1 Introduction

The NetFPGA[2] community has made significant progress in making experimentation with high-speed reconfigurable networking hardware[6] easier and more accessible. They have provided an environment in which hardware modules implemented in Verilog can work together with software running on a host CPU using the primitives of DMA packet transfer and register access over the PCI bus. The NetFPGA’s Virtex-II Pro has two PowerPC hard core processors which have not been used in any reported NetFPGA projects. Putting these hard core processors to work can yield certain benefits. First, the PCI bus severely limits the throughput of packet transfers between the NetFPGA hardware and the host CPU. An on-chip processor would not suffer from this limitation. Second, running the software on an on-chip processor could significantly improve latency of packet transfer and register access. Third, the flexibility of an on-chip general-purpose processor with decent performance opens up new possibilities for experimentation, while also making it easier for non-hardware experts to try their ideas on the NetFPGA. Finally, since they are already taking up space on the chip whether they are used or not, we might as well use them. There are naturally some costs and limitations associated with using these processors, and one goal of this project is to attempt to quantify them. Another goal is to make it easier for others in the NetFPGA community to utilize these processors.

2 Related Work

There is a related NetFPGA project, called NetThreads[5], which implements two custom soft core processors for packet processing. The processors each have four hardware threads (no context switch), which are multiplexed in a round-robin fashion. If all four threads are active, then each issues an instruction every four cycles. The motivation for this arrangement is to avoid pipeline bubbles and to fully utilize the processor. The two processors share access to an input buffer, an output buffer, a synchronization unit, a data cache, and an external memory. Packets are written to the input buffer by hardware logic, and likewise read from the output buffer by hardware logic. The processors are responsible for copying data from the input buffer to the output buffer. The explanation for this arrangement was a two-port limitation on block RAMs. Their performance evaluation indicated that inter-thread synchronization was a significant bottleneck, and furthermore that the second processor provided negligible, if any, performance benefit. A side effect of their multi-threaded design which may or may not be of interest is that the ordering of packets may change due to synchronization effects and variable processing times. NetThreads is intended to be essentially a software-only platform, since they only distribute their bitfile and not their Verilog source. In sections 3 and 4, I will make some comparisons between NetThreads and this work.
3 Design

There are three primary goals which influenced my design. First, to maximize performance. Second, to replace the functions of the host CPU. And third, to make the design easy to use and integrate with custom hardware. Figure 1 shows a simple block diagram of the hardware design, indicating the interactions between modules. The PowerPC processor, labeled PPC405, currently uses only three interfaces: the instruction-side on-chip memory (ISOCM) interface, the data-side on-chip memory (DSOCM) interface, and the device control register (DCR) interface. The other interfaces of the PPC405 are not currently being used, most notably the processor local bus (PLB) and the external interrupt controller interface.

![Figure 1: PowerPC Wrapper Block Diagram](image)

The instruction memory is a single-port block RAM with a data width of 64 bits and currently a total size of 32KB. The data memory is a dual-port block RAM with a processor-side data width of 32 bits and a hardware-side data width of 128 bits. The data memory currently has a total size of 32KB, half of which is reserved for packet buffers, and half of which is used exclusively by the processor for global variables, the stack, and so forth. Both the instruction and data memories provide the processor with single-cycle access, so no cache is necessary. Byte writes to data memory are supported and packets are loaded into memory with the correct endianness. The hardware module which is labeled as MOVE PKTS is responsible for moving packets into and out of data memory. The packet data bus coming into and going out of the system has a width of 64 bits, which motivates the 128-bit data memory port size. The copy-in logic and the copy-out logic have access to the data memory port on alternate cycles. Thus, in contrast with the NetThreads design, I use a single buffer for both input and output. This takes a significant burden away from the processor when packet modifications can be done efficiently in place, which is the common case.

Packets are stored in fixed-size buffers. At any moment in time, only one of three parties (processor, copy-in logic, copy-out logic) has permission to access a particular packet buffer. This access permission is passed between them using three hardware FIFOs (in, out, and free). The FIFOs contain indices to packet buffers, and each FIFO is large enough to hold all the indices. The copy-in logic reads from the free FIFO and writes to the in FIFO. The copy-out logic reads from the out FIFO and writes to the free FIFO. The processor reads from the in FIFO and writes to the out FIFO. The processor currently does not have access to the free FIFO, but this will be changed in the future to allow the processor to generate packets, and to efficiently drop packets. The processor accesses these FIFOs through the DCR interface, which is managed by the DCR MODULE. DCR access is provided to software using the built-in instructions mtdcr and mfndcr.
The **REG MASTER** module gives the PPC405 processor access to the NetFPGA register bus, to read from and write to registers in other modules in the NetFPGA design. The **REGS** module contains registers associated with the PowerPC wrapper module, such as packet counts provided by the **MOVE PKTS** module and status registers written by the PPC405 processor. These registers can be read from or possibly written to by the host CPU. Figure 2 depicts how both the host CPU and the PPC405 processor can have master access to the NetFPGA register bus. The register bus is a ring of modules, and each master has an id. Masters place register requests on the bus and listen for and remove responses to their requests, identified by their id. One function of the **DCR MODULE** is to act as an intermediary between the processor and the **REG MASTER** module, and its job is to translate between the DCR interface and the NetFPGA register interface. There is one difficulty here, which is that the processor’s DCR interface has a 10-bit address, whereas the NetFPGA register bus has a 23-bit address. In order to access the full address space, accesses to the NetFPGA register bus are done using two DCR operations: the first to write the address to be accessed to a special register, and the second to actually perform the read or write to another special register. This also opens up the DCR address space for future special purposes.

