Vx960™

5.0.3 RELEASE NOTES

Intel Corporation

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

Order No. 517912

Release Notes: Vx960 5.0.3

November 1991

1. Introduction

This is the 5.0.3 Release of Vx960, Intel Corporation's offering of the VxWorks real-time operating system from Wind River Systems, Inc., specifically tailored for the Intel i960™ microprocessor family.

These release notes contain information not available in other documents.

2. Package List

You should have received the following items:

- Vx960 5.0.3 on a single tape
- ROMs for your target(s)
- GNU/960 1.3 on a single tape

You should also have received the following documents:

- *Vx960 Programmer's Guide*

NOTE: Although the 5.0.3 Release of the *Vx960 Reference Manual* has been significantly reorganized, the on-line version reflects the old organization.

- *Vx960 Reference Manual*
- *Release Notes: Vx960 5.0.3*
- Various board support packages in electronic form
- *GNU/960 Manual*
- *C: A Reference Manual,* Harbison and Steele
- Miscellaneous GNU/960 documentation
- *Intel, VxGDB 3.2 User's Guide.*

  This is the primary document for discussing how VxGDB works with Vx960 in a host environment. The document discusses the X11 interface. It also discusses configuration issues.
- The original *Free Software Foundation (FSF)* **gdb** 3.2 manual is included in part two of the *VxGDB User's Guide.*

If any of these items are missing, contact your local Intel field office.

3. GNU/960 R1.3

This release was built with Release 1.3 of the GNU/960 toolset from Intel. Development of code to run on top of Vx960 must be done with the same release.

GNU/960 R1.3 is included with Vx960 5.0.3. All applications previously developed for Vx960 must be recompiled with GNU/960 R1.3.

Install the GNU/960 R1.3 toolset before installing or using the Vx960 5.0.3 Release. Vx960 does not install correctly if your GNU/960 R1.3 tape has not been installed. Your G960BASE environment variable must point to the GNU/960 directory for your host. For complete details on installing GNU/960 R1.3 and setting your environment variables, see your GNU/960 R1.3 documentation.

NOTE: The next version of Vx960 will be built with Release 2.0 of the GNU/960 toolset from Intel.

4. Vx960 Installation

Vx960 is distributed on one QIC-24 format tape as a **tar** archive. To install Vx960, you need access to a streaming tape drive that can read a QIC-24 60MB tape. With the tape in the drive, move to the directory where you want the files installed. The following are typical invocations that read the tape:

```
tar xv
```

if your tape drive is the default source for **tar**, or

```
tar xvf /dev/rst0
```

on a BSD (Sun) system.

Another possible device name may be **/dev/tc**. Check with your system administrator for the invocation command line and path for your system.

Once your tape has been read in, run the shell script **vxInstall**. This requires that the G960BASE environment variable is set in the path to the GNU/960 R1.3 directory. To run **vxInstall** from the directory where your Vx960 5.0.3 is installed, run the following command:

```
sh ./vxInstall
```

vxInstall updates files which are specific to your host and creates the Makefiles required for cross development on your host. **vxInstall** prompts you for the host you are using. Currently, the hosts fully supported are Sun-3, Sun-4, i386 System V R3.2, HP 9000/300, and Apollo 400. If you have another host, read the script and **section 8** of these release notes to understand the installation requirements. For more general information on getting started, see your *Vx960 Programmer's Guide*.

5. Board Support Packages

The release has board support packages[1] for the following boards:

Heurikon HK80/V960E™

Cyclone CVME960™

Tadpole TP960™

Intel EV960SX Eval Board™

Intel Tomcat Eval Board™

Intel EV960CA Eval Board™ with the LAN960 daughter board

The Tomcat is included primarily as an example of an i960 board support package. It will be helpful for users porting Vx960 to their own board. The Tomcat is an Intel board designed for benchmarking. It is not commercially available.

6. Boot ROMs

You have received a boot ROM with this release that works with your target (four boot ROMs for the EV960SX board). This ROM boots the Vx960 executable image **vxWorks** for the board support packages (see list above) that came on the tape. For more information on the executable image see **Chapter 2, Getting Started,** in the *Vx960 Programmer's Guide.* If you already have Vx960 4.0.2 boot ROMs, these may still work for booting Vx960 5.0.3, but we recommend that you change the ROMs.

Refer to **Chapter 2, Getting Started,** in the *Vx960 Programmer's Guide* for more information on booting Vx960 and the boot ROM commands.

1. The EV960SX and EV960CA board support packages are not of a sufficiently high quality to be an Intel product but they are provided in this release in the interest of completeness. We plan to provide an upgraded and supported version of these board support packages in a future release.

The defaults for the boot ROMs provided are:

```
boot device:          ei
host name:            host
file name:            /usr/vw/config/<target>/vxWorks
inet on ethernet:     90.0.0.50
host inet:            90.0.0.3
user :                target
```

Neither a gateway nor a password is specifed.

NOTE: The 5.0.3 boot ROMs use a compression scheme to reduce the size of the ROM image. This adds an unnoticeable delay to the boot process. The target **bootrom_uncmp** exists in **/usr/vx/config/<target>/Makefile**. This rule builds uncompressed boot ROMs.

7. Passwords

Vx960 5.0.3 has a security flag that may be enabled (or disabled) in the file **/usr/vx/config/all/configAll.h** or alternatively in **/usr/vx/config/<target>/config.h**. If you are offered a login prompt when accessing a Vx960 target via rlogin or telnet and have not configured your system with a login ID and password of your own, the default login ID is the name **target** and the default password is **password**. Refer to section 9.8 of the *Vx960 Programmer's Guide* or **loginLib(1)** in the *Vx960 Reference Manual* for a discussion of remote login and security.

