Abstract—A short time-to-market is very important for a chip, and verification takes the most (about 70%) of its design time. Network Interface Controller (NIC) is a key component for a supercomputer and other computing systems. To reduce verification time for such a market-demanding product plays a great role in relevant system design. In this paper, a functional verification accelerator NICFlex is presented for a Register Transfer Level (RTL) NIC design. NICFlex accelerates verification process by both a software part and a hardware part. The software part runs as a simulation thread, and the hardware part is mapped into Field Programmable Gate Array (FPGA) logic together with a NIC wrapper. Compared to a conventional simulation verification method using ModelSim, NICFlex can accelerate the functional verification of an RTL NIC design for hundreds of times or more. With extension, NICFlex is promising for any functional verification acceleration of a generic RTL design.

I. INTRODUCTION

With Very Large Scale Integrated Circuit (VLSI) design increasing its complexity, its functional verification under target application system has become mandatory for a successful VLSI design. In the functional verification flow, emulation or prototyping has recently been widely adopted as an integral part. Despite their verification speed and relatively widely functional test coverage, first both emulation and prototyping have a limitation in reducing the time-to-market for they can only work at the late design stage. Therefore, if some design errors or defaults are found by the emulation or prototyping verification, it usually takes a very long period to run the emulation or prototyping again with the bug-fixed. Secondly, there are some risks lying in debugging a VLSI design with an emulation or prototyping application system. For example, one can not guarantee the functional correctness of an application system until the successful chip emulation or prototyping. Once there is a bug, the designer has to search for its possible locations in both the design and the emulation/prototyping application system.

NIC is a key component for a supercomputer and other computing systems, and now it becomes a more complex System-on-a-Chip (SoC), then the above problems for it become more serious. For instance, with a cycle accurate simulation, one can only get about 200Hz performance during the verification process for an RTL NIC design. To alleviate such disadvantages, we presented a functional verification accelerator NICFlex.

II. RELATED WORK

To implement functional verification of an RTL NIC design, there has been a lot of research. These research can be grouped into two categories:

- Independent verification method;
- In-system verification method.

In the first kind of method, interaction effects between a NIC and its application environment are not considered; Hence such kind of verification is preliminary, not self-contained, and has been excluded in our verification acceleration.

The second kind of method can be further classified into two sub-categories: 1) simulation based verification (i.e., simulation with a simulator); and 2) actual system based verification.

Simulation based verification is based on analytical models or simulation models. There are many analytical models for network interconnects[1], [2], [3], [4], [5]. By simulation with these analytical models, exploration space of an RTL NIC design is reduced. However, accurate analytical models are very difficult to obtain for a complex interconnect system. Additionally, simulation models for network interconnects are at various levels of detail, accuracy, and speed[6], [7], [8], so it is difficult to choose a proper model in the simulation. Furthermore, simulation based verification performs detailed simulation without basing on an actual system hardware.

N. L. Binkert et al. developed a simulation environment specifically targeting at network systems with closer NIC attachment to CPU and memory[9]. However, it can only simulate one process and does not focus on a whole design space exploration. Another interesting actual system based verification is an experimental system to study different offload scenarios with an advanced NIC plugged on an I/O bus[10]; Its authors claimed the emulation is faithful and can guide design of the offload engine software; However, its main concern is not an RTL NIC design. D. Hegarty and S. McDonald presented a different actual system based verification; i.e., an FPGA-based configurable network interface system[11]; The platform offered highly efficient solutions for a wide range of applications, but it is just a prototype for one case.

For actual system based verification, system cost is also too high to be affordable for common users. In addition, a NIC designer or system engineer wants to know NIC performance even when an actual system is unavailable (e.g. before delivery.
of a whole system), actual system based verification does not make for this tend.

Our idea is a little similar to those in RAMP project[12] and PROTOFLEX[13]. All these former projects used FPGA technology to facilitate design space exploration and to provide research infrastructure; They focused on multiprocessor architecture and its related software development. However, except for a design space exploration, NICFlex focuses on an RTL NIC functional verification acceleration. There is a similar work by G. Kalokairinos et al.[14]. In the work, a NIC has been implemented on a Dinigoup DN6000K10SC board. Compared to our work, their adopted FPGA Virtex II Pro is with less logic elements, and they did not consider a whole system. Their work also focuses only on a concrete implementation which is not generic. Especially, it does not run together with a CPU simulator, thus it lost flexibility.

III. NICFLEX ARCHITECTURE

NICFlex consists of a DN8000K10PCI board and a CPU simulator/emulator. CPU simulator is implemented in a host computer. It is used to generate verification stimuli for a NIC, the stimuli is generated based on a target network protocol. A target NIC and a NIC wrapper are mapped in the board.

NICFlex architecture is shown in Fig. 1, where its current version is in the dotted block. In Fig. 1, Tbus interface is an interface slightly modified from QL5064 back-end interface[15]. For convenience, we adopt First-In First-Out (FIFO) as the Tbus interface.

