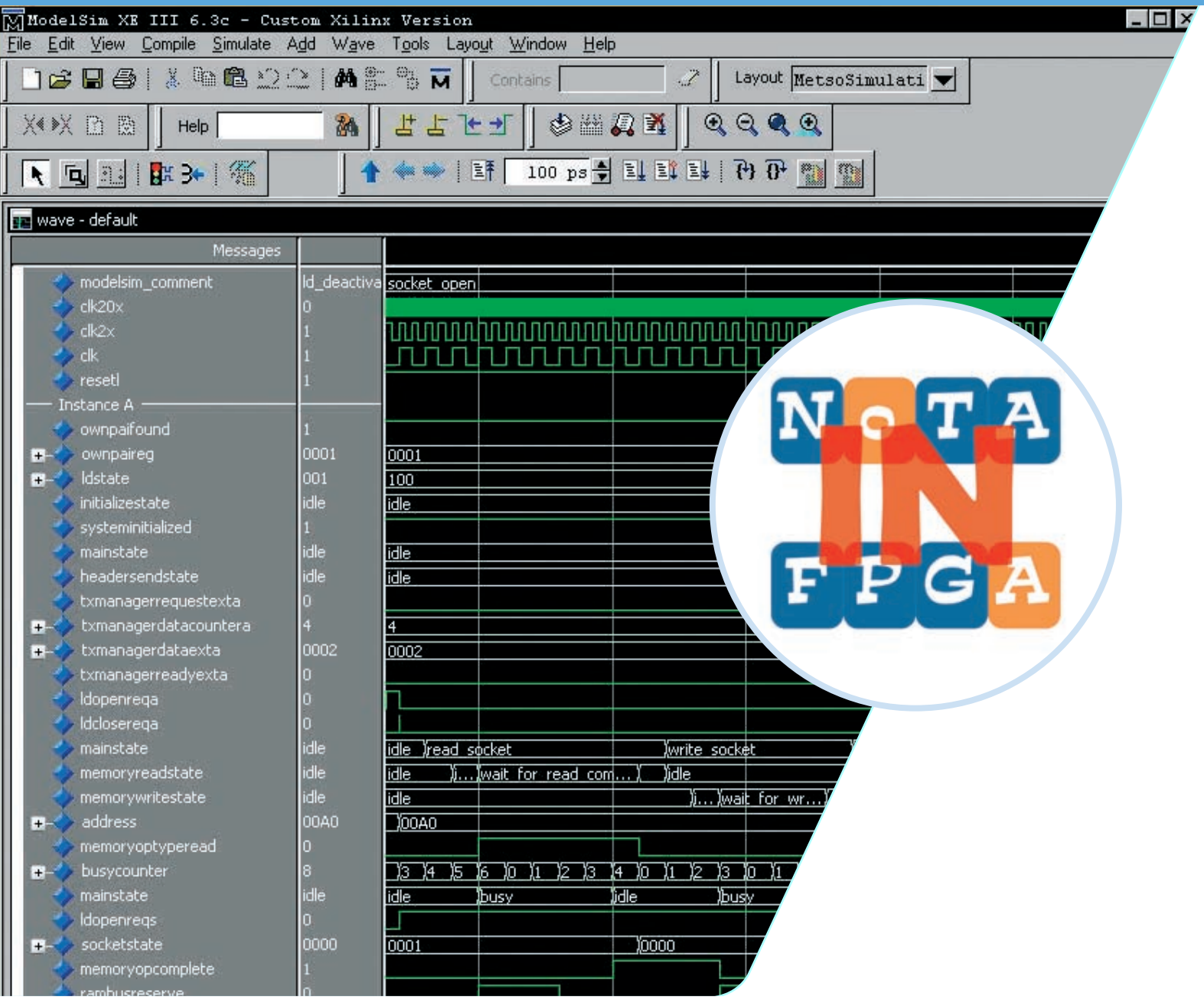

Mikko Metso

NoTA L_INdown Layer Implementation in FPGA

Design Results

ISBN 978-951-38-7186-4 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 3, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 3, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 4374

Technical editing Mirjami Pullinen

Series title, number and
report code of publication

VTT Working Papers 126
VTT-WORK-126

Author(s)
Mikko Metso

Title

NoTA L_INdown Layer Implementation in FGPA Design results

Abstract

NoTA (Network on Terminal Architecture) is an architecture to ease connecting together different devices, or sub-systems inside a single device. DIP (Device Interconnect Protocol) stack is the backbone of each NoTA instance, consising of protocol layers necessary for passing data from one NoTA network endpoint to another.

