Isolated Network Model based on Cell for Software Radio System

Qingguo Zhou*, Shuwei Bai, Nicholas Mc Guire
Distributed & Embedded System Lab (DSLab), Engineering Research Center of Open Source Software and Real-time Systems Ministry of Education
Lanzhou University
Lanzhou, P.R.China
baishuwei@gmail.com, zhouqg@lzu.edu.cn

Jianwen Chen
China Research Laboratory
IBM
Beijing, China
jianwenc@cn.ibm.com

Abstract — In the paper, a novel network communication model based on Cell, which is a multi-core processor system, is presentment. And the I/O data transmission performance of the novel system will be showed in the paper too. In order to meet the high computation capacity request, the multicore platform has been adopted in the software radio system. But the research result presents that the data I/O transmission becomes the bottleneck for the software radio system. To resolve the problem, the paper redesign the network model based on Cell system, moving the protocol stack and net device driver components to SPU from PPU. The novel model cannot only release the PPU resource, but also has efficient I/O transmission performance.

Index Terms — Software Radio, Cell, I/O communication, Isolation mode

I. INTRODUCTION

In order to meet the high computation and adaptability requirements, multicore platform has been widely used for the advanced SR due to their high performance of signal processing. Compared with the single-core processor, the multicore chips do not run that fast, but they promise a higher overall performance by handling more work in parallel. Thus the multicore platforms become more prevalent in vendor’s solutions. Many system and semiconductor companies are developing platforms involving several cores (MCU, DSP, IP, and etc) on a single chip. STI Cell [2] is the latest state-of-the-art multicore processor designed by the joint adventure of Sony, Toshiba and IBM (STI). With the support of SIMD (Single Instruction Multiple Data) instructions and the powerful synergistic processor elements (SPEs), Cell processor can provide significantly high computation capacity, which is appropriate for SR systems. Especially, the BEI(Cell Broadband Engine interface) supports two Rambus FlexIO I/O bus interfaces, and the IOIF and BIF protocols are supported by two I/O interfaces. Both the IOIF and BIF protocols provide fully pipelined, packet-transaction interconnection using credit-based flow control. The two IO controllers are documented as supporting a peak combined input speed of 25.6 GB/s and a peak combined output speed of 35 GB/s [3].

However, for Software Radio system, the data communication between the Baseband processing board and the RF module is characterized by high throughput. The system needs to ensure that the I/O interfaces can support the required throughputs. For example, in the WiMAX PHY system with 20Mbps throughput for both uplink and downlink, if an 8 bit DAC is used, the output of downlink is about 658Mbps. For 3 sectors, the overall downlink throughput is about 1.975Gbps. On the Cell platform, the PPU is a general purpose processor and is not efficient on the network protocols parsing and packets handling. It is observed that in the implemented baseband system, the I/O data communication between the baseband processing module and the RF module consumes a mass of system resources for receiving, parsing and sending packets and it becomes the bottleneck for the SR system on Cell multicore platform.

In this paper, we present an efficient SPU based network module on the Cell platform. With this module, all the network packets parsing and packaging is handled by the SPU, the PPU resources are released and the system performance will be improved greatly.

The rest of this paper is organized as follows. Section II will briefly describe the architecture of the isolated network communication system based on Cell multicore platform. Section III will introduce the design and implementation of the network model In Section IV, the system performance and result analysis are provided. Section V presents the conclusions and the future work.
II. SYSTEM ARCHITECTURE

In the conventional network communication architecture, CPU should do all the work to handle the network packages, writing data to socket, moving data from user space to the kernel space by calling system call, encoding and dividing the data packages in the different protocol levels, and writing the data to the cable. The conventional network communication architecture is showed in Figure 1. All the tasks are executed by the same CPU, the network communication tasks and the intensive computation tasks will interact in a very complex way. In the single-core system, as the limitation of CPU resource, the problem cannot be resolved. But on the asymmetric multi-processor (AMP) platform, some special components can be moved to single core, such as network communication, which will improve the performance greatly. Our work focuses on Cell, the typical AMP platform. The design of the network communication system based on Cell should consider the system architecture. Figure 2 illustrates the network communication high-level architecture based on the Cell system.