![Fig. 1. NICFlex architecture](image)

IV. KEY DESIGN ASPECTS

A. Programming Model

NICFlex uses a packet oriented programming model. A generic packet format makes it easy for the programming for various network behavior simulation. The generic packet format is shown in Fig. 2. Both its command and data are 64-bit.

![Fig. 2. Packet format in NICFlex](image)

NicFlex packet type is determined by the cmdi(command) segment and datasize segment.

The cmd segment provides current active command, its types and encodings are shown in Table I. From Table I, we can see that bit 3 indicates whether the command is a read or write operation except a response command; Bit 2 indicates whether the command is a block operation; Bit 1 indicates whether the command is the last one for current transfer; Bit 0 shows whether the command is a response. All bits are “1” active.

<table>
<thead>
<tr>
<th>Command type</th>
<th>Bit coding and meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block read</td>
<td>0 1 0/1 0</td>
</tr>
<tr>
<td>Non-block read</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>Block write</td>
<td>1 0/1 0</td>
</tr>
<tr>
<td>Non-block write</td>
<td>1 0 1 0</td>
</tr>
<tr>
<td>Block response</td>
<td>X(don’t care) 1 0/1 1</td>
</tr>
<tr>
<td>Non-block response</td>
<td>X(don’t care) 0 1 1</td>
</tr>
</tbody>
</table>

When an active command indicated by cmdi segment is a block operation, the data number is (datasize+1) cache lines and bten is equal to “11111111b” indicating all eight data bytes are valid. When the active command is a non-block operation, the data number has another meaning; i.e., there is only one 64-bit data transfer, the 64-bit data has (datasize+1) valid bytes, and each “1” in bten indicating a corresponding valid byte in the 64-bit data.

Segment addr provides the starting address for the current access. The 40-bit address is global consistent address oriented. Segment reqnum indicates the active request number and segment srcnum indicates which source (node) issues the command. Bit 63 in the packet header is not used and its default value is zero.

B. NIC Wrapper

The NIC wrapper bridges the Tbus interface to the mapped NIC. Because Tbus is a simple FIFO interface, thus the simulator can easily communicate with the RTL mapped NIC by the NIC wrapper. The NIC wrapper architecture is shown in the dotted block in Fig. 3, where clk_inner is clock signal for the mapped NIC, and clk_t is clock signal for the Tbus interface.

![Fig. 3. NIC wrapper in NICFlex](image)
C. CPU Simulator

CPU simulator is used to generate testbench/stimuli for a target NIC’s verification. Based on the packet format defined above, each CPU simulator running in a host can generate a testbench from 256 nodes. These nodes can be abstracted from multicore processors or CPUs in a distributed system. For the simulation size is determined by segment reqnum and srcnum, it is easily scaled to much bigger with a small modification on the two segments. The CPU simulator will generate the testbench/stimuli data used for both a traditional simulation verification and NICFlex verification.

V. EXPERIMENTAL RESULTS AND ANALYSIS

To evaluate our NICFlex accelerator, we referenced a set of network operations commonly used in a distributed and parallel applications as benchmarks[16] and compared the verification results from a conventional simulation method and those from our NICFlex accelerator.

A. Implementation

For the hardware design, synthesis is by Synplify Pro 8.4 from Synplicity; Place-and-route and timing analysis are by Xilinx Integrated Software Environment (ISE) software (Version 8.1i). For conventional verification by simulation, we used Mentor Graphics ModelSim SE plus (Version 6.1b).

We developed hardware part of NICFlex in Verilog HDL (Hardware Description Language) and software simulation testbench on System Verilog. Both hardware design and simulation using ModelSim are done on a PC with a 2.00GHz Pentium IV CPU and 1GB memory.

In Table II, we show the resource usage of NICFlex hardware (i.e., the target NIC and NIC wrapper) in a Xilinx XC4VLX160(FF1513) mounted on the Dini board; Its total equivalent gate count is about 1160000. Please note that the figures shown in Table II may vary slightly with other tools or different versions.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Number</th>
<th>Total utilization (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>25544</td>
<td>37</td>
</tr>
<tr>
<td>RAM16</td>
<td>79</td>
<td>27</td>
</tr>
<tr>
<td>DCM</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>BUFG</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>Bonded IOBs</td>
<td>220</td>
<td>22</td>
</tr>
</tbody>
</table>

We used GCC (Version 4.1.0) as C compiler to develop our simulator, Application Programming Interface (API), and driver for NICFlex on a Fedora core 5 Linux.

In a total view, NICFlex implementation is shown in Fig. 4, where the software part is shown in a dotted block. The simulator is running on a host computer with two 733MHz Pentium III CPUs and 512MB memory.

