Linking Field-Programmable Gate Arrays to Linux PC

Marcus Baines
Science Undergraduate Laboratory Internship Program
Norfolk State University

Stanford Linear Accelerator Center
Menlo Park, California

August 15, 2008

Prepared in partial fulfillment of the requirement of the Department of Energy's Science Undergraduate Laboratory Internship program under the direction of Ryan Herbst in the Central Laboratory Annex of the Stanford Linear Accelerator Center.

Participant: ______________________________
Signature

Project Advisor: ______________________________
Signature
# Table of Contents

Abstract ....................................................................................................................3  
Introduction...............................................................................................................4  
Materials and Methods..............................................................................................5  
Conclusion.................................................................................................................7  
Acknowledgments.....................................................................................................8  
References.................................................................................................................8  
C++ Class................................................................................................................10  
FPGA.......................................................................................................................14  
FPGA and Linux PC Diagram................................................................................15  
VHDL Hardware Diagram......................................................................................16
Abstract

The purpose of this project is to provide a high speed data link between FPGA-based designs and a Linux PC. This is a general purpose design that will be used in many applications, with the primary target being ILC detector development. This project is broken into three pieces: the hardware interface, the software API (Application programming interface), and the data transfer between hardware and the software. Through the use of two program codes, C++ and VHDL, the hardware and software will be able to break up packages that are too large (larger than 8 KB) into smaller frames and send them over the Ethernet, using the PGP protocol, at the theoretical rate of 125 MB per second. Also, through this protocol, the user interface will be rate matched with the Ethernet MAC so handshaking is not required from the receiver of the frames.
Field-Programmable Gate Arrays, or FPGAs, are semiconductor devices that contain many programmable logic components, called “logic blocks,” and programmable interconnects. Logic blocks can be programmed to perform the digital functions of basic logic gates such as AND and OR, or more complex combinational functions such as decoders or mathematical functions. The logic blocks are identical and can be viewed as standard components of the integrated circuit and can each, independently, take on any one of a limited set of personalities. The individual logic blocks are interconnected by a matrix of wires and programmable switches. A user can implement their design by specifying the simple logic function for each block and selectively closing the switches in the interconnect matrix. Since the chip is programmed by the user and not the manufacturer, the words “field programmable” are applied.

The current implementation of this hardware uses a USB link between the FPGA and a PC as a data transport mechanism. This interface is limited to a throughput of around 300 KB per second. In addition to the throughput limitation, there are known bugs with the Linux driver for the USB chip, which causes errors to occur periodically in some setups. Initial tests with a 1Gbit Ethernet link have shown that a data rate of around 80-90 MB per second can be achieved, which is slower than the theoretical rate of 125 MB per second due to a few handshake issues. The hardware interface on the FPGA side will emulate the interface used by the PGP (Pretty
Good Protocol). The PGP supports the movement of data across a high speed communications link (3.25 GB per second) over four independent virtual channels. Each of the four virtual channels appears as a separate set of interface signals to the FPGA hardware, which are further broken down into transmit and receive signals. On the software side of the link, a C++ class will be created to open a link and to receive and transmit frames from and to the user logic in the FPGA. The majority of the effort in this project is the definition of the format of data which moves between the FPGA hardware and the software. The C++ class created will also place a header on the frame to be sent. This header will tell the receiving program what type of frame it is, which virtual channel it is being transmitted over, the size of the data being received, the beginning and end of the frame, and whether or not there was any error during transmission of the frame. This and many other C++ and VHDL (Very High Speed Integrated Circuit Hardware Descriptive Language) programs will be downloaded to program the FPGA to send and receive data to and from the Linux PC.