NOTE: Setting the **flags** field in the boot ROM parameters to 0x20 disables security.

8. Makefiles and Binaries

The relationship between Vx960 Makefiles and GNU/960 tool binaries is established through the use of the G960BASE environment variable. If the **make** program on your host does not automatically look in the environment to expand $(MACRO_NAME) macros ($(G960BASE) for example), you need to do one of the following:

- edit the Makefiles to explicitly define G960BASE or
- invoke **make** with an argument to force the environment to be examined for these missing macros.

On an Apollo 400 system, for example, you may have to use the **-P** flag on the **make** invocation to print out the complete set of macro descriptions. Refer to your UNIX man pages for more information.

All Vx960 5.0.3 libraries and target objects are compiled with the **gcc960** flag **-traditional**. In the future, we hope to remove this flag. This should have no negative performance impact.

The **-traditional** flag does not affect code generation, but is merely used to suppress certain type checking rules of ANSI compilation and for recognition of other forms for traditional C usage. These typing and usage issues will be resolved in a future release and developers are encouraged to write ANSI compatible code. (Some include files do not have prototypes correctly specified for ANSI compatibility.) We realize, however, that you may not be able to successfully compile some of the Vx960 include files without this flag.

Your application code should not use the **-mic-compat** flag. In the 4.0.2 release, the **-mic-compat** flag was required.

The **src/bin** directory contains the source for the tools which execute on your host. To rebuild the tools in this directory type "make".

The only reason to rebuild files in **src/bin** is to provide support for another host.

9. Networking Improvements

Testing has shown that network throughput has improved substantially between Vx960 4.0.2 and Vx960 5.0.3. Testing at Intel showed that TCP throughput approached 900K bytes per second.

There are a number of new shell status commands for Vx960 with the release that give networking statistics and information. These are:

- **ipstatShow** - IP protocol statistics (similar to: BSD **netstat -s**)
- **mbufShow** - network buffer statistics (BSD **netstat -m**)
- **tcpstatShow** - tcp protocol statistics (BSD **netstat -s**)
- **ifShow** - ethernet device statistics (BSD **ifconfig**)
- **icmpstatShow** (BSD **netstat -s**)
- **arptabShow** (BSD **arp**)
- **udpstatShow** (BSD **netstat -s**)
- **inetstatShow** (BSD **netstat -a**)

Problems with the subnet mask have been fixed.

Intel has enhanced the VxWorks **udpstatShow** command to give more information about **UDP** protocol activities on Vx960. In addition to previous totals, the command now shows the following:

- the total number of packets
- the number of input packets
- the number of output packets
- the number of UDP broadcast packets received with no Vx960 server listening for broadcast packets on the receiving UDP port.

9.1 SLIP

Intel supports SLIP and has tested it between a Sun 386i running Sun OS 4.0.0 and Vx960 5.0.3. A public domain Sun driver has been placed in **/usr/vx/unsupported/net/slip** with a README file.

To configure SLIP in Vx960, download **#define INCLUDE_SLIP** in **/usr/vx/config/all/config.h**. See configAll.h, usrConfig.c, or bootConfig.c for details.

---

10. VxGDB 3.2

The VxGDB 3.2 remote symbolic debugger is included in this release. It provides source-level debugging of Vx960 applications from a UNIX workstation.

Please note the following important points:

- Remote debug is already configured in Vx960.

  You do not have to install it in the **vxWorks** executable image. (Other than to perform initial installation of Vx960 itself). The **gdb** tools are found in the Vx960 bin directory; e.g., **/usr/vx/bin/sun4/** or **/usr/vx/bin/ap400**, etc. The remote server library **rdb.a** is already installed in the Vx960 libraries. **#define INCLUDE_RDB** is turned on in **/usr/vx/config/all/configAll.h**.

- The command-line version of **gdb** is supported on all hosts.

- There is a new X interface for the following hosts:
  - Sun-4
  - Sun-3
  - HP9000/300
  - IBM RS/6000
  - dec3100
  - vax-ultrix
  - Sun 386i

- Source code for **gdb** is not supplied on this tape. If you want source code, request it from your local Intel sales office, or send a message to **vx960bugs@ichips.intel.com**. Intel will provide any source code derived from FSF.

There are two **gdb** front-ends. There is a command-line version called **vxgdb960** that is based on the original **gdb** 3.2 version. There is also a new X11R4 based version, called **xvxgdb** which provides visual debugging via an X client. The interface is similar to Sun's **dbxtool**. See the *VxGDB 3.2 - User's Guide* for more information on **xvxgdb**.

10.1 Graphical User Interface

VxGDB 3.2 extends the line-oriented mode of the GNU Source-Level Debugger (GDB), Version 3.2, with a graphical user-interface based on the X Window System. The graphical user-interface lets you enter commands either by selecting a command button with the mouse or typing the command directly into a dialogue window. The *User's Guide to VxGDB* describes how to use the new VxGDB interface.

10.2 Corrected Problems

This section describes errors found in the version of VxGDB that accompanied Vx960 4.0.2 which have been corrected in VxGDB 3.2. For each item, a brief description of the corrected error is presented.

