is realized when used in conjunction with a MicroBlaze™ soft processor or hard PowerPC® solution, drivers, and MOCEAN Network Services.

Hardware and Software Requirements

- XA Automotive ECU Development Kit
  - Xilinx XA3S1600E Evaluation Board
  - Xilinx Platform USB cable or Parallel IV programming cable
  - RS232 serial cable and serial communication utility (HyperTerminal)
- Xilinx Platform Studio v10.1.01 or above
- Xilinx Integrated Software Environment (ISE®) v10.1 or above
- MOST fiber optic cables
- MOST Connectivity Board Rev 2.0 or later.

  You can contact the Xilinx Automotive Team for detailed information about the MOST Connectivity Board at: www.xilinx.com/esp/automotive.htm.

Note: The reference system has been built with EDK v10.1.01 and ISE v10.1. These versions can be obtained from the Xilinx software download page: www.xilinx.com/support/download/index.htm.

Reference System Specifics

The Block Diagram of the MOST NIC system is illustrated in Figure 1.

Block Diagram

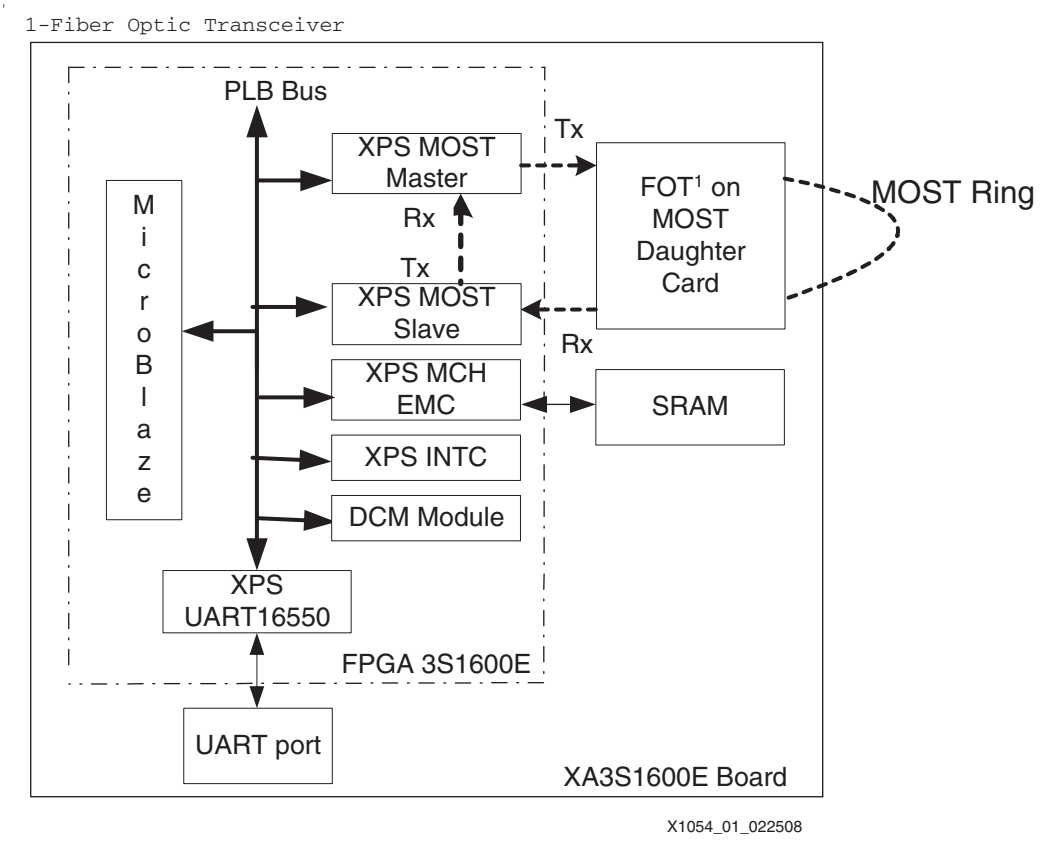

Figure 1: **Reference System Block Diagram**

This reference system is built on a Xilinx Spartan®-3E XA3S1600E board. The components of this system are listed below.

- A MicroBlaze™ processor
- An on-chip hardware debug module
- 8Kb block RAM
- XPS MCH External Memory controller (xps_mch_emc)
- Three DCMs (one DCM for generating the 50MHz PLB CLK and the rest for cleaning up MOST clock, 45.158 MHz MOST COM Clock and 45.158 MHz MOST PLL Clock)
- XPS UART16550 core
- XPS Interrupt Controller core
- Two XPS MOST NIC cores.