**Materials And Methods**

The frames that will be sent between the software and hardware will be raw Ethernet frames. These frames obtain the following standard format: Destination MAC Address (bytes 0-5); Source MAC Address (bytes 6-11); Protocol (bytes 12 and 13); User Data (bytes 14-as much needed); and CRC(Cyclic Redundancy Check) (last 4 bytes automatically appended by the Xilinx FPGA). When the Linux PC sends information to the FPGA, the MAC Address of the FPGA is placed in the Destination MAC Address field, and the PC's MAC address is placed in
the Source MAC Address Field and vice versa if the PC is receiving information from the FPGA. The information that is being sent is placed in the User Data field, which has a maximum size of 8 KB. Even though there is an Ethernet MAC built into the FPGA, we are dealing with only one FPGA, and the MAC address field for the FPGA will not be important. Therefore, when sending information from the Linux PC to the FPGA, the Destination MAC Address field would not be important, and the same is true for the Source MAC Address field when the PC is receiving information from the FPGA. This gave us 6 free bytes to create our header that will contain the valuable information about the data being sent.

The C++ class creates a header of the following format: Serial Number (bytes 0 and 1); Type of Frame (Null, Write, Read) (byte 2); Virtual Channel used and whether it was a SOF(Start of Frame), EOF(End of Frame), EOFE(End of Frame Error), or Error (byte 3); and Size of Data (bytes 4 and 5). The Serial Number field of the header tells the receiver of the frame the sequence of the frame. The Type field has two cases that will be explained later. The next field tells the receiver which virtual channel is used and whether it is the first of, possibly, many frames, last of many frames, or whether there was an error with sending the current frame. The Size field lets the receiver know how much user data to expect. The VHDL code creates a receiver region and a transmitter region that contains two state engines (TxEngine and VcEngine), an 8 KB buffer (FIFO), and a 4:1 virtual channel Mux. (Figure 4) The receiver region is the area which receives the frames passed on by the software. This region reads the header of the frame and decides the course of action it should take. The transmitter region is the area which either sends an acknowledgment of received data (ACK) or the data requested by the
Case 1 is when the PC wishes to receive information from the FPGA. The PC must send a request frame of type Read to the FPGA indicating which virtual channel it wishes to access. The receiver region of the FPGA hardware program will receive the frame and read the header. If no errors occurred during transmission, it will pass the Serial Number, Virtual Channel, and a read request flag to the transmitter region of the program. Receiving the read request flag, the TxEngine uses the Mux to select the virtual channel indicated, while sending a read request flag to the VcEngine. This instructs the VcEngine to pull the data (up to 8 KB) into FIFO. While the data is being placed into FIFO, the TxEngine waits for the VcEngine to return a Done signal. If there is no data waiting on that channel, FIFO will send back data of size zero, while the VcEngine sends back and Done signal with a Size count of zero. Upon receiving the Done signal, the TxEngine sends out a signal to the MAC indicating that it is ready to start sending frames. Once the engine receives the “OK” it generates a frame header of type Null with the Serial Number and Virtual Channel. The software reads the header and notifies the user that no data was accessible on the virtual channel indicated. This action would then return control back to the user, giving them the chance to make a call to a different channel or end the program. However, if there is data on the indicated virtual channel, the VcEngine will pull the data into FIFO while counting each byte. Once this process is complete, the VcEngine will return a Done signal with the data count. The TxEngine will follow a similar process, but will create a frame of type Write (which now includes the Serial Number, virtual channel, SOF(EOF, or EOFE), and size of data) and the obtained data to software with the approval of the MAC. Whether or not
data was obtained, the software will return an ACK frame to the FPGA, letting it know what the software received. One should take notice that both programs are constantly checking for errors in data acquisition and transmission throughout the entire process to determine whether or not the session should continue running.