1. **VxGDB could not access COMM symbols.**

COMM symbols are global variables defined without an initializer, e.g.:

```
int IamCOMM;              /* this is a COMM symbol */
int IamNotCOMM = 0;       /* this is not */

func ()
    {
    ...
    }
```

Memory for these symbols is not allocated until link time, or, in the case of Vx960, until load time. VxGDB did not know the address of COMM symbols allocated by the Vx960 loader. If you attempted to examine or modify a COMM symbol, VxGDB would read data from address 0 and report an incorrect value. If you attempted to otherwise use a COMM symbol in an expression (e.g., dereference a COMM pointer variable), *ptrace* errors could occur.

2. The VxGDB stack trace stopped before the outermost frame was reached.

Under certain circumstances, you might observe problems involving the VxGDB stack-tracing mechanism. They occurred most commonly when the target program consisted of a number of linked object modules. Problems with stack tracing were also affected by the relative locations of Vx960 and the target program in memory, and possibly the relative locations of individual object modules.

When the problems occurred, the **backtrace** (a.k.a. **where**) command would not show a complete stack trace. Other VxGDB commands that move up and down the frame chain were also affected, including: **up**, **down**, **frame**, and **info frame**.

3. The VxGDB stack-variable display had problems leaving the outermost function.

Displays involving stack variables were not handled properly when the outermost function went out of scope. If the **display** command was used to display the value of an expression involving stack variables local to the task entry point, a *ptrace* read error could occur when the program was stepped through the end of that function.

4. **VxGDB printed an error message when displaying an invalid address.**

   An error message was printed after attempting to display an invalid address. The error message took the following form:

   ```
   -> remote_rdbsvc: ptrace error 14 in request 1
   t863992: remote server; status=0xe
   ```

   In most circumstances, VxGDB continued to operate normally after the error message appeared.

5. **VxGDB did not stop the program if jumping to a breakpoint.**

   If you set a breakpoint at a line in a program, then used the **jump** command to continue execution beginning with that line, the program did not stop at the breakpoint.

6. **VxGDB did not attach to the target if Vx960 booted with an incomplete pathname.**

   When booting Vx960, an incomplete pathname for the boot file is interpreted as a relative pathname from the specified user's home directory. However, VxGDB did not resolve an incomplete pathname, and did not find the Vx960 object file when attaching to the target.

7. **VxGDB did not connect to the target system if the object module was not found.**

   The **target** command would fail if VxGDB was unable to find an object module previously loaded into the target system. The **target** command has been modified to print a warning message when this occurs, then continue with normal operation.

8. **The target command could not be cancelled.**

   It was not possible to cancel the **target** command by typing the interrupt character (^C).

9. Exceptions occurring in the target task were not reported.

If an exception (such as Bus Error) occurred in the task being debugged, VxGDB would not inform you that task execution had stopped.

10. Pressing the RETURN key caused the target command to repeat.

Hitting the RETURN key after giving the **target** command caused the command to repeat. This behavior is appropriate for some debugger commands, such as **step** and **next**, but not for **target**.

11. Function arguments of type "float" were displayed incorrectly.

When the target program stopped in a function with one or more arguments of type *float*, the values of such arguments were reported incorrectly by VxGDB. Incorrect values would also be printed in a stack trace display through the function in question.

12. Problems with the interactive function calling mechanism.

The implementation of interactive function calls (functions called from the VxGDB command line as part of C expressions) has been revamped to remedy a number of problems with the earlier implementation. These problems included:

- Only arguments of type *int* were supported.
- Non-integer return values were not supported.
- If the called function failed to return for any reason, the remote debug server would die.

13. The name of the command history file was truncated.

The name of the file in which VxGDB saved the command history (e.g., as specified by the **set history file** command) was being truncated by a single character.

14. RPC failure while the target task was running caused VxGDB to crash.

An RPC failure occurring while the debugged task was running would cause VxGDB to crash.

15. The run command was confused by white space within literal strings

White-space characters within literal string arguments to the **run** command were misinterpreted as argument delimiters. This would usually cause the **run** command to fail.

16. Attempting to attach to a non-existent task produced an inaccurate error message.

The message:

```
ptrace: Not owner
```

was printed when you attempted to attach to a non-existent task on the target system. VxGDB now prints the correct message,

```
ptrace: No such process
```

10.3 Known Problems in VxGDB 3.2

1. External symbol names ending in "_text", "_data", and "_bss" generate spurious warning messages.

Spurious warning messages complaining of files not found may be printed when you enter the **target** command to connect to the Vx960 target. The problem arises only if a program loaded into the target contains external symbols whose names end with: (1) *_text* in the text section, (2) *_data* in the data section, or (3) *_bss* in the *bss* section.

Workaround: Rename the external symbols in question.

2. Incorrect single-stepping near like-named static functions

Source-level single-stepping may not function properly in the vicinity of a static function when other static functions with the same name exist. Erroneous behavior can include:

- The inability to step over a call to a static function whose name is not unique.
- Landing at the opening brace of the function instead of the first line of actual code when stepping into a static function whose name is not unique.

The problem only arises when the object modules containing the like-named static functions are incrementally loaded into the Vx960 target using the VxGDB **load** command or the Vx960 *ld*( ) routine.

3 The "-d" argument now requires a special form.

Extra consideration must be observed when using the **-d** flag to establish initial default directories. Note that a space must separate **-d** and its argument. If multiple directories are desired, they must occur in a colon-separated list (not as repeated **-d** flags). For example:

Correct:

```
vxgdb -d /folk/tools/test
vxgdb -d /folk/tools/test:/folk/tools/production:source/C
```

Incorrect:

```
vxgdb -d/folk/tools/test   no space between -d and /
vxgdb -d /foo -d /bar      must use -d /foo:/bar instead
```

4. VxGDB does not comply with Open Look cut and paste conventions.

If you are using Open Look, you will notice that **Paste** does not work as expected with VxGDB. Instead, VxGDB uses the more general mouse button cut and paste familiar to users of **xterm** (as opposed to **shelltool** or **cmdtool**). In this environment, a cut (actually a copy, as the source text is undisturbed) is made by holding down the left mouse button and dragging the cursor over the desired text. To paste, click the middle mouse button. This method will work both within the VxGDB window and with other X applications (such as xterm) that use this cut and paste convention.