From Figure 2, it is noticed that we have modified three parts on the general platform. 1) The libc is replaced by the tinylib. As we know, the libc is offered by the Linux (the research platform is Linux), and it is located between user application and system call. In the conventional architecture, the user application should call the routines supported by libc to access the system call, and then entry the kernel space. In the new architecture, the libc is replaced by the tinyLib for the communication applications. The tinylib encapsulates some low level routines which are used to access SPU. And some routines used to allocate memory are encapsuled in the tinylib too. The tinylib can offer a serial interfaces to the application programmers. Through those interfaces, the application sends or writes data to or from the protocol stack transparently. 2) The tinylib can access protocol stack directly. In the conventional communication framework, the system calls must be called. In the new framework, tinylib can communicate with the protocol direct. And some function offered by the system call is moved to SPU. 3) The third change is the most important. The protocol stack and the device driver is moved into SPU. In the communication process, data encoding and decoding is the most time-consuming job. Improve the performance of component is the key to improve the communication performance.

In the new architecture, the PPU is used for controlling and synchronizing of multiple cores. The data communication task is assigned to one SPU. This architecture does not only reduce the workloads of PPU, but also improve the network data communication throughput.

III. NETWORK MODULE DESIGN FOR CELL PLATFORM

As described above, for Cell platform, although the computation capacity is sufficient for the SR applications, the network communication is not efficient to meet the huge throughput requirements. In the network module prototype, to improve the performance, the network operations are divided into two parts. One part includes data encoding and package sending functions and the other one is composed of package receiving and data decoding functions. Two SPE are used to handle the receiving and sending packets respectively. Moreover, some special considerations should be taken into account in the network module design.

Five key issues should be resolved in the new network communication model. 1) I/O communication; 2) Network interrupt management; 3) Network protocol stack analysis; 4) packages memory space management 5) and interface encapsulation. In the section, those technologies will be presented.

In Cell B.E. system, PPU, main memory, SPUs and peripheral devices communicate with each other through EIB. Peripheral devices should share the EIB (Element Interconnect Bus) bandwidth with other communication channels in the system, which will affect the I/O throughput badly. In order to improve the network throughput, SPU can access the device RAM directly by EIB and BEI (Cell B.E. Interface Unit) which manages data transfers between the processor elements on the EIB and I/O devices. In the paper, one of the main targets is to test the I/O throughput between SPU and I/O devices in the network model. Two kind of source are most important for device driver. One is I/O throughput and another is device interrupt. In the model, the interrupt is managed by the PPU but the interrupt server routine is located in the SPU. When the network interrupt is triggered, PPU will be activated first. When the PPU receives the interrupt, the PPU will notify the SPU that the network card has triggered interrupt through signal notification register[5], and then the SPU will responses the net card interrupt. The SPU will read and write data
packages based on the interrupt types. Most time in the communication process is spent in the protocol stack analysis. Optimizing the protocol stack program is one of most important problem in the model.

In the network communication, the data block size is about 1Kbytes, or smaller. SPE will parse packages of different size frequently. Worse still, many branches are used in the package parsing components. However, SPE is an in-order processor element, and SPE issues all instructions in program order. If there is dependency between two adjacent instructions, the later one has to wait to be issued until the former one completes. And this could lead to a huge performance loss. In addition, correctly predicted branches execute in one cycle, but a miss predicted branch incurs a penalty of approximately 18-19 cycles. Considering the typical SPE instruction latency of two to seven cycles, mispredicted branches can seriously degrade the packages parsing performance. So in the parsing components of the prototype, three methods are adopted to eliminate them to reduce the impact of branches, inlining, unrolling and prediction.

Inlining and unrolling both will increase the size of basic blocks. In our experiment, the total size of network protocol stack and the network driver used in some embedded system is less than 15Kbytes. And in the SPE, the local store size of SPE is about 256Kbytes, which is big enough to run the network card driver and protocol stack. SPE offers the select-bits instruction which can be used to eliminating branches for simple control-flow statements, such as if-then-else constructs. In the system, a lot of intrinsic instructions also are used to eliminate conditional branches, such as spu_cmpgt, spu_add, and spu_sel.