In the API, we adopted two basic functions shown in Fig. 5.

```c
int bar_write_dword(unsigned int barnum, 
                    unsigned int byte_offset, 
                    dword data)
{
    dword* dword_ptr = (dword*)(ptr_base_bar[barnum] + byte_offset);
    *dword_ptr = data;
    return 0;
}

int bar_read_dword(unsigned int barnum, 
                    unsigned int byte_offset, 
                    dword* data)
{
    dword* dword_ptr = (dword*)(ptr_base_bar[barnum] + byte_offset);
    *data = *dword_ptr;
    return 0;
}
```

We use two text files respectively to store the testbench and the verification result. Such a method can reduce development efforts in simulator. The format of the testbench file is shown in real line block of Fig. 6. For each line in the testbench file on the right, we explain its corresponding meaning in the dotted block on the left. The file format of the verification result is the same as that from ModelSim, which is customized by users.

```plaintext
... 64h0a947c0000002222 read0 1
64h0a947c0000004444 read1 1
... 64h1057ff3c3cca1370 non-block-write0 1
64h0706050403020100 data0 1
64h1067fc0283d2e0 block-write0 1
64h88c888888888888 data1 1
64h94932919088e8d  data2 1
64h9c9b9a99998979695 data3 1
64ha4a3a2a1a0999e9d data4 1
...  ...
```

After we get the result file (i.e., a simple text file), we compare it with the result file gotten from ModelSim with the same testbench, and check whether the verification is passed successfully. Because we focus on verification acceleration, we
assume all verification passed successfully and collect timing information to run these testbenches. The same testbench is used in simulation with ModelSim for comparison. The testbench is designed based on concrete network behaviors with our programming model.

B. Experimental Results

To simplify measure on verification speedup, we designed a set of specific verification cases. That is, we classified the verification into three different groups based on the command type. In each group, we test four cases, i.e., with command number $N_c = 10, 100, 1000, 10000$ respectively. After collection on the verification time $T_{NICFlex}$ using NICFlex for these verification and the verification time $T_{ModelSim}$ using ModelSim SE (Version 6.1b) on a PC with a 2.00GHz Pentium IV CPU and 1GB memory mentioned above, the results are compared to evaluate the verification acceleration.

The verification cases and their verification time results are shown in Table III.

<table>
<thead>
<tr>
<th>Command</th>
<th>$N_c$</th>
<th>$T_{ModelSim} (\mu s)$</th>
<th>$T_{NICFlex} (\mu s)$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>10</td>
<td>1700000</td>
<td>734</td>
<td>2316.1</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>16800000</td>
<td>7223</td>
<td>2325.9</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>16530000</td>
<td>69985</td>
<td>2361.9</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>163260000</td>
<td>687627</td>
<td>2374.2</td>
</tr>
<tr>
<td>Write</td>
<td>10</td>
<td>2400000</td>
<td>698</td>
<td>3438.4</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>24300000</td>
<td>5552</td>
<td>4376.8</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>23620000</td>
<td>54529</td>
<td>4331.6</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>216710000</td>
<td>517612</td>
<td>4186.7</td>
</tr>
<tr>
<td>DMA</td>
<td>10</td>
<td>2900000</td>
<td>3608</td>
<td>803.8</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>28900000</td>
<td>35763</td>
<td>808.1</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>27870000</td>
<td>332816</td>
<td>837.4</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>268030000</td>
<td>3105781</td>
<td>863.0</td>
</tr>
</tbody>
</table>

C. Analysis

From Table III we can see that our NICFlex accelerator can speedup the functional verification for more than two orders. In a detail analysis, the speedup for write command is about 4000 which is in the middle. The speedup for read command is about 2300 which is in the middle. The speedup for DMA command is about 800 which is the lowest; the speedup for read command is about 2300 which is in the middle. The difference is due to the different real CPU cycles to execute each kind of commands.

For read and DMA commands, when the number of commands increases, the speedup increases quite slowly. One important reason for this phenomena is that the schedule of Operating System (OS) can dominate in a certain extent for a small number of commands, and for a great number of commands, the role of schedule will decrease a little.

A big jump in verification speedup appears when $N_c$ of write increases from 10 to 100, which is attributable to that each write command needs a data or several data transfer (a block write for instance) correspondingly.

Based on the increasing speedup along with command-number increase, we can conclude that the PCI bus bandwidth is enough for our NICFlex without any negative effects on the verification, and though simulation with ModelSim is running on a platform with a higher frequency CPU and a bigger memory than NICFlex’ host, our performance evaluation on the different platforms is credible.

VI. CONCLUSION

The NICFlex accelerator bridges the NIC designer and software programmer with a common platform, enables the hardware designer to verify an RTL NIC design as early as possible before its delivery.

Our initial experiments demonstrated NICFlex has an interesting and exciting speedup compared to a conventional verification method using ModelSim. Except for these successful achievements on our research objectives, we are on the way to extend NICFlex for common functional verification acceleration of a generic RTL design.

ACKNOWLEDGMENT

This work is supported by Natural Science Foundation of China (Grant No. 60403025) and by Le Programme de Recherches Avancées Franco-Chinois (PRA SI04-04 grant).

REFERENCES