5. **Source window can redraw itself unnecessarily.**

   Clicking the left mouse button near the bottom of the source window can cause the source window to redraw itself unnecessarily.

6. **Temporary breakpoints disappear.**

   With the X front end, temporary breakpoints disappear after setting them and doing an initial run. They only disappear in the X interface but not in **gdb** proper; i.e., they are still there, but you can't see them in the X front end. They work correctly from the command-line interface.

7. **Quit in X window, task still suspended on vxWorks.**

   If you quit from the X window (the menu-bar at the top) as opposed to doing it explicitly from the **xvxgdb** interface, the task on **vxWorks** under debug is not terminated. Note that this is not the case if you quit from either the **xvxgdb** quit button or via the command line. In the latter case, the remote task is killed.

10.4 Helpful Hints on Using VxGDB 3.2

10.4.1 Loading Files with VxGDB 3.2

When you load a file remotely with **gdb**, you must change the directory so that Vx960 has access to the file. For example, assume that **myfile.o** is stored in the directory **/usr/vx/jrb/tests**. To load this file on Vx960:

```
-> cd "/usr/vx/jrb/tests"
```

To load this file on the host:

```
(vxgdb) load myfile.o
```

Alternatively, if you use absolute path names to do a load, you do not have to change the directory on Vx960; for example,

```
(vxgdb) load /usr/vx/jrb/tests/myfile.o
```

NOTE: An absolute pathname is a pathname that starts from root; not the current directory of your shell.

You can also change the directory on Vx960 remotely from **gdb**; for example,

```
(vxgdb) run cd "/usr/vx/jrb/tests"

(vxgdb) load myfile.o
```

also loads **myfile.o**.

10.4.2 Running Vx960 Functions from GDB 3.2

You can run any function in Vx960 from **gdb**; for example, if you wanted to run **memShow** or **i** on Vx960 after running the **target** command:

```
(vxgdb) run memShow

(vxgdb) run i
```

the output does not go to **gdb**, but goes to the Vx960 console.

You can also redirect the output of a function. For example, if you are using NFS, and permissions are setup correctly, you can do the following:

```
(vxgdb) run i > i.out
```

The file **i.out** will be placed in the current working directory for Vx960. To setup NFS permissions, you may have to use the ***nfsAuthUnixSet()*** command on Vx960 command: e.g.,

```
-> nfsAuthUnixSet("myhost", 2001, 100)
```

to set permissions, and then use

```
-> nfsAuthUnixShow
```

to view the permissions to make sure they are correct.

10.4.3 Size of Executables

gdb runs on the host and under some circumstances may not be able to load large executables. This has nothing to do with Vx960 and everything to do with **gdb** and

the host operating system configuration. If **gdb** has problems downloading a large executable, you can:

- Use the csh **limit** command to see what the hard and soft resource limits are for your system (assuming a BSD-style system). You may have to raise the soft limits for data and stack sizes. You may even have to get the system reconfigured to allow bigger data or stack segments. **gdb** uses a lot of data space for symbol table storage.
- Break up the large file into smaller sections and download only the sections you need to debug with debug symbols (**-g**). You can use the Vx960 incremental loader to advantage here.

10.4.4 Supplied Tools/Invocation of VxGDB 3.2

There are three tools supplied in the bin directory:

- **vxgdb** - is a shellscript. On hosts supporting both the command line and X interfaces, it will choose which **gdb** front end to start.

  vxgdb -l : starts the command-line version (**vxgdb960**)
  vxgdb : starts the X version (**xvxgdb**)

See the *VxGDB User's Guide* for more information.

- **vxgdb960** - is the command-line version of VxGDB 3.2. It may be invoked directly.
- **xvxgdb** - is the X11R4 version. This is best invoked through the **vxgdb** shell script.

You should put the relevant **bin** in your path to access the **gdb** tools; e.g., with csh:

```
% setenv PATH $PATH:/usr/vw/bin/sun4
```

10.5 New INCLUDE Options

The INCLUDE options INCLUDE_GNU_CLIB and INCLUDE_GNU_FLOATLIB can be defined in **/usr/vx/configAll.h** or **/usr/vx/<target>/config.h** to force all of the GNU

libraries to be loaded into the kernel. When an application is downloaded, the loader resolves the library references.

To obtain the standard C library functionality or the floating point library functionality, you can do one of the following:

- If you want the standard C library as part of your application, you can specify the **-lcg** option on the command line or you can specify INCLUDE_GNU_CLIB as a preprocessor option to the kernel configuration.
- If you want the floating point library as part of your application, you can specify the -lfpg option on the command line or you can specify INCLUDE_GNU_FLOATLIB as a preprocessor option to the kernel configuration.

The following table illustrates the choices described above.

	Command-line Option	Preprocessor Option
Standard C Library	-lcg	INCLUDE_GNU_CLIB
Floating Point Library	-lfpg	INCLUDE_GNU_FLOATLIB

When you define INCLUDE_GNU_CLIB or INCLUDE_GNU_FLOATLIB, do not specify the options **-lcg** or **-lfpg** respectively on the **gld960 -r** command line. If you specify these INCLUDE options, the application runs, but memory is wasted since some routines will be loaded twice -- the first time, since they already exist in the kernel and the second time, when the application is downloaded.