In the network model, the main job of SPU is to response the device and to analyze the protocol stack, and with the limit local store size, the data should be transferred into main memory. So an efficient memory management mechanism should be offered in the system. The SPU can get enough free memory to save the data packages received from the net, especially when the packages flooding happened. In the model, a simple memory allocation protocol is implemented between SPU thread and PPU process. In the PPU side, a net thread run as an daemon to offer kinds off service requested by SPU. When the new package arrives, the SPU will send request information to the daemon thread for memory allocation and the new memory address will be sent back to SPU. Then the package will be written into the memory. Same as the reading process, when the package has been written into the device RAM successfully, the SPU will notify the daemon thread to release the related memory in the main memory space. Figure 3 shows the detail architecture of receiver thread in the model.

Friendly interfaces are important for program users. In order to improve the portability of normal network programs, serial routines will be offered to consist with conventional network program. A library which encapsulates kinds of network function, such as socket creation, destruction, reading and writing, will be offered. The daemon thread coexists with the SPU and will be activated when the SPU is loaded.

IV. PERFORMANCE AND ANALYSIS

We implement the network communication module on the Cell multicore platform with the proposed architecture. In the implemented prototype, a simple communication mechanism is used for convenience.

For the communication protocol, the package head consists of data size, frame number, CRC check parity, and destination port as depicts in Figure 4. The protocol adopts the static package size, which is 512B length. It is obvious that the data size is 480Bytes, which means, if the data size is larger than the length, it will be partitioned by the protocol stack. And if the data size is smaller than 480B, the padding segment data will be added.

The prototype system is running on Cell blade QS21. In order to find out which kind of environment will affect the data transmission speed between the I/O device and SPU, the benchmark program will run with different environment configurations. The benchmark will also test the maximum throughput that the SPU can reach to read/write data from/to network I/O port.

<table>
<thead>
<tr>
<th>32bits</th>
<th>16bits</th>
<th>16bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC check parity</td>
<td>Data size</td>
<td>Frame number</td>
</tr>
<tr>
<td>Data Segment: 512B - 32B = 480B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Simple Protocol used in the Prototype

- Check parity: 32bits, the crc-32[6] value of the data segment;
- Data size: 32 bits, the size of the data segment, and it likes too long to specify the data size
- Frame number: 32bits, the serial number of the frame and it’s exclusive in the communication process.
- Port: 16bits, used to identify the user application.
Empty: 16bits, the padding bits for padding.

Data segment: 480Bytes, the valid data in the part.

At the same time, on the Cell platform, all the SPUs share one EIB bus. To simulate the practical environment, numbers of SPU is used to access the EIB bus in a time-sharing way. These SPUs will occupy the EIB bus by moving data between local store and main memory from time to time. Theoretically, when the EIB bus is occupied by other SPUs, the communication bandwidth between the network device and the SPU based network module will be degraded. In our experiment, we use different numbers of SPU to simulate the processing tasks (PT) in a practical system. The performance is provided in Figure 5. It is observed that the throughput of the communication module is just slightly degraded with the increase of processing SPU tasks. It is mainly because that the shared EIB bus on Cell platform has a very high bandwidth which can even reach up to 204.8GB/s.

![Figure 5](image1.png)

It can be also noticed that the throughput of the network communication module can even reach 14GB/s for the worst case, which is sufficient for SR applications.

On the implemented SR system, the PPU is used for tasks synchronization and management. The PPU resource usage is very important for an efficient system. In Figure 6 (a), the PPU usage status for the conventional communication framework on Cell platform, in which all the network communication operations are handled by PPU, is provided. In Figure 6 (b), the PPU usage for the framework with the proposed SPU network module is provided. Although when the system throughput increases, the PPU usage of the two cases will also increase. It can be observed that in the system with the proposed SPU based network module, the PPU usage will just slightly increase with the throughput. It is because that almost all the receiving, parsing and sending operations are handled by the SPU independently.

![Figure 6](image2.png)

**V. CONCLUSIONS**

This paper presents a novel SPU based network communication module for SR physical layer on the heterogeneous Cell multicore platform. A prototype of the communication module has been implemented. The performance of the network I/O communication framework is provided. The results show that the PPU resources are released and the SR system performance can be improved greatly with the new network communication module. From the performance result, it is clear that the novel network architecture can meet the I/O communication requirement, so the network module will be integrated with the SR high level modules in future.

**REFERENCES**