The address map for this system is provided in Table 1. The physical connections of the Master and Slave nodes in the FPGA are internally connected, as shown in Figure 1. The external ring on the MOST connectivity board must be connected by the user.

Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
lmb_bram_if_cntlr	dlmb_cntlr	0x00000000	0x00001FFF
lmb_bram_if_cntlr	ilmb_cntlr	0x00000000	0x00001FFF
xps_uart16550	xps_uart16550_0	0x00010000	0x0001FFFF
mdm	debug_module	0x00020000	0x0002FFFF
xps_mch_emc	xps_mch_emc_0	0x30000000	0x3003FFFF
xps_most_nic	xps_most_nic_master	0x80010000	0x80017FFF
xps_most_nic	xps_most_nic_slave	0x80020000	0x80027FFF
xps_intc	xps_intc_0	0x80030000	0x8003FFFF

Configuring a System with Two XPS MOST NIC Cores

Each of the XPS MOST NIC cores is configured to have an operating mode of Master/Slave C_OPMODE= 0, and word counts (FULL & EMPTY) as 16. The crystal oscillator clock signal is connected as an input clock to one of the DCMs to generate PLB clock. The SRAM on the board stores the software application. The interrupts of the XPS MOST NIC cores are connected to the interrupt line of the XPS INTC. The XPS UART16550 displays messages on the HyperTerminal. This section describes how to set up the EDK system with these cores.

Note that even though both nodes could be configured as either master or slave, the roles of the nodes have been fixed to either master or slave for the purposes of this document. Also, a plastic fiber optic cable ring with a connector must be attached to the corresponding connector on the MOST connectivity daughter board. This cable completes the MOST ring, connecting the master and slave nodes. The MOST connectivity daughter board, in turn, has to be fixed into the Molex socket provided on the XA3S1600E board.

Ports and Parameters for the Cores

All the cores of the system (except MicroBlaze) are slaves on the PLBv46 bus. MicroBlaze acts as a master on the PLB bus. The BUFFER_Irpt ports of the xps_most_nic_master and xps_most_nic_slave are connected to the *Intr* port of the interrupt controller. The interrupt request port of the interrupt controller, *Irq*, is connected to the interrupt port of the MicroBlaze (microblaze_0_INTERRUPT).

Figure 2 and Figure 3 show the GUI for the XPS MOST NIC cores. The parameters shown correspond to the operating mode of the core and full/empty wordcount interrupts.

