Controller for Network Interface Card on FPGA

Suchita Kamble, N. N. Mhala

Abstract: ***The continuing advances in the performance of network servers make it essential for network interface cards (NICs) to provide more sophisticated services and data processing. Modern network interfaces provide fixed functionality and are optimized for sending and receiving large packets. Network interface cards allow the operating system to send and receive packets through the main memory to the network. The operating system stores and retrieves data from the main memory and communicates with the NIC over the local interconnect, usually a peripheral component interconnect bus (PCI). Most NICs have a PCI hardware interface to the host server, use a device driver to communicate with the operating system and use local receive and transmit storage buffers. NICs typically have a direct memory access (DMA) engine to transfer data between host memory and the network interface memory. In addition, NICs include a medium access control (MAC) unit to implement the link level protocol for the underlying network such as Ethernet, and use a signal processing hardware to implement the physical (PHY) layer defined in the network. To execute and synchronize the above operations NICs also contents controller whose architecture is customized for network data transfer. In this paper we present the architecture of application specific controller that can be used in NICs.***

Keywords: ALU, Fast adder, Network interface card, RAM, ROM, Universal shit register, Instruction decoder.

I. INTRODUCTION

As the performance of network servers increases, network interface cards (NIC) will have a significant impact on a system performance. Most modern network interface cards implement simple tasks to allow the host processor to transfer data between the main memory and the network, typically Ethernet. These tasks are fixed and well defined, so most NICs use an Application Specific Integrated Circuit (ASIC) controller to store and forward data between the system memory and the Ethernet. However, current research indicates that existing interfaces are optimized for sending and receiving large packets [1]. Experimental results on modern NICs indicate that when frame size is smaller than 500-600 bytes in length, the throughput starts decreasing from the wire-speed throughput. As an example, the Intel PRO/1000 MT NIC can achieve up to about 160 Mbps for minimum sized 18-byte UDP packet (leading to minimum sized 64-byte Ethernet packet). This throughput is far from saturating a Gigabit Ethernet bidirectional link, which is 1420Mbps. Recent studies have shown that the performance bottleneck of small packets traffic is because that there is not enough memory bandwidth in current NICs [2] [3]. In a back-to-back stream of packets, as packet size decreases the frame rate increases. This implies that the controller in the NIC must be able to buffer larger number of incoming smaller packets. If the controller does not provide adequate resources, the result will be lost packets and reduced performance. The other reason for this problem is that current devices do not provide enough processing power to implement basic packet processing tasks efficiently as the frame rate increases for small packet traffic. Previous research has shown that both increased functionality in the network interface and increased bandwidth on small packets can significantly improve the performance of today's network servers. New network services like network interface data caching improve network server performance by offloading protocol processing and moving frequently requested content to the network interface. Such new services may be significantly more complex than existing services and it is costly to implement and maintain them in nonprogrammable ASIC-based NICs with a fixed architecture. Software-based programmable network interfaces excel in their ability to implement various services. These services can be added or removed in the network interface simply by upgrading the code in the system. However, programmable network interfaces suffer from instruction processing overhead. Programmable NICs must spend time executing instructions to run their software whereas ASIC based network interfaces implement their functions directly in hardware. To address these issues, an intelligent, configurable network interface is an effective solution. A reconfigurable NIC allows rapid prototyping of new system architectures for network interfaces [4]. The architectures can be verified in real environment, and potential implementation bottlenecks can be identified. Thus, what is needed is a platform, which combines the performance and efficiency of special- purpose hardware with the versatility of a programmable device [5]. Architecturally, the platform must be processor-based and must be largely implemented using a configurable hardware. An FPGA with an embedded processor is a natural fit with this requirement [6]. Also, the reconfigurable NIC must have different memory interfaces providing including high capacity memory and high speed memory for adding new networking services [1].