So far, only NoTA software implementations have been published. In this article, an FPGA hardware implementation of NoTA is evaluated. Part of the NoTA DIP (L_INdown layer for simplified RS-232) was designed, synthesised, implemented and verified on a Xilinx FPGA board ML507. The implementation was built purely in FPGA slice logic and no microcontroller was used. In this article, implemented NoTA FPGA demonstration and its challenges and results are briefly discussed.

ISBN
978-951-38-7186-4 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN		Project number
VTT Working Papers 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)		34578
Date August 2009	Language English	Pages 20 p.
Name of project NoTA in Hardware proof-of-concept	Commissioned by	
Keywords NoTA, FPGA	Publisher VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374	

Contents

1. Introduction

NoTA is an open architecture for connecting together devices, or sub-systems inside a single device. It supports different transport media, meaning that it can be used on top of just about any physical layer transport protocol available. This makes it ideal for easy device interconnect as transport specific software and driver design can be replaced by a much simpler protocol interface. This saves design time, leading to fast time-to-market and cost savings. Different sub-systems can be independently tested and verified without a need to have the target system available at hand. Sub-systems can also be easily replaced by others as long as they all support the same protocol architecture.

NoTA protocol stack is divided in two layers, high level interconnect (H_IN) and low-level interconnect (L_IN). H_IN provides application layer for the service and application nodes. L_IN, on the other hand, provides H_IN the means to connect to the physical transport media.

So far, only software implementations of NoTA protocol stack have been published. All those implementations run on top of an operating system, such as Linux, T-Kernel or Symbian. By contrast, here an FPGA hardware implementation is discussed. It has been designed, built and verified to run on a Xilinx ML507 board with Virtex 5 FPGA. There is a PowerPC microprocessor hardware core unit in the circuit but it is not used. Instead, the implementation is built only using general slice logic and available RAM resources inside the chip XC5VFX70T. The goal in this project was to show that a faster hardware implementation is viable, using only the general slice logic.

The project consisted of two phases. In the first phase, FPGA overview of NoTA L_IN implementation was designed, keeping in mind FPGA logic related characteristics [1]. In phase 2, NoTA L_INdown layer was designed and implemented in detail. This document shortly describes the results of project phase 2.

2. FPGA Implementation Overview

In the FPGA implementation, there are two NoTA instances, A and B, that are both located inside a single FPGA chip. There is an RS-232 type serial bus connection between the two instances. As there is no chip-external NoTA traffic, all NoTA functionality is verified using a Xilinx virtual logic analyser tool called ChipScope Pro.

There are a number of NoTA related actions that can be run in the FPGA. Each of these actions are initiated using six of the pushbuttons mounted on the ML507 board. ChipScope Pro is configured to trig to certain chip internal signals, to show the effects caused by each pushbutton. The pushbutton initiated commands correspond to commands that the two L_INdown instances would receive from an L_INup layer if such would be implemented on top of the L_INdown layers.

Next, a demonstrated pushbutton sequence is described.

2.1 Demonstration Sequence for L_INdown Implementation

The NoTA FPGA implementation was tested and verified first by simulating the VHDL (VHSIC Hardware Description Language) source code, and then by conducting the following test sequence and analysing FPGA internal signals in ChipScope Pro window.

2.1.1 System Load

First, the compiled bitfile is loaded into the FPGA and reset signal is deactivated. In this phase, the L_INdown implementation for both instances is created in the chip but neither of them is yet active.