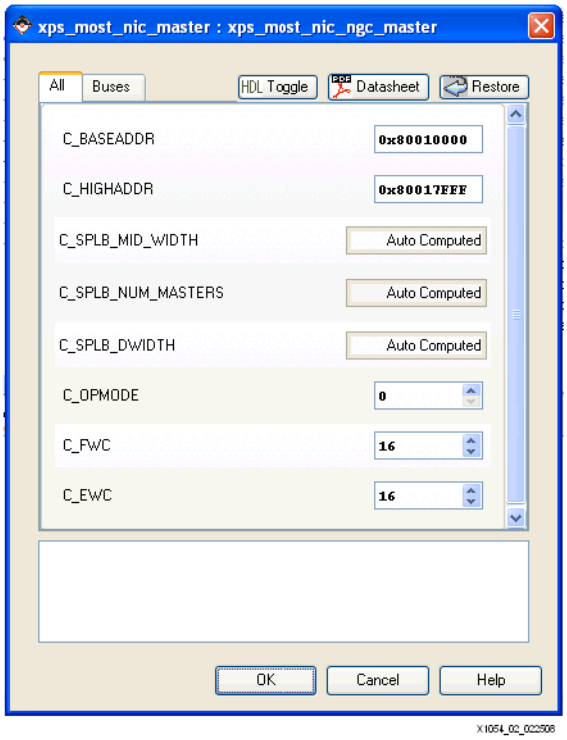

Figure 2: **Parameter Settings for xps_most_nic_master**

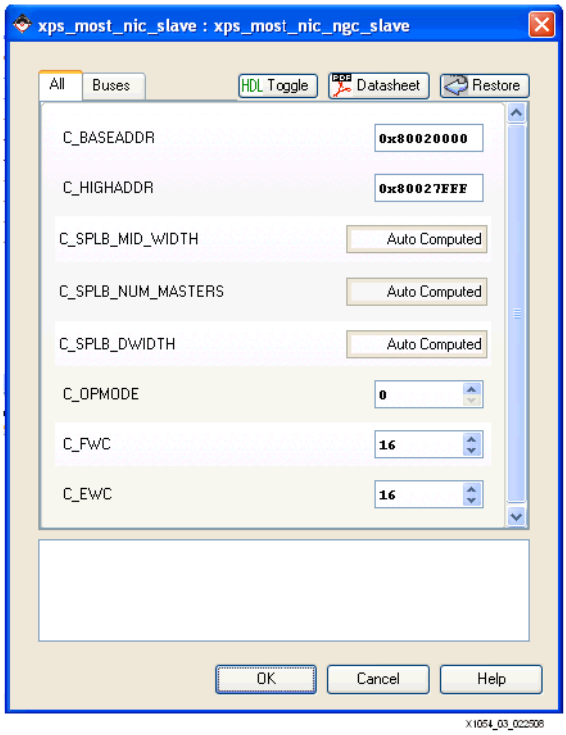

Figure 3: **Parameter Settings for xps_most_nic_slave**

Reference System Clocking

The XA3S1600E board provides a 24 MHz clock. The 24 MHz clock is fed into a DCM, which generates the 50 MHz PLB clock. The MOST clock is taken from IDT PLL on MOST connectivity card.

The Master uses reference clock of PLL as MOST_COM_CLK input and derived clock of PLL as MOST_PLL_CLK input. The Slave uses derived clock of PLL as input for both the clock inputs (MOST_COM_CLK, MOST_PLL_CLK). See Figure 1 for MOST ring connections.

The MOST NIC Software Application

The software application performs the following sequence of actions:

1. Instantiates and initializes driver instances of all peripherals. Two instances of the xps_most driver are instantiated corresponding to the two hardware instances of the XPS MOST NIC core.
2. Sets up the interrupt system for all core interrupts used.
3. Sets up the MOST nodes into master or slave modes.
4. Tests lock status of cores.
5. Sends control messages to Master and slave for channel allocation and deallocation.

   Initially, the Master sends de-allocation message to itself. The Master self-allocates two timeslots through the self-allocation request and the slave gets two timeslots from the Master through an allocation request. Master Allocation Table (MAT) reads are done in Master node to check for proper allocations.
6. Fills up the transmit buffers of the nodes.
7. Writes Logical Channel Enable (LCE) values into both nodes and transmission is enabled for Master Node. Reception is enabled for Slave node and starts transmission.
8. Waits for Tx buffer interrupt of Master, and Rx buffer interrupt of Slave.
9. Resets LCE values by writing zeros to stop transmission and to prevent overflow and underflow errors.
10. Reads back Rx data from Rx buffers when an Rx interrupt occurs and validates it with Tx data.
11. Writes LCE values into both nodes and transmission is enabled for Slave Node. Reception is enabled for Master node and starts transmission.
12. Waits for Tx buffer interrupt of Slave, and Rx buffer interrupt of Master,
13. Reset LCE values by writing zeros to stop transmission and to prevent overflow and underflow errors.
14. Read back Rx data from Rx buffers whenever Rx interrupt occurs and check if it matches with Tx data

The synchronous data transmitted from xps_most_nic_master to xps_most_nic_slave and from the slave to the master is checked for correctness. The application calls the XPS MOST NIC driver functions to set the two XPS MOST NIC cores and the interrupt system for a transmit-receive application. To begin, one XPS MOST NIC core is programmed as a master and other as a slave. Tx buffers are primed with synchronous data and allocation requests are programmed in control buffers. The allocation of timeslots is performed by the master using allocation requests sent by both the nodes.The interrupt handlers are then set, the interrupt system is set up, and all the interrupts of the two XPS MOST NIC cores are enabled. The two XPS MOST NIC cores are then enabled and wait for a stable lock to transmit control messages and synchronous data.

The Master node is configured to use timeslots 0 & 1 for transmission, and timeslots 2 & 3 for reception. The Slave node is configured to use timeslots 0 & 1 for reception and timeslots 2 & 3 for transmission. The allocation of timeslots is accomplished by the allocation requests sent

by both nodes. Initially, the master deallocates all channels and then transmits allocation requests for 2 timeslots to itself. The slave then sends allocation requests for 2 timeslots to the master node.