II. FUNCTIONALITY OF NETWORK INTERFACE CARD

Network interface cards allow the operating system to send and receive packets through the main memory to the network. The operating system stores and retrieves data from the main memory and communicates with the NIC over the local interconnect,

Manuscript received on July, 2012. (Required Details)
Suchita Kamble, Departemnt Name, Institute Name
N. N. Mhala. Departemnt Name, Institute Name

usually a peripheral component interconnect bus (PCI).

Most NICs have a PCI hardware interface to the host server, use a device driver to communicate with the operating system and use local receive and transmit storage buffers. NICs typically have a direct memory access (DMA) engine to transfer data between host memory and the network interface memory. In addition, NICs include a medium access control (MAC) unit to implement the link level protocol for the underlying network such as Ethernet, and use a signal processing hardware to implement the physical (PHY) layer defined in the network [4]. To send packets, the host processor first instructs the NIC to transfer packets from the main memory through a programmed I/O in step 1. In step 2, the NIC initiates DMA transfers to move packets from the main memory to the local memory. In step 3, packets need to be buffered in the Buffer-TX, waiting for the MAC to allow transmission. Once the packet transfer to local memory is complete, the NIC sends the packet out to the network through its MAC unit in step 4. Finally, the NIC informs the host operating system that the packet has been sent over the network in step 5. The steps for sending packets from the main memory to the network are shown in Figure 1 [1].

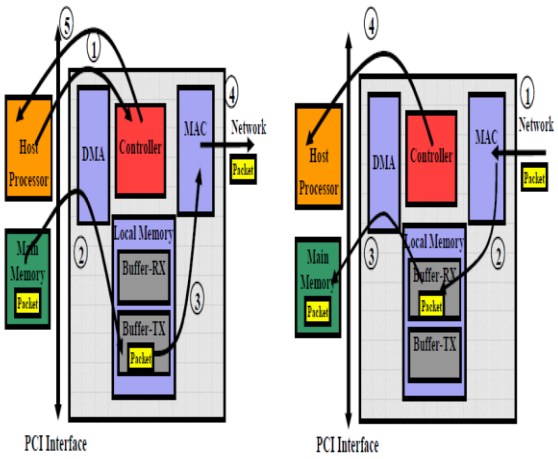

Figure 1 Network Interface card Functionality

III. ARCHITECTURE OF CONTROLLER

The implementation of processor that consists of several blocks such as ALU, instruction fetch and decoding logic, data memory, program memory, control unit, data registers and interrupt controller. Our objective is to designed processor such that data can continuously stream between the MAC processor and operating system. To achieve the continuous stream, data memory is used in the processor to hold the block of data to be transferred between MAC and operating system. In addition, we can heavily apply a clock-gating scheme in order to achieve low power consumption. The features of controller used in the network interface card are as follows.

- 8 bit Processor (8 bit Data bus)
- 8 bit ALU for performing arithmetic and logical operations on signed and unsigned numbers such as Addition, Subtraction, AND, OR, NOT, 1's & 2's Complement and Universal shift register.
- 31 Registers 8 bit each for storing partial results during operation.
- Address and data register to buffered store current address and data.
- Program counter to hold the address of the current instruction.
- Instruction decoder
- control unit
- Data Memory
  - 4 Kbytes,
  - 8 bit data lines
  - Store data during transmitting and receiving.
  - Stack memory
- Program Memory
  - 4 kbytes,
  - 8 bit data lines
  - Store program (instructions)
- Interrupt Controller
  - Hardware interrupts
  - 2 hardware interrupts (one for DMA and other for MAC)
  - Interrupts to inform the status of controller
  - Software interrupts
    - 3 software interrupt instructions to transfer the control of program

IV. INSTRUCTION SET

The controller consist of processor whose instruction set is customize to processes the network data. The instructions in the controller are as follows

- Binary addition and subtraction.
- Bit-wise logical AND, OR, and NOT.
- Compare, Shift / rotate left and right logical.
- Bit-manipulation commands: set, clear, test, and flip.
- General purpose registers transfer.
- Control Instructions such as software interrupts and Call and Ret subprograms.
- Addressing Modes
  - Direct addressing mode
    - Address of the data specified in instruction
    - Data transfer
  - Indirect addressing mode
    - Address not specified in instruction
    - Block data transfer
  - Implicit addressing mode
    - Used for bit manipulation commands
    - Call & Ret instructions

The block diagram of the controller is shown in figure 2.

