SAFENET Internals

Bert Dempsey, John Fenton, Jeff Michel, Alex Waterman, Alfred Weaver
Computer Networks Laboratory
University of Virginia

Introduction

This section of the report describes the implementation of the XTP Unix driver and interface to the protocol processor for SAFENET.

Hardware Platform

The platform for this research project consists of a Dual Ring VME FDDI Board and a VME-based micro-processor installed in a C3 Desktop Tactical Computer (DTC). The DTC is based on the Sun Microsystems Sun4 architecture (see Figure 1). The DTC contains 32 Mbytes of on-board memory. This memory is disjoint from the memory

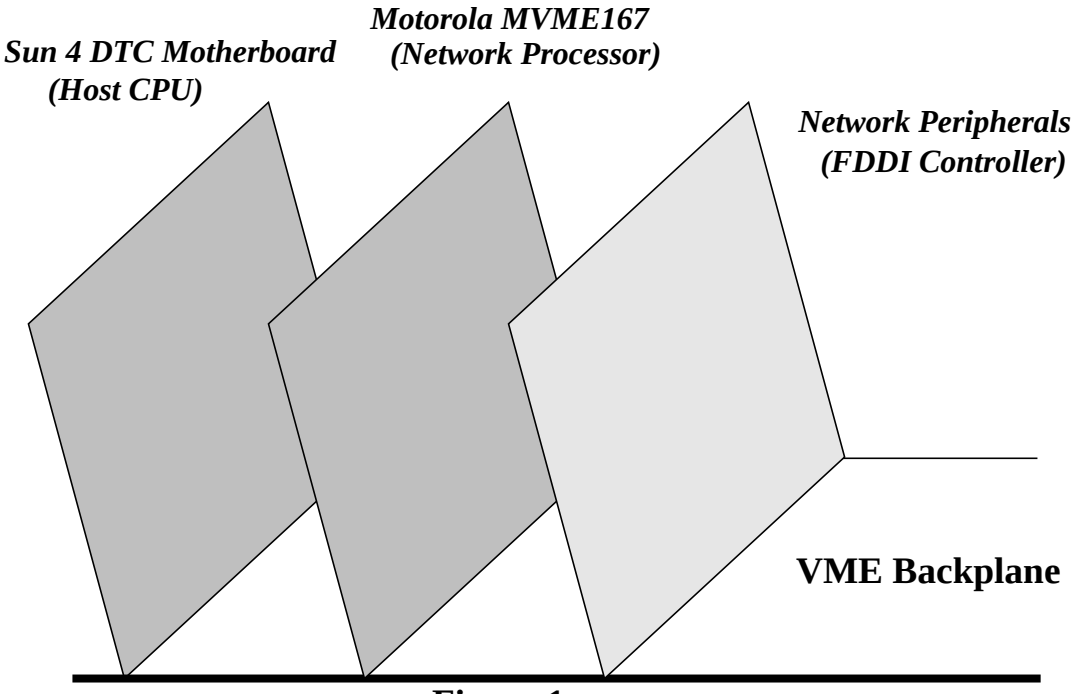


Figure 1

addressable by the VME backplane. The Sun 4 architecture divides VME memory into several subgroups (see Sun's `Writing Device Drivers/STREAMS Programming`). Of these subgroups the two most important are the lowest 1 Mbyte of memory, which is reserved for DMA transfers, and the top 16 Mbytes, which is available to map VME 16, 24 or 32-bit addressed devices.

The configuration uses the Network Peripherals (NP) dual ring VME FDDI board and the Motorola MVME 167 (167) board that each occupy one slot in the VME backplane of the DTC. The 167 is a 68040 based microprocessor operating at 25 MHz with 8 megabytes of onboard memory which is addressable on the VME bus. It is running Software Component Group's pSOS+ operating system version 1.2 and XTP 3.6.

The DTC is configured such that a portion of the 167 memory is mapped into the virtual memory of the OS kernel's 32 bit address area, facilitating communication through shared memory. Communication is also through interrupts generated on the 167 board for the DTC, which will be described later. The NP FDDI board has no processor of its own and is controlled by the 167. The communication between the NP and the 167 is also through both shared memory and interrupts.