Synchronous data is transmitted by xps_most_nic_master, which results in calling the Transmit Interrupt Handler (MOSTSendHandler0). The SendDone[0] flag is set to TRUE after the MOSTSendHandler0 is called. The same flow is true for xps_most_nic_slave.

The software waits until data is transmitted by the xps_most_nic_master and received in RXBUFFER of xps_most_nic_slave, which is communicated using wordcount interrupt. The data is then compared with the transmit data of xps_most_nic_master. Then, the software waits until data is transmitted by the xps_most_nic_slave and received in RXBUFFER of xps_most_nic_master, which is communicated using wordcount interrupt. The data is then compared with the transmit data of xps_most_nic_slave.

The application prints status messages at different points in the program through the XPS UART16550 core.

The following are the system-level software settings used in the design:

- BOUNDARY_DESCRIPTOR 4
- TEST_PAYLOAD_LENGTH 128
- MAX_TX_FIFO_CAPACITY 128
- NO_OF_CHANNELS_TO_TEST 2
- MAX_CONTROL_PAYLOAD_LENGTH 19

Figure 4 illustrates the implementation flow of the software application for the XPS MOST NIC cores operating in normal mode.

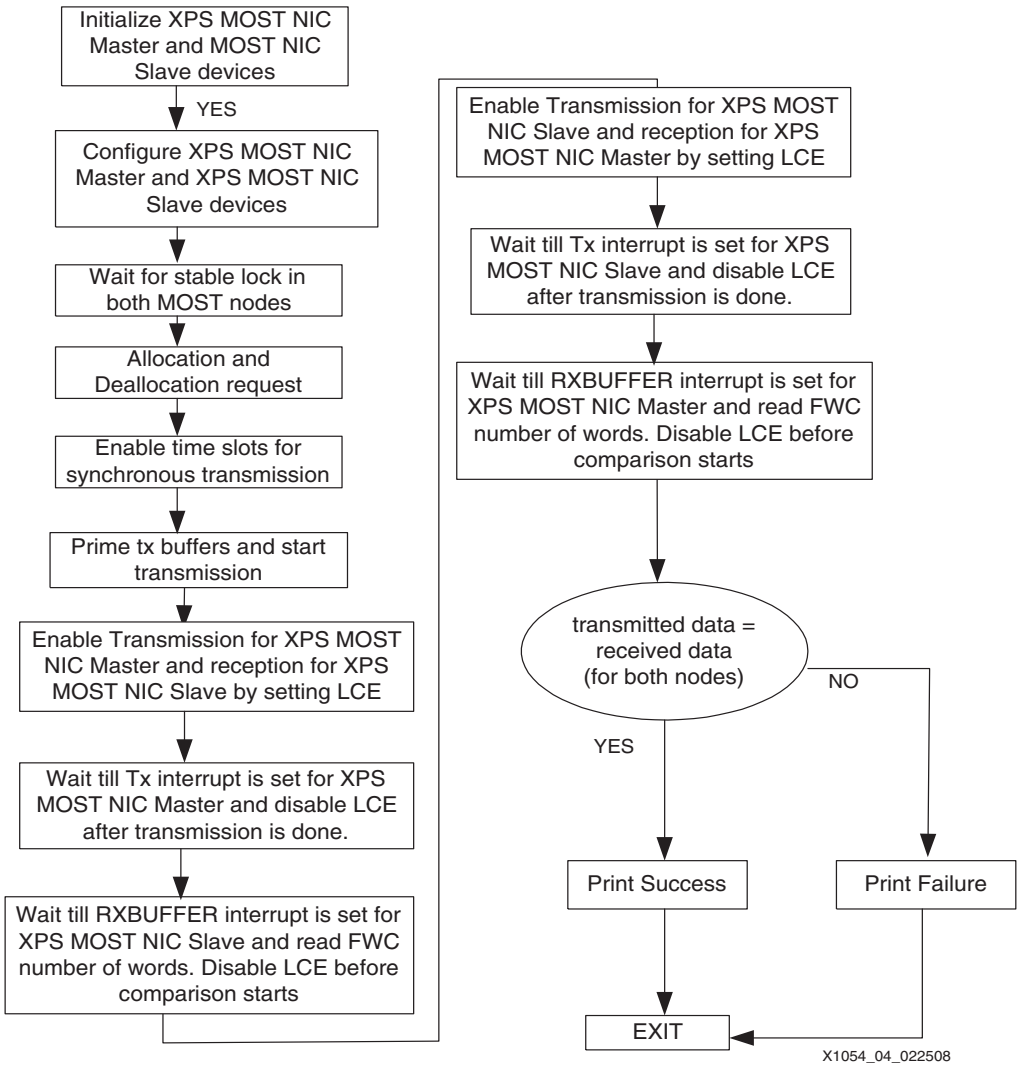

Figure 4: **Software Implementation Block Diagram**

Executing the Reference System

The XA3S1600E board must be connected to the PC via a JTAG cable to download the bitstream. The UART port (X300) of the XA3S1600E board must be connected to the UART port on the PC using a UART cable and the board must be powered-on. A pre-built bitstream, *system.bit* and the compiled software application, *executable.elf*, is available in the ready_for_download directory under the project root directory. The reference system can be executed using the pre-built bitstream together with the compiled software applications, or by generating the bitstream and the software executable in the EDK. Both methods are described in this section.