The following table outlines what will occur if you do or do not specify the INCLUDE options.

	Preprocessor Option Specified	Preprocessor Not-Specified
Command-line Option Specified	The application runs but memory is wasted	The application runs.
Command-line Option Not Specified	The application runs.	The application downloads with unresolved symbols.

For additional information on other INCLUDE options see **Table 8-1. Selectable Vx960 Options** in the *Vx960 Programmer's Guide.*

11. CPU Dependencies

Vx960 includes support for the following processors: i960CA™, i960SA™, i960SB™, i960KA™ and i960KB™. The CPU macro name is used to control include file resolution. Applications should be built with either the **-DCPU=I960XX gcc960** command-line option, or

```
#define CPU I960XX
```

could be defined in the application.

You must use the **CPU** macro name with the **-AXX gcc960** option. You can substitute the following for XX:

- **CA** for the i960CA microprocessor
- **KA** for the i960KA microprocessor
- **KB** for the i960KB microprocessor

New libraries are included in the **lib/<processor>** directories to support all the processors. Since the KA/KB microprocessors are software compatible with the SA/SB microprocessors, only the KA and KB libraries are provided.

12. Known Problems in Vx960 5.0.3

The following list represents problems that we have noted in our testing but are not yet fixed. These are in addition to the other caveats listed in the next section.

1. **Full system won't fit on 1 MB system.**

   It is not possible to boot a fully configured system with symbol table into 1M of memory.

2. **lsOld on ftp network device**

   The command lsOld should work with a ftp network device as created with *netDevCreate*( ) but it does not. lsOld does work with a rsh network device. The ls command is reserved for NFS or local file systems.

3. **The *Vx960 Programmer's Guide* claims that you can exclude the network facilities by undefining the options INCLUDE_NETWORK, INCLUDE_NET_INIT, and INCLUDE_NET_SYM_TBL.**

   The code will not be executed but will still reside in the image.

4. **errno and socket *ioctl* calls**

   socket *ioctl* calls (for example, adding a route table entry or adding arp table entries) have the *ioctl*( ) call return the UNIX errno value directly. errno is not set via *errnoSet*( ). Other *ioctl*( ) calls in non-network code set errno via *errnoSet*( ); e.g., console driver calls. Vx960 is not consistent about when *ioctl*( ) returns errno directly or when *errnoGet*( ) must be called.

5. **The demo/dg programs do not build.**

13. Caveats

The following items might cause you problems.

1. **In section 2.6.5.1 of the *Vx960 Programmer's Guide*, you will find the following example command line:**

```
$ ei(0,0)mars:/usr/vx/config/hkv960/vxWorks e=90.0.0.50
          h=90.0.0.1 u=fred
```

In fact, it should read:

```
[vx960] $ ei(0,0)mars:/usr/vx/config/hkv960/vxWorks
          e=90.0.0.50 h=90.0.0.1 u=fred
```

The space before and after the dollar sign ($), and the dollar sign are necessary.

2. **Heurikon HKV960E console moved to port 0.**

   The console device on the HKV960E has been changed from port 1 to port 0 in order to match the Vx960 documentation. The console now functions on device /tyCo/0.

3. **Strings used in the shell have space allocated that is never freed.**

   To demonstrate a sample problem, type:

```
-> memShow
-> cd "<dir>"
-> memShow
```

4. **Some boards do not generate any software faults or interrupts when invalid memory locations are accessed.**

   Since the **vxMemProbe** routine and the Vx960 system require some type of software notification, invalid accesses are not guaranteed to generate accurate results. For example, if you tried to write to ROM or tried to read from non-existent memory, the operation would not succeed, but you would not get an error message.

5. **Be careful that #include "vxWorks.h" comes first in your source modules.**

   If it doesn't, you may have some problems that will *not* be obvious.

6. **Avoid using ^C from the Shell.**

   When you execute any Vx960 routine from the shell, the routine executes under the shell process. When you type ^C to abort the process, you kill the shell pro-

cess. Any semaphores taken by the shell process remain blocked along with the resources they are protecting. For example, a semaphore remains blocked after you type ^C to abort a file copy operation. Additional file operations are blocked. At the application level, you can use ***taskSafe( )*** and ***taskUnSafe( )*** to protect the calling task from deletion and thus protect any resources used by the tasks. However, since the shell is used mainly as a software development and debugging aid, no such protection is offered for the shell.

7. *diskFormat( )* requires additional file system initialization.

The DOS file system as implemented places the entire File Attribute Table (FAT) in memory. If a floppy drive or winchester drive is formatted, ***diskFormat( )*** does not inform the file system that you want to initialize the device and so previously-existing files may still appear. The ***diskInit*(dev)** command must be issued to tell the file system to create a new file system for the device. Alternatively, you can issue a ***dosFsVolUnmount( )*** or ***dosFsReadyChange( )*** prior to the ***diskFormat( )*** to inform the file system that it must read the disk the next time it tries to access it for the directory structure.

8. When using VxGDB, you must use the Vx960 shell to change to the source/object directory for a VxGDB load-command to succeed.

