Design of a Reconfigurable Cluster Computer Node

Josué Paulo José de Freitas, Alexandra Aguiar, Márcio Kreutz,
Tatiana G. S. Dos Santos, Rafael dos Santos
Universidade de Santa Cruz do Sul - UNISC - Santa Cruz do Sul, RS
{josuefreitas,xandaaguiar}@mx2.unisc.br, {kreutz, tatianas,rsantos}@unisc.br

Abstract

The goal of this work is to design the first prototype of a reconfigurable cluster computer node. The ACR (Agregado de Computadores Reconfigurveis, in portuguese) project pursue, among others, the objective of aggregating the technology of reconfigurable logic devices with computer clusters. An ACR cluster is composed of reconfigurable nodes that exchange messages through an embedded interconnection network enabling the distributed processing of applications on reconfigurable cluster nodes. This paper presents how the first version of an ACR cluster node was developed.

1. Introduction

The main goal of the ACR project is to integrate the spatial processing of reconfigurable logic devices [1] into distributed/parallel processing models by using message passing methods to communicate across reconfigurable nodes.

Most of the traditional clusters are based on off-the-shelf conventional PC (Personal Computer) architectures because they are widely available at relatively low cost. However, for a large number of applications, only a small subset of the whole instruction set is really needed [2] and, similarly, only a little portion of the hardware available in these systems is used. On top of that, a general purpose microprocessor is developed to obtain the best performance based on an average workload, thus not offering optimized resources for specific applications. High power consumption, physical space, heat dissipation, and other aspects can also be considered of negative impact.

In this context, the first node was designed integrating the communication stack into an FPGA with a FemtoJava microcontroller generated by the Sashimi tool [3]. A free and open source communication stack was modified to enable sending data through the User Datagram Protocol (UDP)[4]. The integration of the communication system within the FPGA with the processor allows for a significant reduction of the communication latency since all communication layers are implemented in hardware.

2. FemtoJava microcontroller

The FemtoJava microcontroller is an ASIP generate from a Java specification by the Sashimi tool. The Java source code, developed according to some patterns, is compiled like any other Java application and analyzed by the Sashimi tool. The tool selects for the FemtoJava microcontroller only the necessary logic and arithmetic units to the execution of the target application, saving hardware, and then generates the memory initialization files (mif) for the RAM (data) and ROM (instructions).

3. Comunication stack

The communication stack is a conceptual model where the communication protocols are organized upon, for example a TCP/IP or UDP/IP stack. The communication stack used in the first version of the node is an UDP/IP stack. Originally it provided only support to receive UDP packets, but we designed and implemented a UDP send block and now UDP packets can also be sent through the network. In this first prototype the stack works in full-duplex mode at 10MB/s.

All layers are implemented in hardware. The Application layer, represented by the FemtoJava Microcontroller and the application running on it, make the communication with the Transport Layer. On the other side, the Data link Layer communicates with the Physical Layer (PHY), that is ASIC already implemented on the prototyping board. Figure 1 shows how the whole system looks like, including the above explanation.

4. Node Structure

The node core is composed by three structures: the communication stack, the FemtoJava microcontroller and the integration layer. The integration layer is the structure that coordinates the data receive and send processes to/from the communication stack and the microcontroller.

The integration layer is composed by two FIFO (First In First Out) structures and three Finite State Machines (FSMs). The Sending FIFO stores the outgoing data generated by the FemtoJava Microcontroller to be send trough the UDP/IP stack. The Receiving FIFO stores the incoming data from the UDP/IP stack. The FSMs have to make the coordination of the reading and writing processes of the FIFOs and to generate the signals (interruptions) to the communication stack and the microcontroller, respectively. Figure 1 shows the structure of the node.

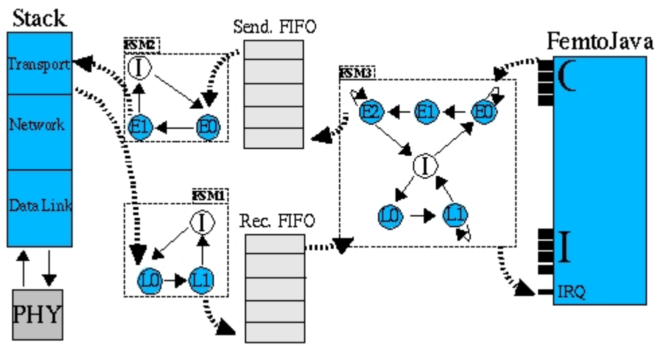

Figure 1. Node structure and the intra-node data and control flow

Receiving FSM (FSM1): this FSM receives a signal from the stack, notifying that a data is arriving to the Receiving FIFO. A *we* (write enable) signal is asserted telling the Receiving FIFO to proceed with the reading from the bus.

Sending FSM (FSM2): this FSM has as input signal the inverted value of the output port *Empty* of the Sending FIFO. Every time that this FSM is in the Idle state and the *Empty* signal is low, the FSM will generate the signals to proceed the reading from Sending FIFO and tell the communication stack that a valid data will be available on the bus.

Due to the full duplex operation of the stack, which allow sending and receiving at the same time, was possible to decouple into two different FSMs the integration between the FIFOs and the communication stack.

FemtoJava-FIFOs FSM (FSM3):this FSM is dedicated to synchronize the reading, from the Receiving FIFO, and the writing, to the Sending FIFO, of the data to and from the FemtoJava microcontroller.

When a data is available on the Receiving FIFO, the FSM generates an interruption signal, which is connected to one Interruption Request (IRQ) port, making the IRQ routine of the application proceed with the reading from the input port. After that, the data is processed and a record request, through the second output port, is made. The processed data is recorded in the microcontroller first output port, and finally written into the Sending FIFO by this FSM.

The number of states of each FSM is based in the number of cycles necessary for each operation.

5. Results

The node structure is now validated by simulation. A testbench simulating the behavior of a PHY was manually developed. This testbench simulate the receiving of an Ethernet Frame with an IP datagram on it and an UDP packet into the IP datagram. The data field of the UDP packet contains the data that will be processed by the FemtoJava microcontroller.

The whole system has maximum clock speed of 64.91MHz and uses 3.089 logic elements and 6.272 memory bits in a Cyclone EP1C20 FPGA (20.000 LEs and 294.000 Memory bits). Some values like the number of logic elements and memory bits may vary in the next version of the node, since more complex applications will require more RAM and ROM space in the FemtoJava microcontroller and new features will be added to the communication stack.

References

[1] Dehon, A.; Wawrzynek, J., Reconfigurable Computing: What, Why, and Design Automation Requirements ?, in Proceedings of the 1999 Design Automation Conference, pp. 610-615. June 1999.

[2] Patterson, David A.; Ditzel, David R.: The Case of Reduction Set Computer, ACM SIGARCH Computer Architecture News - Volume 8 , pp. 25 ? 33. Issue 6. 1980.

[3] ITO, Sergio Akira. Projeto de Aplicaes Especficas com microcontroladores Java Dedicados. Porto Alegre: CPGCC UFRGS. 2000.

[4] Postel, J.: RFC 768: User Datagram Procotol. 1980.