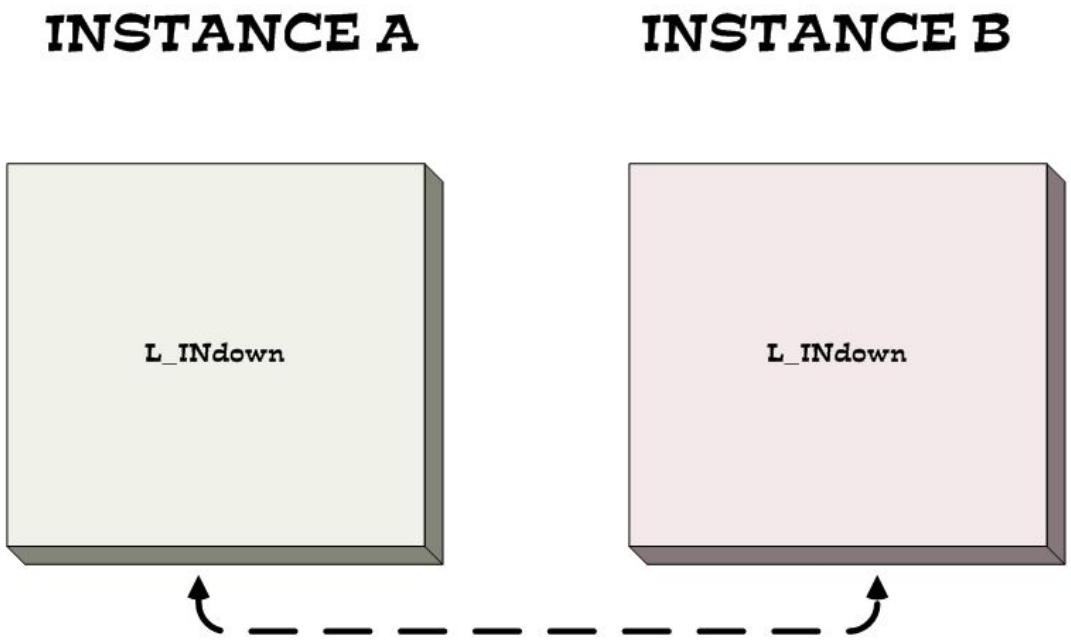

Figure 1. Sequence Step 1: System Load.

2.1.2 Layer Activation

After the system has been loaded and reset signal is released (this can be seen by verifying LED blinking), L_INdown layer needs to be activated on both ends. A pushbutton is connected to an activation request signal, coming from the top of each L_INdown. When the button is released, the layer state is changed from IDLE to ACTIVE. The layer goes to ACTIVE state immediately after the activate command only if the instance is a manager node instance. If not, that instance first goes to RESOLVING state, starting to advertise itself by sending advertisement messages to the manager instance via the serial link. The messages are sent in fixed intervals. Manager address is fixed (here to the value 0x1) and a node without an address uses a temporary address 0.

In this implementation, instance A is set to act as a manager node.

When the manager node receives an advertisement message, it assigns a new address for nodes with address 0 and sends a mode set message back, containing the new address. Once a valid new address is received by instance B (in this case), it also changes to ACTIVE state.