Note: The XPS MOST NIC is a licensed core and the bitstream generation will fail in the absence of a valid license. To obtain a license key for this core, go to: www.xilinx.com/ipcenter/most/most_registration.htm.

Primary Set-up for Executing the Reference System

1. Using HyperTerminal or a similar serial communications utility, map the utility operation to the physical COM port to be used
2. Connect the board UART port to this COM port.
3. Set the terminal settings as follows:
   - baud rate to 9600

- data bits to 8
- parity to None
- flow control to None.

See Figure 5 for the HyperTerminal settings.

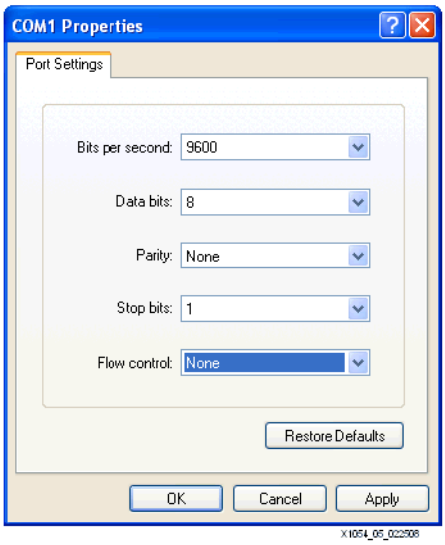

Figure 5: **HyperTerminal Settings**

Using the Pre-Built Bitstream and the Compiled Software Applications

To execute the system using files in ready_for_download directory in the project root directory:

1. Change directories to the ready_for_download directory.
2. Use IMPACT to download the bitstream by using the following command:

```
impact -batch xapp1054.cmd
```

3. Invoke XMD and connect to the MicroBlaze processor by using the following command:

```
xmd -opt xapp1054.opt
```

4. Download the executable by using the following command:

```
dow executable.elf
```

5. Run the software application using the **`run`** command.

Using EDK

To execute the system using EDK:

1. Open the desired XMP file (`system.xmp`) inside EDK.
2. Use **Hardware** → **Clean Hardware** to clean previously generated hardware files.
3. Use **Hardware** → **Generate Bitstream** to generate a bitstream for the system.
4. Use **Software** → **Clean Software** to clean the previously generated software libraries/applications.
5. Use **Software** → **Build All User Applications** to build the software application.
6. Download the bitstream to the board with **Device Configuration** → **Download Bitstream**.
7. Launch XMD with **Debug** → **Launch XMD**
8. Download the executable by the following command:

```
dow executable.elf
```