![Figure 2: NetFPGA Register Bus with Two Masters](image)

A wrapper module contains all of these functions, exposing only four interfaces: incoming and outgoing packet interfaces, and incoming and outgoing register interfaces. This is a standard interface for a NetFPGA pipeline module, and can be easily integrated as shown in Figure 3. This is a significant simplification over instantiating and using the PPC405 primitive directly, which has 117 distinct interface signals.

![Figure 3: Integration of PowerPC Wrapper into NetFPGA Pipeline](image)

It is very likely that a designer will not want all packets to be processed in software, in which case a classifier of some sort should precede the PPC405 wrapper module and choose which packets are to be processed in software according to some application-specific criteria. This classifier is likely to be controlled by the processor itself at run-time. Because the processor may not be able to run at line rate, care should be taken regarding the back-pressure of flow control at the input to this module, which could adversely affect the preceding modules. This phenomenon is called head-of-line blocking, and can be fixed by placing a FIFO module in front of the PPC405 wrapper module which either enqueues packets or drops them depending on whether or not there is room for a full-sized frame, resulting in no back-pressure to the preceding modules. This functionality was not built into the PPC405 wrapper itself because it may not be desired in all configurations.
The developer is faced with a simple programming environment and API. The PPC405 processor runs a single thread of execution with no operating system. There are no interrupts and no context switches. The current API consists of a function to get an incoming packet index, a function to send an outgoing packet, a function to read from a register, a function to write to a register, and a pointer to the array of packet buffers. The function to get an incoming packet polls the in FIFO in a loop until a packet is available. The DCR instructions take several cycles, but running the processor at a faster clock rate (say, at 250MHz rather than the 125MHz of the user logic) can increase the polling rate. The NetThreads package comes with a similarly low-level API.

The development environment consists of a GNU powerpc-eabi cross-compiler toolchain, a Makefile template, a custom linker script which maps the instruction and data memory contents to specific memory regions, a simple boot code to initialize the stack, a small header file containing the above API, and custom tools that convert bin files into Verilog header files for initializing block RAMs during simulation. For the purposes of initializing block RAMs within pre-synthesized bit files, a bmm file is required mapping block RAM primitives to specific memory regions. This file is required during the synthesis process to track the physical locations of those block RAM instances, so that they can be initialized later using Xilinx’s data2mem tool and an elf binary. Xilinx’s Embedded Development Kit is not required or used in this process, especially since the newest version has dropped support for the Virtex-II Pro. If changes need to be made to the memory layout (such as increasing the size of instruction or data memory), then there are multiple places where that change should be reflected, including the Verilog code, the software, the bmm file, and the linker script. This process could be automated with a script in the future.

So far, only trivial applications have been written on this platform, such as choosing the output port based on the input port (this information is stored in packet metadata, as described in NetFPGA documentation). I have begun efforts to use LwIP[1][3], a lightweight TCP/IP stack which can be used without an operating system, on this platform.

Table 1 summarizes some differences of unequal significance between the NetThreads project and my PowerPC design.

<table>
<thead>
<tr>
<th>NetThreads</th>
<th>PowerPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>soft core</td>
<td>hard core</td>
</tr>
<tr>
<td>125MHz clock</td>
<td>125MHz or 250MHz clock</td>
</tr>
<tr>
<td>4 threads per processor</td>
<td>single thread</td>
</tr>
<tr>
<td>separate input and output buffers</td>
<td>single input/output buffer (zero copy)</td>
</tr>
<tr>
<td>off-chip memory</td>
<td>on-chip memory only (currently)</td>
</tr>
<tr>
<td>no register interface</td>
<td>register master and slave</td>
</tr>
<tr>
<td>bitfile-only distribution</td>
<td>easy integration with custom HW modules</td>
</tr>
</tbody>
</table>

Table 1: Comparison of NetThreads and my PowerPC design

4 Evaluation

I have done some initial performance evaluations of the host CPU vs. my PowerPC design. While I have not personally evaluated the NetThreads platform, I will compare with some of the numbers reported in their own evaluations. Table 2 contains a basic performance summary, including best case throughput and latency for packet transfers and register access latency. NetThreads latency was not reported, and register access latency does not apply to that platform.

