AVR32705: AVR32AP7 Networking Performance

Features

- **Linux TCP and UDP performance measurements**
  - **ATSTK1000 (32-bit SDRAM bus width)**
  - **ATNGW100 (16-bit SDRAM bus width)**
- **Improved RX buffer management in the Linux MACB driver**

1 Introduction

This application note documents the TCP/IP and UDP/IP performance of the ATSTK®1000 and ATNGW100 development boards running the 2.6.23-rc7 version of the Linux® kernel. It also describes three optimizations of the MACB driver's receive path and documents how they impact the performance.Details on the optimization techniques applied to the MACB driver for the Linux kernel is described in this document. The result of this optimization work is available through Atmel®'s Linux Kernel through Atmel's Linux Support webpages.

Prior knowledge to Linux is not required to understand the optimization techniques, although preferred. Development Tools (ATSTK1000 and ATNGW100) for the AVR32AP7 micro-controllers are available from Atmel.

32-bit AVR® Microcontrollers

Application Note

Rev. 32066A-AVR32-02/08

2 Benchmarking tools

2.1 Iperf: The TCP/UDP Bandwidth Measurement Tool

Iperf is a TCP and UDP bandwidth measurement tool [1]. It can measure the maximum TCP bandwidth as well as UDP data loss at a given bandwidth, making it very suitable for measuring the impact of optimizations across the whole TCP/IP stack. In this application note, it is used to measure the performance of the low-level MACB Ethernet driver in particular.

The most basic use of iperf involves running one instance in server mode and one instance in client mode, sending packets from the client to the server. Thus, to measure how the target's TCP/IP stack performs on the receive (RX) side, first start iperf in server mode on the target:

```
iperf –s
```

Then, run iperf in client mode on the development host:

```
iperf –c 192.168.1.2
```

Replace "192.168.1.2" with the target's actual IP address.

By default, the iperf client will generate packets as fast as possible for 10 seconds and report the average bandwidth. For details about how to customize this, please see the iperf documentation.

To measure the target's transmit (TX) performance, simply do the above procedure the other way around: Start iperf in server mode on the development host, then start it in client mode on the target with the host's IP address as a parameter.

3 Linux MACB driver improvements

The transmit and receive paths of the MACB ethernet driver have stayed mostly the same since the initial version distributed with the STK1000 BSP 1.0. This section describes three optimizations that can be done to improve the performance of the receive path:

1. Store the first fragment of each packet into RAM with a two-byte offset.
2. Use non-coherent memory for the RX buffers.
3. Pass the RX buffers up the stack without copying them into a linear buffer first.

3.1 Linux socket buffers (skbuff)

The Linux kernel stores all information related to a network packet in a structure called "skbuff", or SKB for short [2]. The raw packet data can be stored in a linear data area in the SKB itself, but the SKB also contains a list of "fragments" where additional data can be passed. Each fragment is passed as a physical page reference along with offset and size information indicating which part of the page contains the data. The structure is declared in include/linux/skbuff.h and looks like this:

```
struct skb_frag_struct {
    struct page *page;
    __u32 page_offset;
    __u32 size;
};
```

As the Linux networking stack processes the incoming packets, data is copied from the fragments into the linear data area as needed. The code parsing the Ethernet header, however, does not do this, so a minimum of 14 bytes must be copied into the linear data area by the driver.

3.2 Optimization 1: Offset the first RX buffer by two bytes

An Ethernet header is 14 bytes long. This is not a multiple of 4, so if the packet data is word-aligned initially, whatever comes after the Ethernet header will not be, resulting in poor performance. The existing MACB driver solves this by adding two bytes of padding to the SKB data area, making the Ethernet payload properly word-aligned. However, since the DMA buffer does not have the same padding, the source and destination buffers will be aligned differently when the data is copied, resulting in poor performance of the copy operation itself.

The MACB controller includes a RBOFF field in the NCFG register which specifies the amount of padding to be added before a packet is stored in RAM. For packets that span multiple buffers, this padding is only added to the first buffer. By specifying 2 bytes of padding in this field and adjusting the RX copying code accordingly, a minor performance increase is expected due to faster memcpy performance.