9. Run the software application using the **run** command.

The status of the software application is displayed in the HyperTerminal data screen. At the end of the software application, the output at the HyperTerminal should be as follows:

```
++++++++++++++++++++++++++++++++++
Xilinx MOST Reference design test
++++++++++++++++++++++++++++++++++
[INFO] Entering main()
[INFO] Entered MostSyncIntrExample()
[INFO] Enabled Master Mode for device0()
[INFO] Enabled Slave Mode for device1()
[INFO] Enabled Normal Mode for Most0 and Most1 nodes()
[INFO] Logical, Alternate, Group addresses set()
[INFO] Setting Interrupt Handlers for MOST0
[INFO] Setting Interrupt Handlers for MOST1
[INFO] Filling Tx Buffers with sync data()
[INFO] Tx Buffers filled with sync data()
[INFO] Programming Tx and Rx CRT()
[INFO] Tx and Rx CRT programmed()
[INFO] Filling Tx Buffers with sync data()
[INFO] Tx Buffers filled with sync data()
[INFO] Programming Tx and Rx CRT()
[INFO] Tx and Rx CRT programmed()
[INFO] Enabling MOST devices()
[INFO] Checking Lock status
[INFO] MOST Node0 SR=B90
MOST Node0 (Master) successfully locked, SR=B90, ERCR=1
MOST Node0 SR=B80
[INFO] MOST Node1 (Slave) successfully locked, SR=B80, ERCR=0
Info: Writing deallocation request message...DONE
Info: Reading from the response FIFO...DONE
Info: Writing allocation request message...DONE
Info: Reading from the response FIFO...DONE
Info: Allocation Table...
Info: MAT[0] 80807070
Info: MAT[1] 70707070
Info: MAT[2] 70707070
Info: MAT[3] 70707070
Info: MAT[4] 70707070
Info: MAT[5] 70707070
Info: MAT[6] 70707070
Info: MAT[7] 70707070
Info: MAT[8] 70707070
Info: MAT[9] 70707070
Info: MAT[10] 70707070
Info: MAT[11] 70707070
Info: MAT[12] 70707070
Info: MAT[13] 70707070
Info: MAT[14] 70707070
Info: Writing allocation request message...DONE
Info: Reading from the response FIFO...DONE
Info: Allocation Table...
[INFO] MAT[0] 80808282
[INFO] MAT[1] 70707070
[INFO] MAT[2] 70707070
[INFO] MAT[3] 70707070
[INFO] MAT[4] 70707070
[INFO] MAT[5] 70707070
[INFO] MAT[6] 70707070
[INFO] MAT[7] 70707070
[INFO] MAT[8] 70707070
```

```
[INFO] MAT[9] 70707070
[INFO] MAT[10] 70707070
[INFO] MAT[11] 70707070
[INFO] MAT[12] 70707070
[INFO] MAT[13] 70707070
[INFO] MAT[14] 70707070
[INFO] Enabling Buffer interrupts()
[INFO] Enabling LCE
Successful transmission by MOST Master
Successful reception by MOST Slave
RxPayload[0][0] =        0
RxPayload[0][1] =        1
RxPayload[0][2] =        2
RxPayload[0][3] =        3
:
:
:
RxPayload[0][124] =       7C
RxPayload[0][125] =       7D
RxPayload[0][126] =       7E
RxPayload[0][127] =       7F
[INFO] Received data of Slave matched with expected (sent) data of Master
Successful transmission by MOST Slave
Successful reception by MOST Master
RxPayload[1][0] =       10
RxPayload[1][1] =       11
RxPayload[1][2] =       12
RxPayload[1][3] =       13
:
:
:
RxPayload[1][124] =       8C
RxPayload[1][125] =       8D
RxPayload[1][126] =       8E
RxPayload[1][127] =       8F
[INFO] Received data of Master matched with expected (sent) data of Slave
++++++++++++++++++++++++++++++++++
Test complete!
++++++++++++++++++++++++++++++++++
```

References

1. *XPS MOST NIC Controller* (v1.00a) Xilinx Product Specification, DS638
2. XA1600E Reference Guide 1.5

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/25/08	1.0	Initial Xilinx release.

Legal Disclaimer

WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.

XILINX PRODUCTS ARE NOT DESIGNED OR INTENDED TO BE FAIL-SAFE, OR FOR USE IN ANY APPLICATION REQUIRING FAIL-SAFE PERFORMANCE, SUCH AS APPLICATIONS RELATED TO: (I) THE DEPLOYMENT OF AIRBAGS, (II) CONTROL OF A VEHICLE, UNLESS THERE IS A FAIL-SAFE OR REDUNDANCY FEATURE (WHICH DOES NOT INCLUDE USE OF SOFTWARE IN THE XILINX DEVICE TO IMPLEMENT THE REDUNDANCY) AND A WARNING SIGNAL UPON FAILURE TO THE OPERATOR, OR (III) USES THAT COULD LEAD TO DEATH OR PERSONAL INJURY. CUSTOMER ASSUMES THE SOLE RISK AND LIABILITY OF ANY USE OF XILINX PRODUCTS IN SUCH APPLICATIONS.