LOCKHEED MARTIN

Advanced Technology Laboratories
1 Federal Street – A&E 2W
Camden, NJ 08102
(609) 338–4250

Application Notes –SVI Verification Task, Case 2 – RNI Encapsulation Study: VHDL Model Description

Date: September 26,1995

Author: Greg Buchanan

Contact: Janet W. Wedgwood (janet.e.wedgwood@lmco.com)

Simple Remote Network Interface (RNI) Encapsulation Study
Using the Cypress HOTLink high–speed serial link
and the PCI fabric interface to implement a sensor–to–PCI RNI

I. Objective of SVI Verification Task, Case 2

The objective of this verification task is to create and demonstrate a VHDL example of a simple Remote Network Interface (RNI) Element. In this case the RNI concept is demonstrated in a sensor interface application which links the Cypress HOTLink high–speed serial link to a PCI interconnect fabric. The RNI model is demonstrated by instantiating the RNI model into the WSSPT SEM–E model, and then exercising the RNI in conjunction with multiple WSSPT PE's. The HOTlink model is available for reuse. Due to its proprietary nature, the WSSPT and PCI models are not available for reuse.

II. Model Description Overview

RNI Model:

The RNI model consists of three functional blocks; each block of the RNI is interconnected via an SVI interface (Figure1):

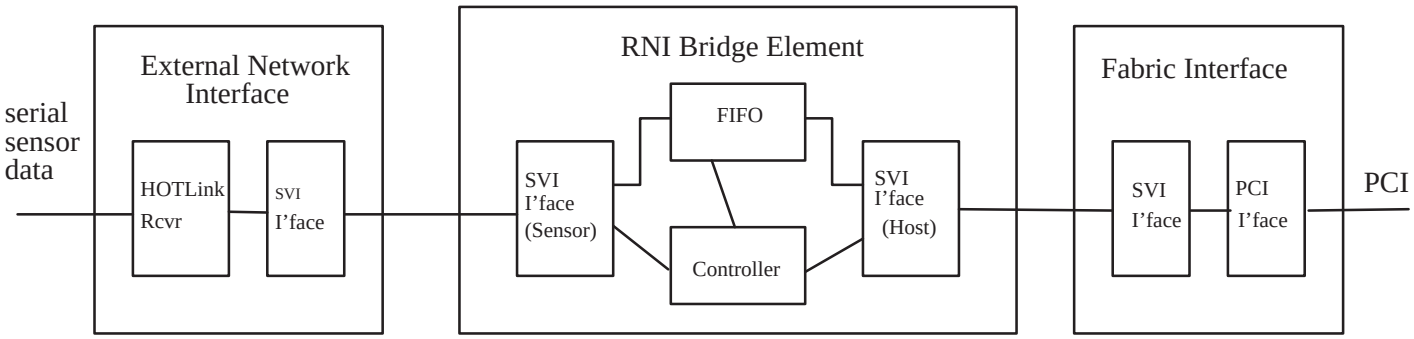

Figure 1 – RNI Model

1) The External Network Interface – consisting of an SVI encapsulated HOTLink receiver model (the sensor). The External Network Interface model is made up of the HOTLink receiver model (in this case, an LMC SwiftModel), and an SVI interface block.

2) The RNI Bridge Element – consisting of four functional blocks: a) the SVI interface to the External Network Interface (or sensor) block, b) the SVI interface to the Fabric Interface (or host–side) block, c) a FIFO block which buffers data flowing from the External Network Interface to the Fabric Interface, and d) a controller block which controls data to and from the host and sensor (Fabric Interface and External Network Interface).

3) The Fabric Interface – consisting of an SVI encapsulated PCI interface. This model was prepared during Case 1 of the SVI Verification Task, and is essentially unchanged from its original form.

Verification Model:

This section describes the HOTlink–ti–PCI RNI as it operates when the RNI model is instantiated into the WSSPT SEM–E board model. (See Figure 2.) The simulation models for this are not available due to their proprietary nature. This information is presented here to indicate how the HOTLink model might be used in a system model.

The original WSSPT SEM–E model consists of from one to four processing nodes interconnected with a PCI–like bus. Each processing node in turn can consist of from one to four processing elements (PE's) interconnected with a proprietary interconnect known as "IOBUS". The verification model is configured as a single processing node containing two PE's with a second processing node replaced by the RNI. The purpose of the verification model, while consisting of a somewhat artificially contrived configuration, is to demonstrate that the RNI can send and receive data across a PCI interconnect fabric. In this case, the first PE in the node functions to pass configuration data to the RNI, after which time the RNI acts as a continuous sensor interface, and passes packets of data to the second PE. The second PE in turn stores the packet data in its memory bank. A dump of the memory at end of simulation enables confirmation of a successful data transfer.

In addition, the WSSPT SEM–E model also contains a stimulus block, which consists of a HOTLink transmitter and file I/O process which provides serial test data to the sensor.