DTC Operating System modifications

The standard configuration of the operating system included with the DTC is not capable of communicating with the attached Motorola processor. A number of configuration files must be changed, and a new operating system kernel built before communication can take place. We assume the reader is familiar with the kernel build process, and is capable of modifying the configuration (see Sun's `Writing Device Drivers/STREAMS Programming`). The following changes must be made:

1) Change directory into (/usr/sys/sun4/conf) and copy the kernel configuration file (DTC) to the file (XTP).

```
cd /usr/sys/sun4/conf
cp DTC XTP
```

2) Edit (XTP) with

```
vi XTP
```

Add the following line after the definition of the Ethernet controller.

device xtpmo0 at vme32d32 ? csr 0x1554F000 priority 2 vector xtpFintr 0xFF

3) Edit (XTP) with

```
vi XTP
```

and add the following line at the beginning of the file:

sundev/xtp_mo.c optional xtpmo device-driver

4) Edit (/usr/sys/sun/conf.c) with

```
vi /usr/sys/sun/conf.c
```

Before the definition of the cdevsw[] structure, add the following lines:

```
#include “xtpmo.h”
#if NXTPMO > 0
int xtpopen(), xtpclose(), xtpread(), xtpwrite(), xtpstategy();
#else
#define xtpopen nodev
#define xtpclose nodev
#define xtpread nodev
#define xtpwrite nodev
#define xtpstrategy nodev
#endif
```

and after the last entry of the cdevsw[] structure an entry must be added for this new device. Add the following:

```
{
xtpopen, xtpclose, xtpread, xtpwrite, /* <device number>*/
nodev, nodev, nodev, nodev, 0,
},
```

Where <device number> stands for the major device number which this device defines. The numbering is sequential so if the entry before this is N, the xtp device will be number N+1. The number is important later on when the device specific file must be added to the file system.

5) The following files must be copied from the included tape to the (/usr/sys/sundev) directory:

xtp_mo.c
xtpmo.h
xtperror.h
safenet.h
sla_shrt.h
moreg.h

6) Change directories into (/usr/sys/sun4/conf) and configure the new kernel:

```
cd /usr/sys/sun4/conf
config XTP
```

This will configure the new kernel you specified with the above changes to the existing kernel.

7) Change directories into (/usr/sys/sun4/XTP) and type

make

This directory is created as the result of the `config XTP`. The result will be a file named (vmunix), the new kernel.

8) A new entry must be made in the directory (/dev) for the XTP device. The major device number was defined above as <*device number*> and the node's name is xtpmo.

mknod xtpmo c <*device number*> 0

Hardware Configuration

VME bus access is obtained by requesting it from the system controller, the Sparc processor in our configuration. Both the DTC and the 167 have the ability to act as system

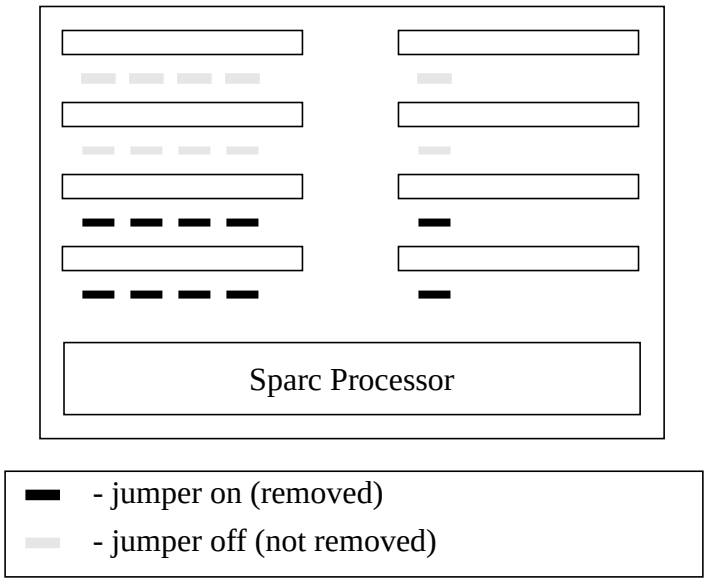


Figure 2

controller. The criteria for system controller is that the occupant of the lowest numbered slot of the VME bus must control all other's access, and the processor of the Sparc must reside in the lowest slot. Any peripheral on the bus that accesses VME memory must first request and receive access rights. Interrupts are used for communication between the NP and 167, and the 167 and DTC to signify data pending on the interrupter.

The choice of which slot to use for the different peripherals is arbitrary, but care must be taken to remove the row of jumpers that reside between the slot and the system controller (i.e. the DTC). The jumpers "jump" the different bus signals over unoccupied slots, so removing the jumpers allows the occupant of the slot to assert bus signals. The 167 must be configured not to act as system controller. The current configuration requires that the NP be configured to interrupt at VME level 3, and the 167 to interrupt at VME level 2. These are both configured in software, so no modifications are needed.

The 167 is configured to handle the level 3 VME bus interrupts, and the DTC handles all others, ignoring level 3. This allows for direct communication between the NP and 167 on bus level 3, and the 167 and DTC on bus level 2. A device driver is invoked by the DTC kernel upon receiving an interrupt on VME level 2.