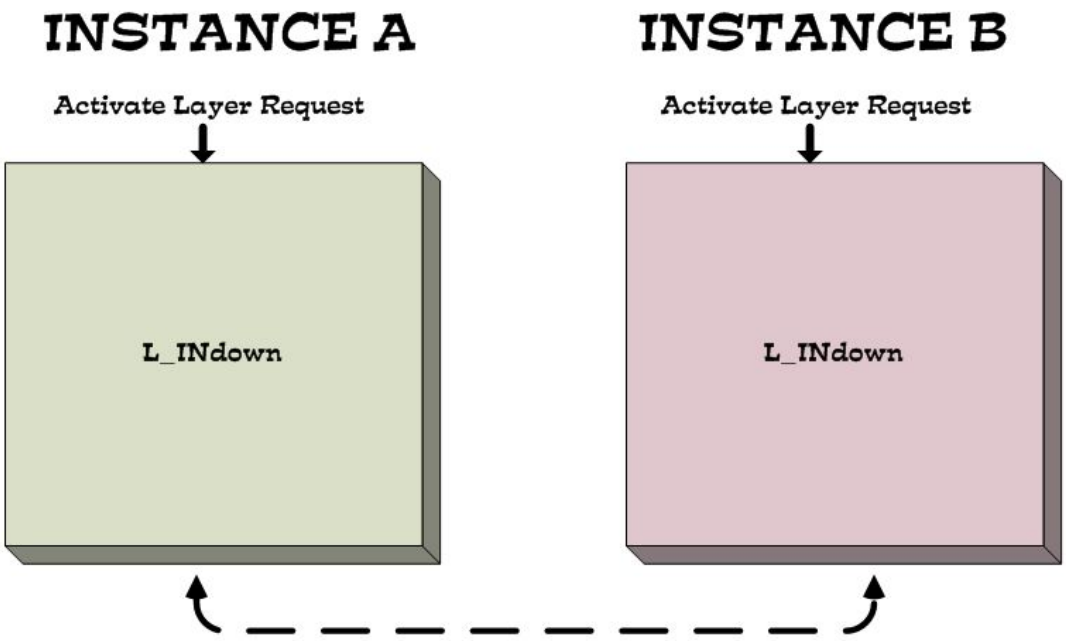

Figure 2. Sequence Step 2: Layer Activation.

2.1.3 Opening of Sockets

The next task is to create sockets for data transmission. At this phase, only connectionless (CL) sockets are supported. When a proper pushbutton is pressed, socket number 0xA0 is opened in instance A, and a socket 0xB0 is opened in instance B. Both instances can have multiple number of open sockets, used for passing data between instances. Socket opening is illustrated in the next figure.

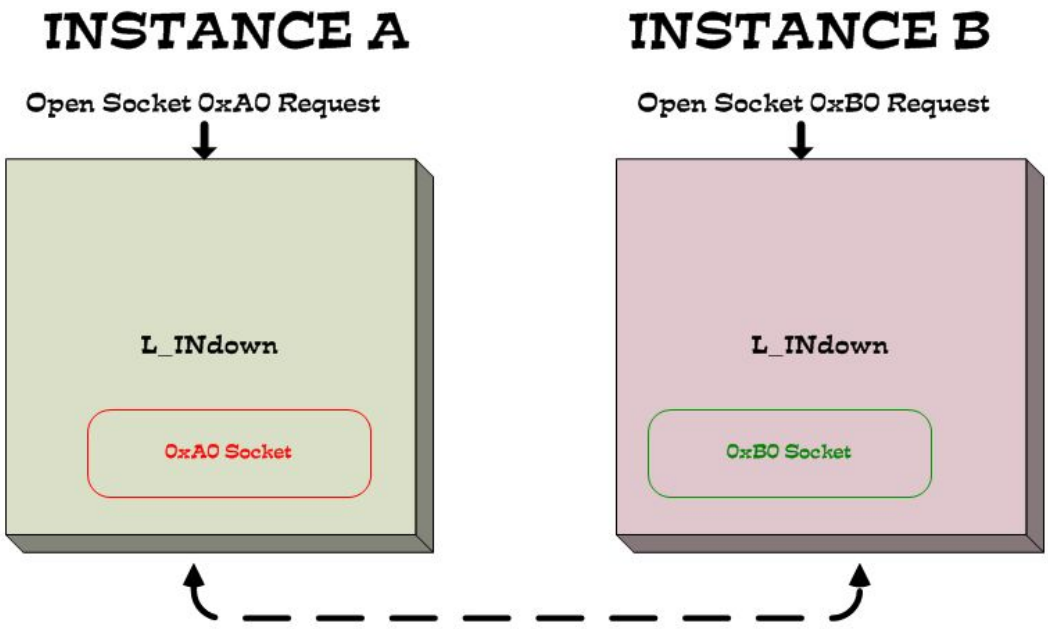

Figure 1. Sequence Step 3: Socket Opening.

2.1.4 Data Sending and Reception

Once open sockets exist between two instances, data can be sent from one to another through those sockets.