Case 2 is when the PC wishes to send information to the FPGA. In this case, the software creates a frame header of type Write and follow a process similar to that of a Case 1 of the read request. Before the software starts to send the data, it must also send a request to the Ethernet MAC built into the FPGA and wait for the “OK” to start. Once received, the software program creates a Serial Number, marks the virtual channel chosen by the user, and creates an array based on the size of the data provided by the user. Once all information is placed in the correct areas, the frame is sent to the FPGA. As stated before, the receiver region takes in the frame and reads the header. This time it passes the data on to the hardware and passes the Serial Number, Virtual Channel, Size of data, and an ACK flag to the transmitter region. The TxEngine will send an ACK frame back to the PC with the Serial Number, Virtual Channel, and Size of data in the correct fields. Take note that the software program follows this same procedure when it receives a frame from the FPGA.

Conclusion

In conclusion, the C++ software program has been completed and appears to be what is needed. However, we haven't been able to test the transmission rate because the VHDL program has not been fully completed yet. There are many steps to follow and conditions to consider. I
continue to work on this program with hopes of completing it before the end of the internship. It is believed that once the program is done and downloaded to the FPGA, the transmission of the frames will be much closer to, if not approximately, the theoretical rate of 125 Mb per second.

Acknowledgments

These programs were written in Building 84 Central Laboratory Annex at Stanford Linear Accelerator Center (SLAC). I would like to take this opportunity to thank my mentor Ryan Herbst. He has provided me with an unlimited amount of help and guidance throughout this entire program. I would also like to thank Steve Rock, Farah Rahbar, and Susan Schultz for their guidance (and paychecks) during this internship. Finally, I would like to thank the Department of Energy, Stanford University, and the SULI program for giving me the chance to participate in this wonderful and exciting internship.

References

http://www.fpga4fun.com/FPGAinfo1.html


Figure 1: C++ Class

The following is a copy of the C++ class written for the software program:

```cpp
#include <iostream>
#include <iostream>
#include <unistd.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <linux/if_ether.h>
#include <linux/if_arp.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <asm/types.h>
#include <iostream>
#include <iomanip>
#include <sstream>
```
#include <string>

using namespace std;

class PgpOverEthernet {

    string interface;
    int sock;
    struct sockaddr_ll socket_address;
    struct ifreq ifr;
    struct ethhdr eh;

public:
    PgpOverEthernet(string interface);
    ~PgpOverEthernet();
    int pollFlowControl(int vc);
    int open();
    void close();
    int transmit(int vc, char*buffer, int size);
    int receive(int vc, char*buffer, int maxSize);

};
int SN;
int packetSize;

class Type {
public:
    static const int Null = 0x00;
    static const int Poll = 0x01;
    static const int Write = 0x02;
    static const int Read = 0x03;
};

class VC/Frame {
public:
    static const int vc0/F_SOF = 0x01;
    static const int vc0/F_EOF = 0x02;
    static const int vc0/F_EOFE = 0x04;
    static const int vc0/F_Error = 0x08;
    static const int vc1/F_SOF = 0x41;
    static const int vc1/F_EOF = 0x42;
    static const int vc1/F_EOFE = 0x44;
    static const int vc1/F_Error = 0x48;
    static const int vc2/F_SOF = 0x81;
}
static const int vc2/F_EOF = 0x82;
static const int vc2/F_EOFE = 0x84;
static const int vc2/F_Error = 0x88;
static const int vc3/F_SOF = 0xc1;
static const int vc3/F_EOF = 0xc2;
static const int vc3/F_EOFE = 0xc4;
static const int vc3/F_Error = 0xc8;
};

class PollFlow {
public:
    static const int AFull = 0x40;
    static const int Full = 0x80;
};
Figure 2: FPGA
Figure 3: FPGA and Linux PC

Field-Programmable Gate Array

VHDL Hardware → Ethernet MAC

Ethernet MAC → Ethernet Link = 125 MB per second → C++ Software

Linux PC
Figure 4: VHDL Sections

- Reads Header
- Mark Read Request flag
- Pass Data on to Hardware
- Mark ACK flag
- Pass SN and VC to Transmitter

Incoming Frames

Outgoing Frames

VHDL Hardware

16