A number of hardware configuration changes must also be made. As stated above, after choosing the VME slots to place the NP and 167 board in, remove the rows of jumpers situated under each slot (See Figure 2). This will allow the occupant of the slot to communicate along the VME bus.

The 167 must be reconfigured to not act as system controller. Refer to the reference manual for proper jumper reconfiguration. The EPROMs that are included with a standard 167 board do not have the XTP 3.6 software installed. We have provided a number of EPROM sets that include both pSOS+ and XTP 3.6. These EPROMs must be installed on the 167. The pSOS+ EPROMS will replace the set currently resident on the board in Bank A. The XTP 3.6 EPROMS must be placed in the vacant Bank B sockets. Refer to the MVME167 Single Board Computer User's Manual for chip replacement procedures.

The NP board must be modified to allow non-supervisor mode accesses. This simply says that the NP will respond to requests from a user, not just from privileged users. Refer to the hardware manual of the NP board for correct jumper configuration.

The DTC motherboard must be configured to ignore VME level 3 interrupts. The location of this jumper is found in the Sun Hardware Reference Manual included with the DTC.

With the installation of the modified NP and 167 boards and the new kernel, the hardware environment is complete.

Unix Device Driver Interface

The hardware and software that comprise the communication system are now installed. The communication that occurs between the components of our system will be discussed next, along with the interfaces associated with each.

The C-Library interface to the device driver is the standard Unix character device interface (See Figure 4). It utilizes system calls such as open(), read(), and write(). The open() is used to initiate a dialog between the user and the device. For bidirectional data transfers the user invokes the read() and write() system calls to transmit control blocks to the 167. For example, a connection is established by writing a control block to the 167 board requesting a connection. The board will respond with an acknowledgment if successful or a negative acknowledgment otherwise. This is an asynchronous request-acknowledgment protocol between the user and XTP. Requests will be delivered to the 167 board in control blocks with acknowledgments being returned in acknowledgment blocks.

When a user requests information, a control block is created and sent to the device driver and XTP processor. The control block identifies the connection with which the command is associated, and pointers to any required data. The device driver maintains enough information for each connection to notify the user when either a response arrives or an error condition arises.

The device driver maintains the status of all XTP connections and the allocation of connection identifiers along with shared memory data buffers associated with each. A four megabyte buffer for each host is allotted for incoming and outgoing connections. Memory is returned to the buffer pool on reception of a reply from XTP.

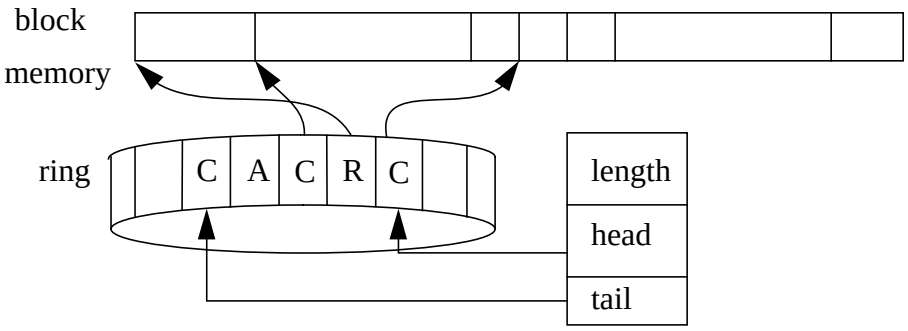
When a control block is queued for the 167 (described below), the device driver returns control to the user. When the device driver is informed of a reply (through a VME

bus interrupt), a user signal is sent to the originator of the request informing it of the pending reply. The process will respond with a read() system call, retrieving the pending response.

The user is informed of inbound information (i.e., data bound for some connection), through the use of the signal() library call. The user process will specify a service routine that will be invoked upon reception of a predefined signal using the signal() call. When information arrives for a certain process, this service routine will be invoked to handle the incoming data.

XTP Interface

The interface to the protocol processor consists of two communication ports. The first port is the Host Master Interface (HMI). The HMI is a command/response interface fixed in memory (on the MVME board) when pSOS+ and XTP are compiled. This interface is used to initialize the MVME board, the protocol state and the second communications port. It keeps the Motorola processor in lockstep with the SPARC processor until communication over the second port begins. The second communication port consists of two streams (see Figure 4), the “From Network stream” (FN stream) and the “To Network stream” (TN stream) . The FN and TN streams allow both processors to command and acknowledge each other asynchronously. Each stream consists of a fixed size ring structure and they share a fixed block of memory. The actual size of each structure is configurable through the HMI. The ring structure is used to pass commands, responses and acknowledgments. Commands and responses can have variable length data associated with them which are placed into the fixed block of memory (See Figure 3). This allows other commands to progress when a previous command is blocked indefinitely. All block memory management is handled by the DTC kernel.