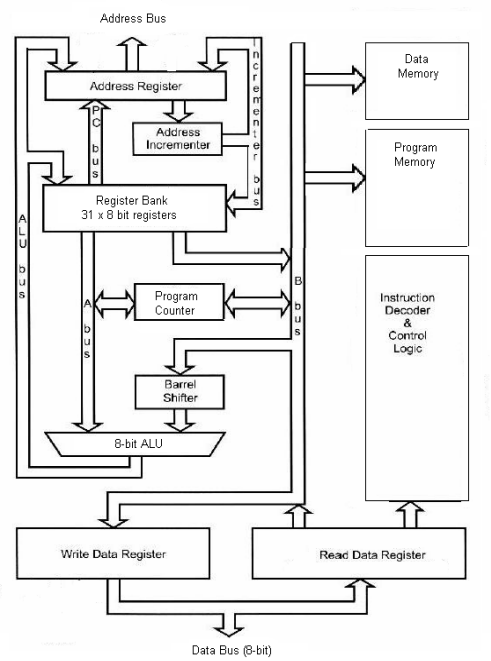

Figure 2: Block Diagram of Controller

V. IMPLEMENTATION

Xilinx Synthesis tool (VHDL) and ModelSim Simulator can be used for implementation of the controller [5] [7] [8]. Xilinx ISE Design Tool is the ideal downloadable solution for FPGA and CPLD design offering HDL synthesis and simulation, implementation, device fitting, and JTAG programming.

Following are the steps of designing digital circuits in FPGA using VHDL [5].

1. Start
2. Use a schematic or a Hardware Description Language (HDL) to design the logic block.
3. Use the tool to synthesis the logic block.
4. Use Timing Analyzer to find and optimize the critical path(s).
5. Use Floor planner and FPGA Editor to optimize the area and routing length of the block.
6. Use FPGA Editor to map the function block into a hard macro.
7. Delay-matching hard macro, which has a comparative delay and dimension with the function block.
8. Finish

VI. CONCLUSIONS

We presented the architecture of processor that consists of several blocks such as ALU, instruction fetch and decoding logic, data memory, program memory, control unit, data registers and interrupt controller. Clock gating scheme reduces static and dynamic power consumption in the circuit. The processor can be implemented using VHDL (Xilinx Synthesis Tool and / or NI Modelsim Simulator and / or Altera Quartus II Design Software) and targeted for implementation in FPGA. Thus a reconfigurable NIC allows rapid prototyping of new system architectures for network interfaces. The architectures can be verified in real environment, and potential implementation bottlenecks can be identified.

REFERENCES

1. Toshio Fujisawa, et al, "A Single-Chip 802.11a MAC/PHY With a 32-b RISC Processor", in IEEE Journal Of Solid-State Circuits, Vol. 38, No. 11, November 2003.
2. J. R. Allen, et al, "IBM PowerNP network processor: Hardware, software, and applications," in IBM Journal of Research & Development, Vol. 47, No. 2/3 March/May 2003.
3. Xiaoning Nie, et al, "A New Network Processor Architecture for High-speed Communications," in IEEE Workshop on Signal Processing Systems, 1999.
4. H. Peter Hofstee, "Power Efficient Processor Architecture and The Cell Processor," in Proceedings of the 11th International Symposium on High-Performance Computer Architecture, 2005.
5. D. L. Perry, " VHDL", Tata Mcgraw Hill Edition, 4th Edition, 2002.
6. C. Maxfiled, "The Design Warriors Guide to FPGAs", Elsevier, 2004.
7. J. Bhaskar, " VHDL Primer", Pearson Education, 3rd Edition, 2000.
8. J. Bhaskar, " VHDL Synthesis Primer", Pearson Education, 1st Edition, 2002.