<table>
<thead>
<tr>
<th></th>
<th>Host CPU</th>
<th>NetThreads</th>
<th>PowerPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best Case Throughput</td>
<td>186Mbps</td>
<td>&gt;0.7Gbps</td>
<td>3.48-3.95Gbps</td>
</tr>
<tr>
<td>Best Case Latency</td>
<td>60μs kernel, 120μs user</td>
<td>Unknown</td>
<td>192ns</td>
</tr>
<tr>
<td>Register Access Latency</td>
<td>1.7μs</td>
<td>N/A</td>
<td>256ns</td>
</tr>
</tbody>
</table>

Table 2: Performance Comparison

The throughput for the host CPU was measured using the iperf tool between two idle hosts connected
by a lightly-loaded Gigabit switch. One of the hosts had a NetFPGA loaded with the reference NIC design, and one of the NetFPGA interfaces was used for the test. Being a TCP goodput test, the result of 186Mbps is a conservative under-estimate of the raw data rate. Nevertheless, it is clearly inferior to both NetThreads and the PowerPC in this regard.

The best case latency for the host CPU was a rough measurement using the ping utility. The numbers are derived from three tests: IP routing by NetFPGA hardware, IP routing by the Linux kernel with the NetFPGA as a NIC, and IP routing by the userspace version of the Click Modular Router[4], with the NetFPGA as a NIC. The latencies reported by the ping tool showed about a 60µs difference between NetFPGA hardware and the kernel, and about a 120µs difference between NetFPGA hardware and userspace Click. Again, this test was performed on idle systems on a lightly loaded network, with minimum packet sizes, a best case scenario.

The host CPU register access latency was measured by repeatedly reading a NetFPGA register 10,000,000 times in a loop.

NetThreads reports that the raw throughput is at least 0.7Gbps (or 58K pkts/s) when simply copying maximum-sized frames from the input buffer to the output buffer. It may be more than that since they did not test it to the limit, at least not for maximum-sized frames. Performance is not so good with minimum-sized frames, but they did not provide concrete numbers for processing minimum-sized frames with both processors. They provided throughput data for running some specific applications, including UDHCP, a regular expression-based classifier, and NAT, for which their reported measurements range from about 500 pkts/s (UDHCP) to about 22,000 pkts/s (NAT under certain conditions). Unfortunately I don’t have data to compare with their application-level results yet.

The measurements for best case throughput, latency, and register access latency for my PowerPC design were done in cycle-accurate simulation. For these measurements, the CPU does nothing to the packet. It just receives the index from the in FIFO and sends it back out the out FIFO. The CPU was being clocked at 250MHz. The throughput for minimum-sized frames was 3.48Gbps, or 5.18M pkts/s. In this case, the CPU was fairly busy with its simple task. With the same test at 125MHz, the throughput was 3.6M pkts/s, clearly limited by the CPU. For maximum-sized frames, the throughput was 325K pkts/s, or 3.95Gbps. In this case the CPU spent most of its time polling for incoming packets, so the throughput is only limited by the ability of the MOVE PKTS module to copy packets into and out of memory. The best case latency was measured from entrance to exit of the wrapper module, with a minimum-sized frame and an empty queue. The register access latency was performed with a very small register chain.

The cost in FPGA resources of NetThreads and my PowerPC design is shown in Table 3. The resource utilization of a simple NIC is shown as a baseline for comparison. The PowerPC design adds a mild 7% in FPGA slices, and a more sizeable 15% in block RAMs. This is with a total of 64KB of block RAMs for instructions and data. Of course, memory requirements depend on applications, so this allocation is not fixed. A 32KB instruction memory is large enough for LwIP. NetThreads adds 29% in slices and 64% in block RAMs. The higher resource utilization comes at no surprise, since these are soft processors.

<table>
<thead>
<tr>
<th>Block RAMs</th>
<th>Nic</th>
<th>PPC</th>
<th>NetThreads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>7%</td>
<td>22%</td>
<td>71%</td>
</tr>
</tbody>
</table>

Table 3: FPGA Resources Summary

5 Conclusion

This design employing the hard core PowerPCs seems to demonstrate considerable potential for decent packet processing performance. An essential feature is the high memory throughput provided by the 128-bit hardware memory port and zero-copy processing by software. With the primitives of packet transfer and register access, the embedded PowerPC core can assume the basic functions of the host software. It still does not have all the features and resources of a workstation, but is sufficient for a wide variety of applications. A TCP implementation such as in LwIP would be very valuable. If more memory is desired, a memory controller could be used to connect the NetFPGA’s 64MB of DRAM to the Processor Local Bus. Furthermore, a featureful OS could be installed, such as embedded Linux.
would not be ideal for pure packet processing purposes, but since there are two PowerPC hard cores, one could run embedded Linux with the 64MB of external memory while the other could be used for higher-performance packet processing. I plan to share this design with the NetFPGA community so that others can take advantage of the provided flexibility with ease.

References