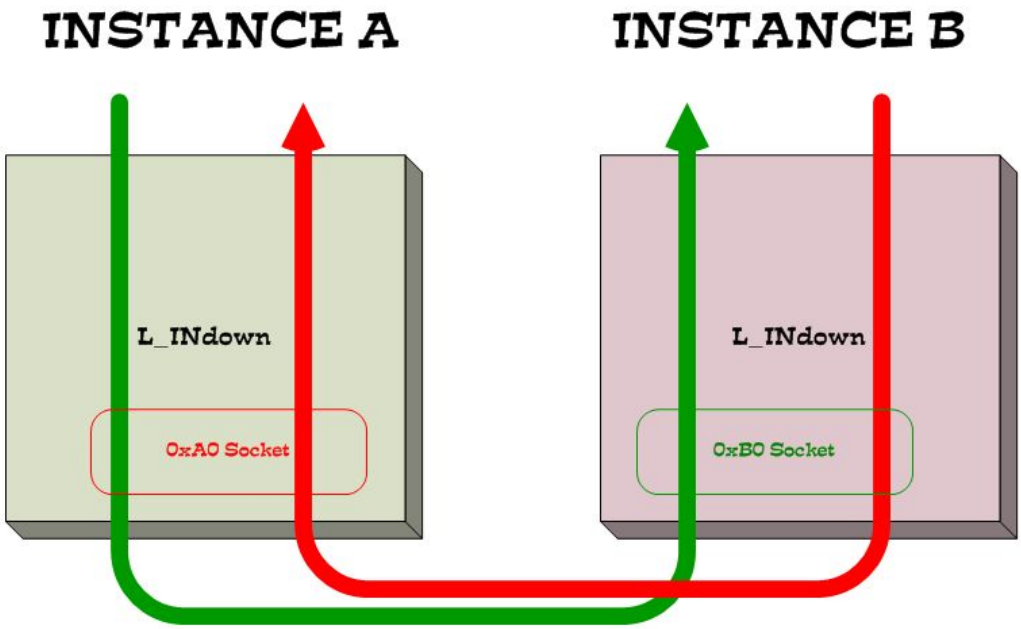

Figure 2. Sequence Step 4: Data Sending and Reception.

2.1.5 System Closing

Once the layers have been activated and sockets are open, data can bee freely sent from the top of one L_INdown layer in one instance to the top of another L_INdown in another instance.

Also system closing was demonstrated. It is done in the opposite order compared to the opening: first, sockets are closed, and then the layers are deactivated.

3. Main Blocks Of FPGA L_INdown

Both of the instances described in the previous chapter consist of similar functional blocks, coded in VHDL. A block level diagram of such L_INdown implementation is shown on the following page.

LdState Vertical Control block determines the current state of the whole layer of a particular instance. During layer activation and deactivation, state control signals are applied through the State Control interface to change the current state.

Socket handling is done in Socket Vertical Control block. The block is closely connected to the Dual Port RAM where a socket address sub-space exists, with information of each socket state.

If the current instance is a manager instance, manager control related data is transferred through the Manager TX/RX Data buses. Manager can, for example, assign a new PAI (Peer Access Information) address to an instance. There is a dedicated sub-address space for PAI addresses in the RAM. In this case, Manager TX Data bus is used to 1) send a corresponding mode set message to the receiving instance, and 2) to write the new address to the local RAM. On the other hand, Manager RX bus informs the manager logic of an arriving advertisement message. Local RAM Control block handles all RAM write tasks. The RAM can be read also by other blocks. RAM Arbiter has the RAM accessing control.

The current PAI address of each instance is stored in Own PAI Register. Manager PAI is a constant and pre-programmed in VHDL source code.

All L_INdown data traffic through the RS-232 type interface is handled in TX Packet Processing and RX Packet Processing blocks. In receive direction, serial data is first converted to 16-bit parallel form, fed through a FIFO buffer and then processed according to the packet's header field information. It can be either addressed to the receiving L_INdown instance, or to be forwarded on to the upper layer through another FIFO interface (FIFO UP RX). On transmit side, packets to be serialized can also come either from the upper layer, or the local L_INdown control logic.