C = command A = acknowledgment R= response

TN or FN stream interface

Figure 3

The TN and FN stream, HMI, and block memory reside in the memory of the 167. The location of the streams and HMI are also configured in the kernel of the DTC. At system start-up a handshaking protocol is used to assure the DTC that the 167 is present. The kernel may then map the block memory (located on the 167, addressable from the VME bus) into kernel virtual memory for later use.

System Data Flow

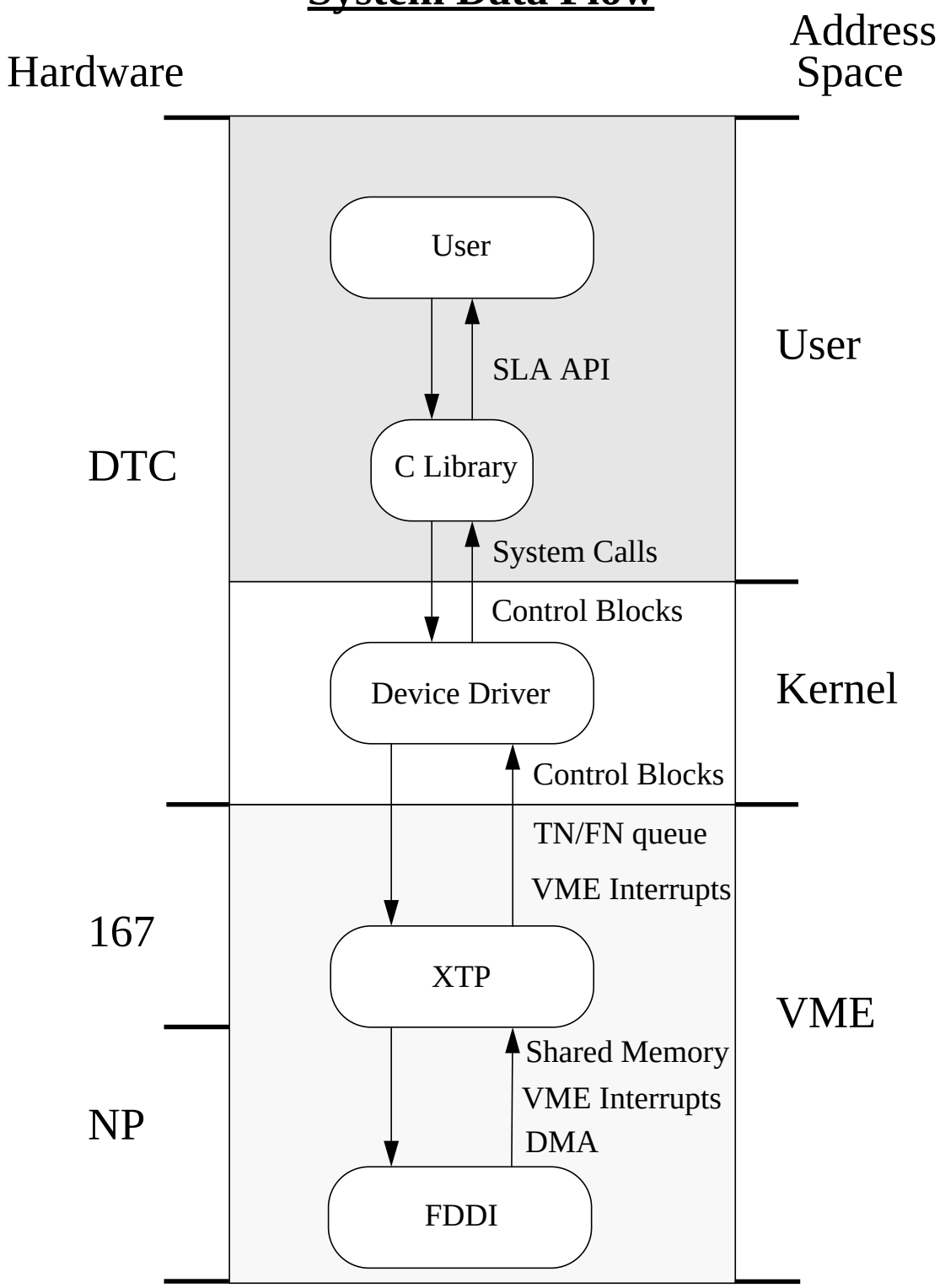


Figure 4

References

[1] *Writing Device Drivers*, Sun Microsystems, Inc., Mountain View, California, March 1990.