9. When using VxGDB, you cannot step into library functions like *strcpy( )*, or *strcmp( )*; you must next over them.

10. Ethernet address setup on Intel Tomcat, EV960SX, and EV960CA boards

The Intel Tomcat, EV960SX, and EV960CA boards do not have a mechanism for storing an ethernet address on board in hardware. Both the boot ROM and the Vx960 kernel must have hardwired, ethernet addresses in software. Those addresses must agree. To make this situation more apparent, the ethernet address has been moved out of **/usr/vx/config/<target>/sysLib.c** into a file called **enet.h**; e.g., **/usr/vx/config/evsx/enet.h**. The boards themselves have assigned ethernet addresses on a label. For each system you want to use, you should do a separate build of both the boot ROM and the Vx960 kernel.

You could establish a separate configuration directory and copy the hardware address on the label into the **enet.h** file. Note that the current ethernet hardware prefix for Intel is: 0:aa:0:3. (The ethernet address is six bytes long; e.g., 1:2:3:4:5:6).

The ethernet address on the SX board is located on the DB25. For the EV960CA board, the label is located on the 596 daughter board.

If there is no label with an ethernet address, you will have to create a unique address. You should use the BSD command **arp -a** to make sure that you have chosen a unique address and you should use the Intel prefix, possibly choosing a much higher fourth byte to avoid overlap with current Intel products (e.g., replace the "3" above with ff).

A sample file is shown here:

```
 /* enet.h - Ethernet address */

/*

Description
This file exists for the sole purpose of hardwiring an ethernet
address for boards like the evsx that have no means (e.g., no NVRAM)
for storing a per board ethernet address.

Note:0x00 0xaa 0x00 as the first 3 byte prefix means INTEL.

Instruction
The ethernet address is on the board on a sticker.

            6 bytes...
            Get it, and set it in here.

    Then rebuild both boot ROM and Vx960. You must do this PER board;
    i.e., each board should be unique.

*/

unsigned char sysEnetAddr [6] = {0x00, 0xaa, 0x00, 0xff, 0x00, 0x01};
```

14. Reporting Problems

Refer to **2.8 Troubleshooting** in the *Vx960 Programmer's Guide*. The electronic mail address for bug reports is:

```
vx960bug@ichips.intel.com
```

15. Example of SCSI/DOS Setup

The following example shows how to add a floppy drive to a Vx960 system. This example system consists of a Heurikon board with a SCSI connection to a Teac FD-55GS 751-U floppy drive. Vx960 is configured to use the SCSI device driver and the DOS file system.

1. Make the following code changes and rebuild Vx960.

   In config.h:

   ```
   #define INCLUDE_SCSI
   #define AUTO_SCSI_CONFIG
   ```

   In usrConfig.c, turn on the following options:

   ```
   #if TRUE/* by default, these are NOT turned on */
       scsiDebug = TRUE;       /* enable SCSI debugging output */
       scsiIntsDebug = TRUE;  /* enable SCSI interrupt debugging */
                              /* output */
   #endif /* FALSE */
   ```

2. Setup the Teac FD55GS 751-U floppy drive.

   Jumper for:

   bus ID = 3 on the SCSI board of the drive.
   logical unit = 1 on the FDD board. Other jumpers are left as defaults.

   Connect the Teac to the SCSI port on the Heurikon chassis with a SCSI cable. Connect power to the Teac drive.

NOTE: It is possible to jumper the floppy driver for different bus IDs and logical units and then change the code in ***usrScsiConfig( )*** in usrConfig.c.

3. Run Vx960 and review the output to the shell to confirm that the two SCSI controllers are communicating.

   Run the following from the shell:

```
-> usrScsiConfig
-> scsiShow
```

   NOTE: It is necessary to insert a floppy disk in the drive, since configuration data is read from the boot sector of the disk.

4. Make the following code changes and rebuild Vx960.

   In config.h, remove:

```
#define AUTO_SCSI_CONFIG
```

   In usrConfig.c, change the code to the following:

```
#if FALSE                   /* by default, these are NOT turned on */
scsiDebug = TRUE;           /* enable SCSI debugging output */
scsiIntsDebug = TRUE;       /* enable SCSI interrupt debugging */
                            /* output */

#endif                      /* FALSE */

/* If you are not using a Winchester device, then put the */
/* following #ifdef around the Winchester configuration code */
/* in usrConfig.c.
*/
#ifdef SCSI_WINCHESTER
/* configure Winchester at busId = 2, LUN = 0 */
...
```

5. Run Vx960 to confirm that the drive is configured properly.

   Run the following from the shell:

```
-> usrScsiConfig
-> scsiShow
-> iosDevShow
-> ls "/fd0/"
```

If these commands have acceptable results, your device is working and you can proceed. Be sure to read the section on ***dosFsLib*** in the *Vx960 Reference Manual.*

6. Configure the DOS File System

   Note that ***dosFsDevInit( )*** is called by ***userScsiConfig( )***. ***dosFsDevInit( )*** associates a block device with the dosFs file system functions. If the third parameter (*pConfig) is NULL, then configuration data will be read from the boot sector of the disk device. If the device is a floppy drive, then a floppy disk must be in the drive for this to successfully complete. Alternatively, the configuration data can be passed to ***dosFsDevInit( )***. Passing the configuration data is useful if you want to change some of the drive characteristics, e.g., auto-sync mode or change-no-warning.