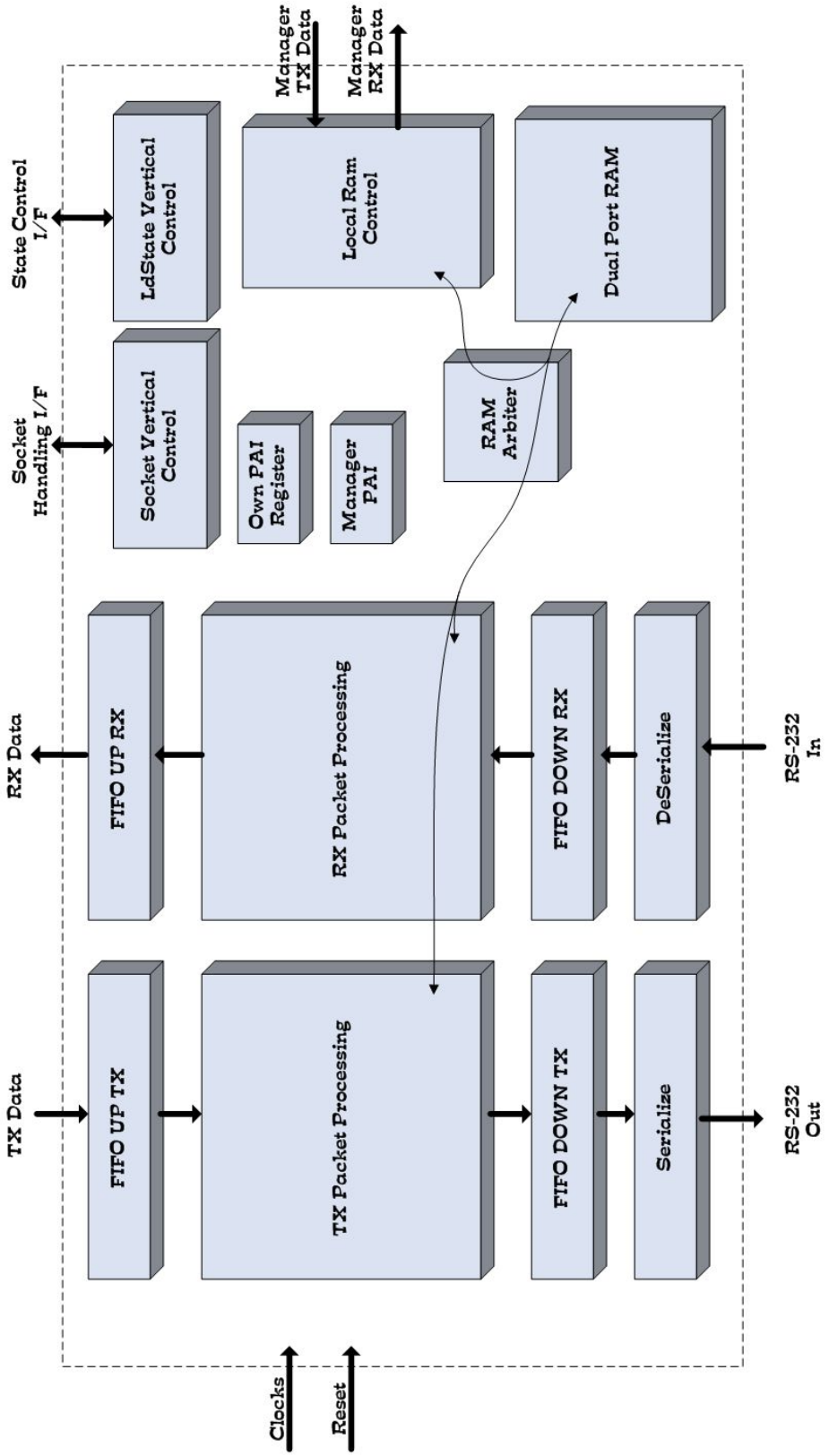

Figure 5. L_INdown Block Diagram.

4. Simulation

Ways of data processing in an FPGA greatly differ from processing in a microprocessor. The main difference is that a microprocessor program code is executed sequentially, whereas an FPGA implementation must be seen to run in a much more parallel manner. As a result, implementing the system for FPGA is very different compared to a software implementation. Due to the parallellism, timing is a critical thing that needs be kept in mind constantly during FPGA design. Every block can be active all the time and driving output signals according to their inputs after each clock tick.