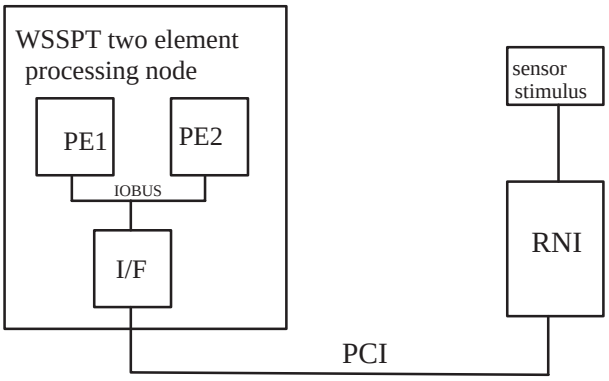

Figure 2 – RNI Verification Model

III. Theory of Operation

For a more detailed description of the Cypress HOTLink transmitter/receiver (CY7B923/933), refer to the Cypress data sheets. For the description of the SVI protocol, refer to the Standard Virtual Interface Specification Version 0.5

Overview of Operation of HOTLink Receiver:

The HOTLink Receiver is the receiver component of a high–speed serial data transfer component set. The receiver can accept serial data at rates from 160 to 330 Mbits/second. Serial data is received by the receiver as ten–bit words which are deserialized, decoded, and checked for transmission errors. The recovered 8–bit byte is presented in parallel to the receiving host along with a byte rate clock, which is one–tenth that of the serial data rate. The component employs the 8B/10B encoding scheme, which was chosen to improve the transmission characteristics of a serial link and to ensure sufficient data transitions in the serial bit stream (one per byte) to make clock recovery possible. It is possible to select a decode–bypass mode, which allows the full ten bits to be presented to the host in an undecoded format.

Special characters, or control codes (compatible with Fibre Channel and ESCON standards), may also be decoded when the receiver is in the decode mode. When the receiver decodes a byte it identifies as a special character, it asserts a ”special character” flag in conjunction with the parallel data. If the decoder finds that the encoded data does not decode to a valid data or special character byte, it presents a violation code on its parallel output and simultaneously asserts a ”received violation symbol” flag. If numerous violations are received within an established time period, it is generally assumed that frame synchronization has been lost (described further below).

A special character, the ”comma” (K28.5), is sent by the transmitter as a pad character whenever new valid data is not presented for transmission. This ensures that the receiver will continue receiving serial data and thereby remain synchronized with the transmitter. In order that the receiving FIFO not become filled with these pad characters, the ”ready” signal – indicating valid data out of the receiver – is inhibited while ”comma” characters are being received. Only the last ”comma” in a string of pad characters will be accompanied by a ”ready” signal. (The presence of ”commas” between packets of data makes it an ideal packet delimiter.)

The clock synchronizer function is performed by an embedded phase–locked loop that tracks the frequency of the incoming bit stream and aligns the phase of its internal bit–rate clock to the serial data transitions. A byte–rate reference clock (which must be identical in frequency to the transmitter) is externally supplied to the receiver for clock synchronization; a byte–rate clock which is phase and frequency aligned to the incoming data is provided by the HOTLink receiver to synchronize downstream logic.

In order for the receiver to correctly decode the incoming serial bit stream, it must be in proper frame synchronization – that is, it must know which bit in the serial stream is the first bit of a new encoded byte. Framer logic in the receiver checks the incoming bit stream for the pattern that defines the byte boundaries. This logic looks for the special character ”comma,” and when it is found, the free–running bit counter in the synch logic is reset thereby reframing the receiver. Random errors in the serial data could possibly cause some bit patterns to become a ”comma” character, and thus cause an erroneous data–framing to occur. The input signal ”reframe” prevents erroneous data–framing by inhibiting the framer logic unless the ”reframe” signal is asserted. When ”reframe” is de–asserted, the receiver will deserialize the incoming data without

trying to reframe the data to incoming patterns. When ”reframe” is asserted, the ”ready” signal is inhibited until a ”comma” is detected; ”ready” resumes normal operation once a ”comma” has been received and the receiver is reframed.

Overview of Operation of HOTLink to PCI RNI Element:

The detailed description of the HOTLink to PCI RNI Element is the VHDL code itself, which contains detailed comments explaining its operation and defining critical signals. Following is a conceptual overview of the RNI:

The RNI contains the logical facilities to provide the following features:

1) Receive a serial 8B/10B encoded bit stream, deserialize it, and present it as a parallel byte over a PCI interface, with all the requisite PCI interface transfer signals.
2) Accommodate completely asynchronous and unique clocking on both the host–side and the sensor–side of the RNI.
3) Buffering of the decoded data (in a FIFO) in order to accommodate different clock rates, to allow for the accumulation of complete packets of data before transmitting data to the host (minimizing monopolization up the PCI bus), and to allow simultaneous receiving of new data while in the process of transmitting previously received data over the PCI bus.
4) Treat invalid data characters according to a number of common error handing protocols (use bad data, use previous good word, substitute zeros).
5) Count errors in a packet, and when an error limit is reached (determined by the host) the packet is declared corrupt and loss of synchronization is assumed. The RNI informs the host of a synchronization error, purges the FIFO of the corrupt packet (if there are multiple packets in the FIFO, the RNI ”tags” the corrupt packet in its error stack, and purges the FIFO when the corrupt packet comes to the top of the FIFO), and automatically goes about reframing itself.
6) Each RNI component – External Network Interface, RNI Bridge Element, and Fabric Interface – is an SVI encapsulated component suitable for inclusion in a RASSP MYA Reuse Library.
7) Receive configuration data from the host that will determine: a) which serial data port (A or B) of the HOTLink receiver will accept data, b) the ”error limit”, or the number of code violations which will be allowed in a packet before it is declared ”corrupt,” c) which error handling protocol is to be used when a code violation is received, and d) whether the HOTLink receiver ”reframe” signal is to be asserted.
8) In addition to configuration data which is determined by the host and may change as required by the system, a number of VHDL ”generic” parameters are passed to the RNI which establish the RNI configuration at run time. These configuration items are ”design specific”, and therefore can not be changed by the host: HOTLink refclk period, maximum error limit, error stack depth (these last two constrain hardware resources), and pci message length (which is used to construct the WSSPT PCI message header).