   There are two commands that are undocumented in the *Vx960 Programmer's Guide*, ***dosFsConfigShow( )*** and ***dosFsConfigGet( )***, which allow you to obtain the current configuration characteristics of a disk drive.

   dosFsConfigShow (char * deviceName)

   To use this function from the shell:

```
-> dosFsConfigShow "/fd0/"
device name:               /fd0/
total number of sectors:   2400
bytes per sector:          512
media byte:                0xf9
# of sectors per cluster: 1
# of reserved sectors:     1
# of FAT tables:           2
# of sectors per FAT:      7
max # of root dir entries:224
# of hidden sectors:       0
removable medium:          TRUE
disk change w/out warning:FALSE
auto-sync mode:            FALSE
volume mode:               UPDATE (read/write)
value = 0 = 0x0
```

   The values shown above can be saved so that they can be passed as the volume configuration parameters to ***dosFsConfigInit( )***.

   dosFsConfigGet ((DOS_VOL_DESC *vdptr, DOS_VOL_CONFIG *pConfig)

To use this function from the shell:

```
-> pConf=malloc(100)
new symbol "pConf" added to symbol table.
pConf = 0x7fdbd0: value = 8379376 = 0x7fdbf0 = pConf + 0x20
-> dosFsConfigGet (pSbdFloppy, pConf)
value = 0 = 0x0
```

The values in pConf above can be used as the third parameter in a call to *dosFsDevInit( )* to initialize a device with unformatted media.

16. New Format for the *d( )* and *m( )* Commands

NOTE: The following description of the *d( )* and *m( )* commands represents a change from the Vx960 4.0.2 release and the Vx960 5.0.3 beta release.

The *d( )* command for dumping memory, and the *m( )* command for modifying memory have been changed from 16-bit (short) to 32-bit (long word) formats to match the native byte order of an Intel i960 CPU.

There is a known problem with the *d( )* command in this release. The command is invoked as *d*(**adrs, nwords**) where:

- **adrs** is the address to start displaying from.
- **nwords** is the number of words to display.

The actual number of words displayed is **2*nwords**.

17. arpLib

Intel has ported the BSD ***arp*(8)** utility to Vx960 to permit manipulation of the network arp table. Two routines are added to Vx960 if **#define INCLUDE_ARPLIB** is defined in **/usr/vx/config/configAll.h**. (These routines are not included in the binary by default). The two routines are ***arpAdd*( )** for setting arp entries and ***arpDelete*( )** for removing arp entries from the arp table. Note that arp entries can be made permanent or published (proxy arp) via the ***arpAdd*( )** call. Please see the **arpLib.nr** man page in the next section for more information.

NAME

arpLib - arptable subroutine library

SYNOPSIS

arpAdd( ) - add an individual arp table entry
arpDelete( ) - delete an arp table entry
STATUS arpAdd(host, eaddr, flags)
STATUS arpDelete(host)

DESCRIPTION

This library provides the following routines: ***arpAdd( )*** for setting and ***arpDelete( )*** for deleting entries in the network arp table. The arp table contains entries where each entry consists of an Internet address (IP address) and a paired ethernet hardware address. The table is used at the link level to map local network IP addresses into hardware level ethernet addresses for transmission of packets to local network hosts.

Table entries are put in automatically via broadcast arp messages normally, when a packet must be sent to a local host, and there is not already an arp cache entry for that host. Normally these cache entries timeout after a number of minutes. However, you might choose to program in arp entries to make them permanent, published, or set to use Berkeley trailers. This is done with the flags variable in the ***arpAdd( )*** call. A permanent entry does not timeout. A published entry can be used for doing "proxy" arp; i.e., having one host give arp replies for another host. See the ***arpAdd( )*** manual entry below for more information.

SEE ALSO

inetLib, routeLib, etherLib, netShow

NAME

***arpAdd*()** - add an individual arp table entry

SYNOPSIS

```
STATUS arpAdd(host, eaddr, flags)
    char    *host;    /* host name or IP number as string */
    char    *eaddr;   /* ethernet number as colon separated string */
    int     flags;    /* arp settable flags */
```

DESCRIPTION

This routine adds an arp table entry to the network arp tables. The parameters must be a valid Internet address and a valid ethernet address plus any special flags. The ethernet address should be represented as bytes separated by colons.

EXAMPLE

The call:

```
arpAdd("90.0.0.3", "0:80:f9:1:2:3", 0x4)
arpAdd("90.0.0.4", "0:80:f9:1:2:4", 0)
```

will add an arp entry for the machine with IP address 90.0.0.3, ethernet address 0:80:f9:1:2:3, of a permanent type; i.e., the entry will not time out. The second call adds a normal entry for an IP address of 90.0.0.4. This entry will eventually timeout. Entries, unless made permanent, do timeout eventually and are removed from the table.

Settable flags are as follows:
(see if_arp.h)
no flags 0
ATF_PERM 0x04
ATF_PUBL 0x08
ATF_USETRAILERS 0x10

RETURNS

OK, ERROR, errno if not ERROR.

SEE ALSO
arpLib

NAME

arpDelete( ) - delete an arp table entry

SYNOPSIS

```
STATUS arpDelete(host)
    char *host;
```

DESCRIPTION
This routine is used for deleting an arp table entry from the arp table. The host IP address must be specified.

EXAMPLE
The call:
arpDelete("91.0.0.3")
would delete the arp entry for IP address 91.0.0.3 from the arp table.

RETURNS
OK or ERROR.

SEE ALSO
arpLib

18. Routines Removed from the Board Support Packages

In the current version of Vx960, many routines have been moved from the board support package to the kernel. The previous prefix to these routines was "sys"; the new prefix is "vx". For example, the routine ***sysACWGet( )*** has been renamed to ***vxACWGet( )***. Unfortunately, this information was not included in the *Vx960 Reference Manual*. The previous routines can still be used if they have been incorporated into the existing BSP or included in an application.

The following man pages describe the affected routines.