A thorough system simulation is often critical in finding possible error causes and bottlenecks before the code is actually synthesized. On the following page is a view from a Modelsim simulation window where an L_INdown layer socket opening is simulated.

Even though FPGA programming can be more time-consuming, the result is often significantly faster and more power efficient than a corresponding software implementation.

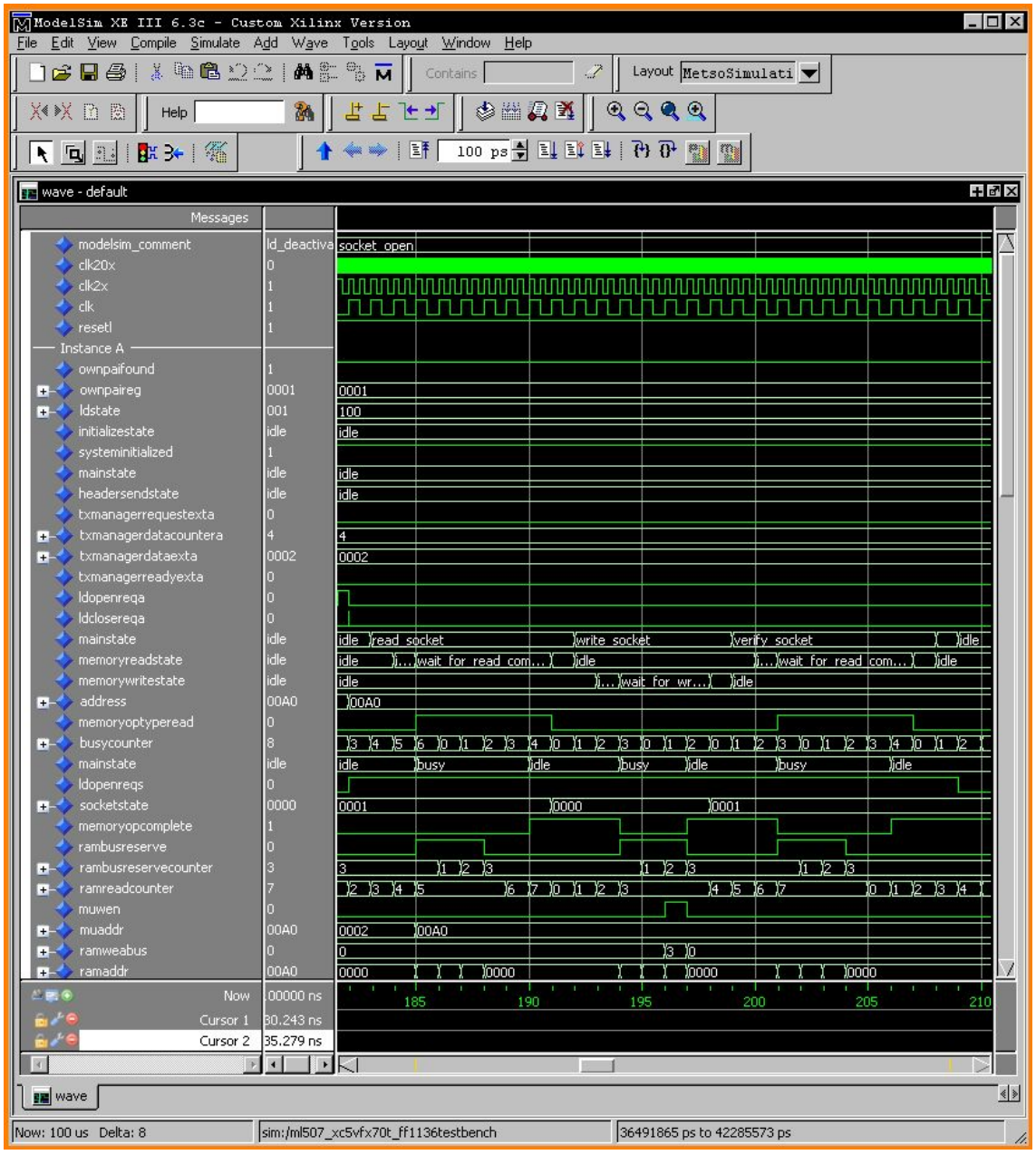

Figure 6. Simulation window of socket opening.

