NetFPGA Architecture

Last modified: 05/31/06
Greg Watson (gwatson@stanford.edu)

This document describes the architecture of the NetFPGA system at Stanford University. See http://klamath.stanford.edu/NetFPGA/
NetFPGA is partially funded by the NSF under Grant EIA-0305729.

Table of Contents

Board.................................................................................................................................2
PCI....................................................................................................................................2
Ethernet.............................................................................................................................2
  Linux Host to User card (tx) path.................................................................4
  User card to Linux Host (rx) path.................................................................4
Configuration.................................................................................................................4
User mode......................................................................................................................5
Appendix A  Byte Ordering..........................................................................................6
Board

The NetFPGA board is used in two ways: as a system controller and for user (student) designs. We focus on the controller application as that is a superset of the functions required by the user application.

The block diagram is shown in figure 1. Two FPGAs are used: a small Spartan-II device provides the PCI interface while a large Virtex2Pro device provides the Ethernet MACs (and is where the user designs will reside). The terminology used for each device is: CNET is the Controller Network device; CPCI is the Controller PCI. UNET and UPCI are the User Network and PCI devices.

PCI

The actual PCI protocol is implemented using IP provided by Xilinx. This provides a standard 5V 32bit/33MHz Master/Target PCI. This is augmented with a simple DMA controller that will allow packets to be DMA'd between the controller board and the Linux operating system.

In addition a simple register read/write module will allow single 32-bit reads and writes to both the PCI device and the Virtex2Pro.

Ethernet

The CNET provides four tri-mode MACs (10/100/1000) connected to an external quad PHY. The controller board acts like a switch – its role is to transfer packets between the Linux host and a user board – up to four user boards can be connected to a single controller board. As such, networking performance is fairly important.

We assume that Ethernet-level flow control is not being used. Consequently we exploit the inherent network speed to simplify the transmit path, as described below.

Clock Domains

There are two main clock domains shown in figure 1: the PCI clock domain (33Mhz) and the system clock domain (62.5MHz). The CPCI handles the clock domain crossings (because it is to difficult to extend the PCI clock into the CNET).

The CNET device also has a number of additional clocks associated with each of the MACs. These are described further in the CNET Architecture document.
Figure 1 NetFPGA Block diagram
**Linux Host to User card (tx) path**

In this direction the Linux driver should verify that the TxFIFO of the appropriate MAC has sufficient space for the packet to be transmitted. Assuming gigabit operation then this will almost always be the case, and if not then the TxFIFO should be empty very soon (15 microseconds). If, after some period of time, the TxFIFO has not become available, then the driver should discard the packet and flag the error.

Having determined that the TxFIFO is empty the driver should set up the PCI DMA engine to DMA the packet from kernel space to the appropriate TxFIFO, and signal completion when done.

Note: in order to meet PCI timings the PCI DMA controller may need to have some local FIFO space so that incoming burst read data can be stored before being passed across to the Virtex device.

**User card to Linux Host (rx) path**

This path is more complicated due to the fact that the controller card might receive a packet from all four MACs simultaneously. Consequently additional buffering should be provided. The 1Gbps peak bandwidth of PCI means that it is not possible for the system to sustain reception of packets on all four MACs, however it should try to buffer as many packets as possible so as to reduce the drop probability.

With this in mind the system should provide large RxFIFOs (as large as possible) for each MAC. The Rx Arbiter unit will then transfer packets from the RxFIFOs to the external SRAM, and notify the Linux driver that a packet has been received.

The Linux driver should then set up the DMA engine to DMA the packet from SRAM to kernel memory.

Note: the PCI DMA controller will probably need some FIFO space to ensure that data is available before initiating a PCI burst write.

Note: for diagnostic purposes the Virtex should support direct PCI reads and writes to the MAC Tx- and RxFIFOs.

**Configuration**

The PCI design is expected to change only rarely, and is needed at power-on, and so will load up from Flash whenever power is asserted to the board.
The Virtex2Pro must be configured by downloading a device file through the PCI device. This will require application support running on the Linux host, and will provide a very simple FIFO-like interface to program the V2P.

**User mode**

When the NetFPGA board is in user mode it will only implement one of the four MACs. In addition it needs to provide a 'VLAN' layer between the user logic and the FIFOs such that the board appears to have many (four, perhaps eight) MACs. Depending on what FPGA resources are available, there might be a separate FIFO provided for each virtual MAC.

The user mode board will probably only support single word accesses over the PCI bus – DMA will not be supported. This is done in order to simplify the user logic. Figure 2 illustrates the user view of the NetFPGA board. Note: this is a logical view; it still has the same hardware as the controller board.

![Figure 2 User view of the NetFPGA board](image-url)
Appendix A  Byte Ordering

The following describes the byte ordering within the NetFPGA system. The byte ordering in the host system might be different, in which case the PCI interface must reverse the byte ordering as packets enter and exit the board.

When packets are transferred across a 32-bit bus they should be ordered such that the first byte appears in bits 7:0, the second in 15:8, etc.

So a 65 byte packet (excluding FCS) with MAC Destination Address AA-BB-CC-DD-EE-FF and Source 11-22-33-44-55-66 and final bytes 00....00 01 02 FF would be passed as:

<table>
<thead>
<tr>
<th>Word</th>
<th>Bytes</th>
<th>Value</th>
<th>EOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>DD CC BB AA</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>22 11 FF EE</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>66 55 44 33</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>02 01 00 00</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>XX XX XX FF</td>
<td>1</td>
</tr>
</tbody>
</table>

Note: This is not the endian-ness of the system. NetFPGA only operates on packets which are streams of bytes – it has no concept of endian-ness. The packets must already be in network byte order (big-endian) – see the C functions htons, htonl, ntohs, ntohl, etc.