NAME

vxALib - Intel i80960 miscellaneous assembly routines

SYNOPSIS

***vxAtomicModify*( )** - interface to the i80960 ATMOD instruction
***vxCacheInvalidate*( )** - invalidate the instruction cache
***vxSysctlSend*( )** - send a SYSCTL message to the processor
***vxIACSend*( )** - send an IAC message to the processor
***vxACWGet*( )** - read the ACW
***vxPCWGet*( )** - read the PCW
***vxTCWGet*( )** - read the TCW
vxIMRClear( ) - clear the interrupt mask register
***vxIMRSet*( )** - set the interrupt mask register
***vxIPNDClear*( )** - clear the given bit(s) in the IPND register
***vxIPNDGet*( )** - return the IPND register
***vxICONGet*( )** - read the ICON register from the control table
***vxICONSet*( )** - set the system ICON (interrupt control) register to value in g0
UINT8 vxAtomicModify (value, pMem, mask)
vxCacheInvalidate ()
vxSysctlSend (f12, f3, f4) ...
vxIACSend (iac) ...
vxACWGet ()
vxPCWGet ()
vxTCWGet ()
vxIMRClear (mask)
vxIMRSet (mask)
vxIPNDClear ()
vxIPNDGet ()
vxICONGet ()
vxICONSet ()

DESCRIPTION

This module contains miscellaneous Vx960 i80960 support routines written in assembly language.

SEE ALSO
vxLib

NAME

vxAtomicModify( ) - interface to the i80960 ATMOD instruction.

SYNOPSIS

```
UINT8 vxAtomicModify (value, pMem, mask)
        UINT8 value; /* value to place in memory */
        UINT8 *pMem; /* location to modify */
        UINT8 mask; /* bit mask for value */
```

DESCRIPTION
This routine atomically modifies a location in memory.

RETURNS
The original value in the memory location.

SEE ALSO
vxALib

NAME

vxCacheInvalidate( ) - invalidate the instruction cache

SYNOPSIS
vxCacheInvalidate ( )

DESCRIPTION

This routine invalidates the instruction cache. The entire instruction cache is flushed.

SEE ALSO

vxALib

NAME

vxSysctlSend( ) - send a SYSCTL message to the processor

SYNOPSIS

```
vxSysctlSend (f12, f3, f4)
        UINT32 f12; /* arg0 contains field2 messageType field1 */
        UINT32 f3; /* arg1 contains field3 */
        UINT32 f4; /* arg2 contains field4 */
```

DESCRIPTION

This routine sends a SYSCTL message to the processor. This is for the I960CA processor only.

SEE ALSO

vxALib,Chapter 2 - Programming Environment of the 80960CA User's Manual

---

NAME

vxIACSend( ) - send an IAC message to the processor

SYNOPSIS

vxIACSend (iac)

UINT32 *iac; /* arg0 contains pointer to iac structure */

DESCRIPTION

This routine sends an IAC message to the processor. This is for the I960SX or I960KX processors only.

SEE ALSO

vxALib, Chapter 11 - IACs 80960SA/SB Reference Manual

NAME

vxACWGet( ) - read the ACW

SYNOPSIS

vxACWGet( )

DESCRIPTION

This routine returns the value of the Arithmetic-Controls register.

SEE ALSO

vxALib, Figure 2-4 in 80960CA User's Manual

NAME

vxPCWGet( ) - read the PCW

SYNOPSIS

vxPCWGet( )

DESCRIPTION

This routine returns the value of the Process-Controls Register.

SEE ALSO

vxALib, Figure 2-5 in 80960CA User's Manual.

NAME

vxTCWGet( ) - read the TCW

SYNOPSIS

vxTCWGet( )

DESCRIPTION

This routine returns the value of the Process-Controls Register.

SEE ALSO

vxALib, Figure 2-5 in 80960CA User's Manual.

NAME

vxIMRClear( ) - clear the interrupt mask register

SYNOPSIS

```
vxIMRClear (mask)
        UINT32 mask; /* bits to clear in interrupt mask reg */
```

DESCRIPTION

This routine clears bits in the interrupt mask register. This is for the I960CA processor only.

SEE ALSO

vxALib, Figure 6-12 80960CA User's Manual

NAME

vxIMRSet( ) - set the interrupt mask register

SYNOPSIS

```
vxIMRSet (mask)
        UINT32 mask; /* bits to set in interrupt mask reg */
```

DESCRIPTION

This routine sets bits in the interrupt mask register. This is for the I960CA processor only.

SEE ALSO

vxALib, Figure 6-12 80960CA User's Manual

NAME

vxIPNDClear( ) - clear the given bit(s) in the IPND register

SYNOPSIS

```
vxIPNDClear( )
```

DESCRIPTION
This is for the I960CA processor only.

SEE ALSO
vxALib, Figure 6-12 80960CA User's Manual

NAME

vxIPNDGet( ) - return the IPND register

SYNOPSIS
vxIPNDGet ( )

DESCRIPTION
This is for the I960CA processor only.

SEE ALSO
vxALib, Figure 6-12 80960CA User's Manual

NAME

vxICONGet( ) - read the ICON register from the control table

SYNOPSIS
vxICONGet ( )

SEE ALSO
vxALib, Figure 6-10 80960CA User's Manual or Chapter 5 Interrupts in the 80960SA/SB Reference Manual

RETURNS
Current value of ICON register

NAME

vxICONSet( ) - set the system ICON (interrupt control) register to value in g0

SYNOPSIS
vxICONSet ( )

SEE ALSO
vxALib, Figure 6-10 80960CA User's Manual or Chapter 5 Interrupts in the 80960SA/SB Reference Manual

RETURNS
Previous value of ICON register