5. Resource Utilization

Utilization of FPGA resources in the current L_INdown implementation is shown below. It includes two NoTA L_INdown instances (A and B), and some additional logic needed for such as ChipScope Pro cores, clock signal generation and pushbutton functionality.

```
Device Utilization Summary:

   Number of BSCANs                          1 out of 4      25%
   Number of BUFGs                           6 out of 32     18%
   Number of DCM_ADVs                        2 out of 12     16%
   Number of FIFO36_EXPs                     8 out of 148     5%
   Number of External IOBs                  11 out of 640     1%
      Number of LOCed IOBs                   9 out of 11     81%

   Number of RAMB18X2s                       2 out of 148     1%
   Number of RAMB36_EXPs                     2 out of 148     1%
   Number of Slice Registers              1533 out of 44800   3%
      Number used as Flip Flops           1525
      Number used as Latches                 8
      Number used as LatchThrus              0

   Number of Slice LUTS                   1677 out of 44800   3%
   Number of Slice LUT-Flip Flop pairs    2271 out of 44800   5%
```

Figure 7. FPGA utilization.

We can see from the utilization summary that the current L_INdown implementation uses only a fraction of all resources available in the circuit.

As an estimate, we can say that NoTA L_INup layer construction would require roughly the same amount of logic resources as L_INdown. H_IN, on the other hand, is a bit larger and it might require about double of the resources compared to L_INdown. The amount of slice registers used for one L_INdown instance is roughly half of the amount utilized in the current demonstration setup with two of those instances. When adding up these estimates we can say that one NoTA instance with full NoTA DIP stack would require ½ * 1533 * 4 = 3066 slice registers, in which case 14 NoTA instances with full stack would fit inside this particular FPGA.

Worth noting is that the current L_INdown version still lacks some functionalities, such as connection-oriented packet (stream) transfer, that will require a bit of additional logic. Node implemented on top of the NoTA stack also requires some logic, depending on its purpose.

Also, there are FPGAs on the market with far greater amount of programmable logic resources than there are in the one used for this demonstration. And the current version of the FPGA implementation is not resource optimized. So we need to regard this as a very rough estimate.

6. Summary

The main goal of this project was to show that a NoTA L_INdown layer was viable to implement in FPGA hardware, without using any microprocessor component or an operating system below the NoTA DIP. Even though no resource or speed optimization was in the scope of this study, the two NoTA instances with a connecting serial interface were seen to easily fit inside a Xilinx Virtex 5 FPGA and the performance was fast. The system can concurrently send and receive packets, while doing other control tasks (such as socket operations) at the same time as well.

The constructed implementation does not support all the functionality there is in the current NoTA release (3.0) specification. For example, connection-oriented (CO) data transmission is not yet possible. Still, the basic NoTA functionality is there and it has been verified to work on an actual circuit board. This encourages us to aim further towards a full NoTA protocol stack running in an FPGA.

References

[1] Metso, M. RautaNota Project Phase 1 Research Report. VTT, Feb. 24, 2009.

[2] Eriksson, T. Low Interconnect (L_IN) Release 3.0, Protocol and Interface Specification. Ver. 0.5, Sept. 1, 2008.

As the amount of portable devices keeps growing, there is a constant need for connecting them together easily to form more intelligent and easy-to-use systems. NoTA is an architecture to enable these connections.

VTT has been working on this issue for years from both software and hardware perspective. This document presents implementation results of the first phase in building a hardware NoTA protocol stack. It not only shows that NoTA can be applied in hardware, but the implementation can be made very fast with only a fraction of available FPGA logic resources used. These achieved results encourage to continue with NoTA hardware design to reach an FPGA system running a full NoTA stack. Also the main functional blocks of NoTA L_INdown layer FPGA implementation are presented, which provides a basis for the design of other NoTA layers in an FPGA as well. This document is intended for anyone interested in the implementation of a socket based software component in an FPGA.

ISBN 978-951-38-7186-4 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)