Even though this optimization may not increase performance much by itself, it is a prerequisite for the last optimization where the buffers are passed up the stack as-is.

3.3 Optimization 2: Use non-coherent memory as RX buffers

The MACB driver currently uses coherent, i.e. non-cacheable, memory to store the RX data. This simplifies the code a bit because there's no need for any cache synchronization with such buffers, but it may also give suboptimal performance. By mapping the RX DMA buffers in non-coherent, cacheable memory, the CPU will be able to do burst accesses to transfer data from memory into the data cache.

The second patch replaces the coherent DMA buffers with regular pages obtained from the page allocator. Since each MACB receive buffer is 128 bytes and the default page size is 4096 bytes, each of these pages map 32 RX buffers.

Using single pages as receive buffers is also a prerequisite for the third optimization since the SKB fragment list consists of a list of single pages.

3.4 Optimization 3: Avoid copying of fragments into the linear data area

When one or more packets have been received, the current incarnation of the MACB driver scans the receive DMA descriptors and copies all the 128-byte DMA buffers that make up a packet into the linear data area of the SKB. The last optimization aims to avoid this copying by passing one or more references to the DMA buffers themselves up the stack.

Since Optimization 2 already replaced the single contiguous, coherent DMA buffer with multiple single pages, this becomes relatively straightforward. An SKB is allocated with just enough room for the Ethernet header plus the alignment padding. Then, each page that covers one or more of the receive buffers is added to the fragment list, after having its reference count incremented using `get_page()`. Finally, the Ethernet header is copied into the linear data area and the page offset and size of the first fragment is adjusted so that it doesn't cover the Ethernet header anymore.

When the SKB isn't needed anymore, `kfree_skb`() is called, freeing the linear data buffer and decrementing the reference count of all the pages in the fragment list using

put_page(). Eventually, when a page has been freed from all the fragment lists it was on and it is no longer used as a DMA buffer, the reference count drops to zero and the page is freed.

4 Results

The iperf measurement results are shown in Table 3-1. The measurements were obtained as follows.

- TCP RX bandwidth: “iperf –s” on the target and “iperf –c 192.168.1.231” on the host.
- TCP TX bandwidth: “iperf –s” on the host and “iperf –c 192.168.1.21” on the target.
- UDP RX datagram loss: “iperf –s –u” on the target and “iperf –c 192.168.1.231 –u –b 60000000” on the host.
- UDP TX datagram loss: “iperf –s –u” on the host and “iperf –c 192.168.1.21 –u –b 60000000” on the target.

Table 3-1. Iperf measurement results

Board	Kernel	TCP Bandwidth (Mbit/s)		UDP Datagram Loss @ 60 Mbps	
		RX	TX	RX	TX
ATSTK1000	2.6.23-rc7	54.6	69.1	14%	0%
	+ RX offset	54.5	71.3	17%	0%
	+ noncoherent RX	58.8	70.3	0.039%	0%
	+ avoid RX copy	66.1	65.5	0%	0%
ATNGW100	2.6.23-rc6	43.6	56.3	44%	0%
	+ RX offset	43.6	55.3	50%	0%
	+ noncoherent RX	47.6	55.7	46%	0%
	+ avoid RX copy	50.7	50.9	19%	0%

In the end, the three optimizations combined result in a 21% bandwidth increase on the STK1000 and a 16.3% bandwidth increase on the NGW100.

5 Further improvements

Some of the results were a bit surprising. Since the NGW100 has lower memory bandwidth than the STK1000 (16- vs 32-bit SDRAM databus), it is very strange that it apparently had less benefit from avoiding the copy operation. Also, the TCP TX bandwidth became lower on both the STK®1000 and the NGW100 when the “avoid RX copy” patch was applied.

Unless this is pure noise, it seems to indicate that the new driver handles certain corner cases less than optimally, and there might be more room for improvement in the driver.

6 References

Information about AT32AP7 is available on http://www.atmel.com/avr32/

1. Iperf website: http://dast.nlanr.net/Projects/Iperf/
2. How SKBs work: http://vger.kernel.org/~davem/skb.html

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-
Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr32@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature