EVE Tech Notes:

Sections Include: **On Page:**

1. How to Build EVE EPROMs:

1) From $PAM/pam3/elf: make pam3.st.o

2) From $PAM/pam3/vx/config/dcom332 make CPU=MC68010 TOOL=gnu eve.hex

   (NOTE: Check config.h file to make sure it is setup for EVE code and not for vxWorks. See info on config.h.)

3) Copy "eve.hex" to PC which has the pilot prom programmer on it (/ap/advin is PROM directory). Make sure ftp is in ASCII mode.

4) On PC, open DOS window:

```
cd c:\ap\advin
advin
```

5) Select "spEPROM" and hit return.

6) /C(onfig) W(idth) M(otorola Way)

7) /F N(ame) "eve.hex"

8) /F F(ormat) S(-record)

9) /F L(oad)

10) /P P(rogram)

2. *EVE VxWorks Info.*

Currently using 5.1.1

Could upgrade to 5.2 (see vx/config/dcom332/READ.ME for Dynatem ethernet card info).

IMPORTANT: Before trying to compile stuff for vxWorks, do:
"source $PAM/pam3/bin/vx51_env"
This sets up: "alias vxmake make CPU=MC68010 TOOL=gnu"
Which may need to be changed for different CPU’s

config.h $PAM/pam3/vx/config/dcom332
Must be modified between EVE make and vxWorks chips. Search for the "/* PAM debugging stuff " string near the end of the file. For vxWorks use "#if 1" and for EVE use "#if 0". This could probably be stuffed into the make file with a compiler define, but it has always been easier to do it by hand, because there are so many dependencies on "config.h".

NOTE: BOOT_LINE definition for Vx startup over the network, including the address of the host to get it from, the devel system addresss, user, and startup script locations. Example:

```
#define DEFAULT_BOOT_LINE \
"es(0,0)toucan:/net/pam/pam3/vx/config/dcom332/vxWorks e=128.117.80.6
      h=128.117.80.71 g=128.117.80.251 f=08 tn=elf
      s=/net/pam/pam3/elf/startup.dcom u=militzer$
```

../all/usrConfig.c #ifdef PAM_ROM - has hook to start eve code on boot

MakeSkel file is used to generate the Makefiles. When upgrading to vx5.2, this will be used to create the new vx5.2 "Makefile.MC68010gnu".

Makefile.MC68010gnu. This has been modified after its initial creation to
have all the EVE code dumped in. See the eve.hex target
and such in this file.

Various make targets.

Note: Vx Normally is compressed in ROM and on bootup uncompresses
its kernel copying itself and executes out
of RAM. Depending upon the make, can either be a
networked download or standalone version with or without

the shell.
ROM Resident Versions: Vx executes from the ROM.
This leaves more RAM for user development code applications.
Runs a little slower due to slower ICs.
Generally requires a larger EPROM (512).
Resident versions for either stand-alone or networked operation can be built

IC speed: EPROM 27C4096 = 120nS (larger 512kb ROM for resident versions)

vxmake   Creates a vxWorks.hex file for burning into EPROM.
This is the normal development EPROM. It runs out of
RAM and download the symbol tables, etc. from the network

vxmake vxWorks.res_rom.hex -- Standalone, ROM resident version of
vxWorks which includes the shell. Executes from the
EPROM rather than uncompressing into RAM.
It boots much faster and leaves a lot more room
in RAM for development code. In order to make this, the EPROM
size must be set to 0x8,0000 in both "config.h" and
"Makefile.MC68010gnu". Normally this is at 0x4,0000 for the
standard 27C2048 EPROMs.

vxmake eve.hex -- Standalone, EVE Version for burning into EPROM and PAM
station. Does not include the Vx shell, networking,
debugging, spy/monitoring, etc.
This version uncompresses and runs from RAM.

vxmake eve.res_rom.hex -- Standalone, ROM resident, EVE version.
Executes out of ROM and does not include the
shell.   It loads faster, but requires the
27C4096 EPROMs.
There may be some performance issues which need to be
monitored due to the slower ROMs.

EPROM sizes: Are changed in Makefile.MC68010gu and config.h.
Can move from current size 27C2048 to a 27C4096.

Math Library

The vxWorks math library has been modified for the DCOM332 board.
The reason is as follows: The 68332 processor uses 99% of the 68020

instructions.   However, vxWorks is told the processor is a 68010. The 68020 math library can do a 32 bit divide instruction, but the 68010 math library must use 16 bit divides. Thus the change.

Math libraries: Pulled out of 68020 and put into 68010 library.
"gccMathALib.o" Library is "libMC68010gnuvx.a".

BUGS in vxWorks 5.1. Math

    The single precision math library is partially crap.
    Some effort was made a couple years ago to have Wind River deal
    with this, but they claimed the math library was purchased from
    a 3rd party, and they couldn't fix it. Work arounds have been
    used in the EVE code.
    Mostly, double precision routines have been used instead of bad
    single precision ones.

Vx Board Support Package:

  System Library: vx/config/board/sysLib.c

    Initialization functions
    Memory / Address space functions
    Bus Interrupt functions
    Clock / Timer functions
    Mailbox / location monitor functions

    Not all of these are used with any specific board.
    Some boards won't have the capabilities, others may have
    jumpers instead.j

    The user does not typically call these, they're done through the
    config modules usrConfig.c and bootConfig.c
    And set up with the config files:

  tty Driver

    tyCoDrv.c provides console driver
    User does not call.
    Interfaced through usrConfig.c and bootConfig.c

Configuring VxWorks

  vx/config/all/usrConfig.c   Compilation of this includes these
                              definition header files:...

  vx/config/all/configAll.h   Selects Vx options and features (global)

vx/config/-board-/config.h Selects Vx options and features (target-specific)
This file can redefine modules included/excluded from
the global config file which are inappropriate for the board.
default boot parameter string for ROMs: ie
user/target-id/host-id to load from

Board Dependent Libraries

==IN VXWORKS DIR==

Serial Card Drivers
- z8530Serial.c   (not currently used--requires same IRQ as rtc)
- m68681Serial.c
- m68332Serial.c
- tpuSerial.c
- vser80drv.c

Real-Time Clock Library

DCOM332 also has tpuPulseCntLib.h

==IN ELF DIR==

syncLib.h -- Provides task real time task scheduling based on
IRQ from real time clock.   DCOM332 base at 1 sec.

Other Libraries
----

pcmcia -- Driver for DMEM20 PCMCIA. This driver can be compiled with or without interrupts for read/write. Just change "#undef USE_INTR" to "#define USE_INTR". Has been tested with 5MB and 10MB cards, but will also need to work with 40MB cards for GAME. Hopefully there shouldn’t be any problems. If there are, the most likely source is compatibility between a vxWorks format and an MS-DOS version. There is a distribution version of this code under pcmcia/dist which contains a README file.

xmodem -- Xmodem. This has been pretty stable. One minor change made in last couple years to fix Ymodem protocol conflict with Procomm Plus.

fft   -- FFT for bandpass Covar. Stable.

dirlib -- Wildcard handling routines. Stable.

==

3.   ***EVE Code Overview***:

==

These are somewhat old notes left by Matt...

Directory Descriptions (under /net/pam/src/elf)
----

goes	Stand alone code for running the SE110 transmitter from workstation.
sio-msgs	Definitions for fake serial sensor messages to use for testing.

File Descriptions (under /net/pam/src/elf)

Makefile	Current Makefile for DCOM332 based EVE
	These next are now in a subdirectory /Makefile.saves
Makefile.dcom332	Older version of Makefile for DCOM332
Makefile.pvsbc2	Older version of Makefile for PEP PVSBC2 based EVE, Note this is a MC68000 so it may not handle the extra processing of the current EVE software.

IMP232Params.h	Communication parameters for TRIME IMP2323 bus
adaptFcast.h	Despiker data structure
data_struct.h	Main set of data structure files for EVE
dirLib.h	Include file for dirLib wildcard handling
feve_global.h	Assorted global defines
floatNum.h	Definition for bad Float value--this has been superceded
fstatus.h	Status parameter types and structures
goesUtilRtns.h	Odds and ends of global routines. Originally John M.'s
pollParams.h	Polled sensor timing definitions
sBusParams.h	Sensor bus standards
standard_defs.h	John's standard stuff used for GOES transmitter code
sync.h	Task synchronizer structure def
syncLib.h	Task synchronizer routines
tgtXmtrCodes.h	Definitions for TGT GOES transmitter (not currently used)
transmitter_defs.h	Definitions for SE100 GOES transmitter
hashFunc.h	Hash library routines

tgtXmtrCode.c	Telonics GOES transmitter code (not currently used)
tgtControl.c	Routines for TGT transmitter (not currently used).
xmtrCode.c	Communication routines for SE100
goesSio.c	Read/Write routines to SE100.
goesControl.c	Auxiliary control routines for SE100 transmitter.
goespack.c	Generates PPF type data packet
goesUtilRtns.c	Library of assorted routines. Originally John M.’s routines
goesXmit.c	Task to send data to GOES transmitter (#ifdef for TGT)
goesXmitProcess.c	Parses a "GOES:" command. Originally part of feve.c.
sync.c	Real-time clock task synchronizer.
sioTalk.c	Serial port talk routine.
sBusTalk.c	Sensor bus talk routine.
cfft.c	FFT routines
tickClockSyncer.c	Synces vxWorks tick() with real-time clock.
argCnt.c	Counts scanf, printf arguments.
imkoTalk.c	TRIME, IMP232 bus talk routine.
pollTalk.c	Polled sensor talk routine.
feve.c	Main routine. Configuration loader/unloader.
fixFormat.c	Converts \n, , etc. to the appropriate character
stats.c	EVE statistics routines (ie DO commands).
async_in.c	"SERIAL:" sensor task code.
msg_maker.c	Creates and sends field terminal messages.
ftertas.c	Processes requests for field terminal messages.
fhantas.c	Old code for Handar GOES transmitter. Not used.
fstatus.c	Status parameter handling.
readData.c	Reads EVE data, with optional reset code for final output.
getString.c	Configuration file string reader.
bbram.c	Initializes the MS-DOS file system on battery backed RAM.
sysTerm.c	System terminal interface commands.
idle.s	Idle task -- executes a stop instruction to save power
dataSync.c	data time synchronizer code.
packTask.c	Handles "OUTPUT:" command to munch data at specific times.
fileSave.c	FILE: command handling. Saves data to disk or other device.
analog_in.c	Analog input sensor code.
adBoard.c	O+R VADC21 A/D input.
binSioIn.c	"BINSERIAL:" routine for binary serial data.
sBusIn.c	"SBUS:" routine for sensor bus input.
pulseIn.c	"PULSE:" routine to read pulse counts.
func.c	Set of math routines for "CREATE:" command.
bpCovar.c	Bandpass Covariance routine.
imp232In.c	TRIME sensor input.
adaptForecast.c	Despiker routine.
auxConsole.c	Allows establishing an auxiliary console port.
pcCard.c	Initialization for PCMCIA card.
pollSio.c	"POLL:" sensor input.

outputProcess.c   Parses "OUTPUT:" command. Originally part of feve.c.

trimMean.c   Trimmed mean calculation for bpCovar.c
hashFunc.c   Hash table routine.

EVE Development Boot Up / Environment

BOOT PROCEDURE: for checking out new code on the development system 'Elf'.
        The vxWorks bootline is normally set to load and execute the startup file
        "$PAM/pam3/elf/startup.dcom".   This in turn does a
        "cd /net/pam/src/elf", and some of the serial drivers are started.

To talk to the development system "elf", either do an "rlogin elf" or talk
over the serial console port either with Pcplus on a PC or Kermit on the Sun.

To load and EVE type at the command prompt:
        "<leve"

The development version of EVE is broken down into three object files
which must load in the order given. The problem with combining
everything into a single file, is that if debugging info is created (by
compiling with -g (or make ___.g)), then there may not be enough RAM to
load the whole mess. (This may be somewhat improved with
vxWorks.res_rom.hex.)

Debugger is awkward to use with Vx5.1. Requires "make" twice to use it.
After EVE successfully loads type:
        "sp eve"
This should start things rolling. If using the console port and not an
rlogin, it will be necessary to type "taskSuspend 0" after EVE starts.
This causes the vxWorks shell to suspend itself so that the "EVE>"
prompt can be accessed. A "^C" will revive the shell.

Vx Boot Up Description:

```
  1) get Vx into main memory (boot ROM)
  2) sysInit()         Vx startup entry point.  (See RAM_LOW_ADRS)
                       found in sysALib.s, locks interrupts, sets stack
                       ptr, jumps to usrInit...
        [usrConfig.c    of Vx has 3 routines including ]
  3) usrInit()         First routine to run under vx.  Initializes
```

basic environment: cache, zeroes mem (bss segment), initializes interrupt and exception vectors (base addr) initialize/start multi-tasking kernel (TCBs, stack, lock level).
Note: calls kernelInit() which initializes the all important memory pool for dynamic allocation of practically everything.

4) usrRoot() first task to run under multi-tasking environment. For most of these, see the config setup and the INCLUDES there in configAll.h

   a) usrClock() Initializes system clock and connects to interrupt.
      Note: the system clock rate per Vx can be set to 50,60 or 100 ticks/sec. With PAM/DCOM332 it’s 60.
   b) i/o system no. of drivers, files.
   c) create console, stdin/out/err
   d) exception handling, signals, debugging and logging
   e) standard i/o
   f) file system device creation (dos file system for PAM)
   g) floating point
   h) performance monitoring / spyLib and subroutine exec timers timexLib
   i) network initialization (not in PAM_ROM), via INCLUDE_NET_INIT and uses the config statement, generally the ’boot line’ noted above with config.h
      Some of the network facilities loaded are also included in the configAll.h file
   j) Optional stuff:
   k) symbol table is created and loaded: ie names/addresses of functions and variables. For standalone systems (ie PAM_ROM) it is not loaded over the net but included with the rom.
      For devel. systems this includes the symbols of Vx accessible via the shell (vxWorks.sym)
   l) executes the startup script via INCLUDE_STARTUP_SCRIPT in configAll.h and the boot line in config.h For PAM this is $PAM/pam3/elf/startup.dcom
   m) spawns the shell for development environment (ie not PAM_ROM) or else calls ’eve_main’ (for PAM_ROM) which gets the user code going....

5) eve_main() The main EVE entry point, which does some additional system checkout.

   a) Initialize and verify real-time clock time.
   b) Load the serial drivers on the DCOM332 board

c) Load the vser80 serial driver if board is present
e) Check if the A/D board is present
f) Initialize the battery backed RAM disk.
g) Check if PCMCIA board is present and initialize that.

Then a PROM time/date is printed and the configuration load is started.

The configuration loader uses a getString() routine to grab lines of text. The getString routine handles comments and "<file.dat" sub files. The first word of a line is compared to a list of EVE commands. If a match is found, the routine which handles parsing and programming for that command is executed.

How-to For EVE Modifications

Adding new statistics routine (EVE "DO:")

A statistics routine has a single output variable, but may have more than one input parameters. Each statistics function requires three routines: an in, an out, and an init. The in routine is accessed to add new data. The out routine is called to output the value, and the init is called to re-initialize all the variables. All statistics are in the "stats.c" file.

Look at "avgIn", "avgOut", and "avgInit" for a good example. After the routine has been written, add everything to the structure at the end of the file. Recompile, and presto.

Adding a new calculation (EVE "CREATE:")

A calculation can have one or more input variables, one or more output variables, and one or more constants. These routines are also easy to add. Simple functions should be added directly to "func.c". See any of the existing routines for examples on how things work. The new routine must be added to the structure at the end of the file for the whole thing to work.

There are two complex "CREATE:" commands, the despiker and the bandpass covariance. These routines are in separate files, but they are referenced in the structure too.

A (*startFunct)(void *[]) is called for complex data initialization requirements, and then on an "EVE stop" command the (*killRoutine)(void *[]) is called for cleanup. The simple functions don’t use this.

Changing Bandpass Covar

One anticipated change to the bandpass routine is to input air temperature for calculation of the time constant rather than relying on the sonic temperature.   This will require modifying the funcTable[] structure at the end of the "func.c" file to add the new parameter. Also, the pointers in bpCovar.c must be shuffled a bit to squeeze in the new parameter. Then use the air temperature instead of the sonic temperature in the routine.

A second change is massaging the data a bit more when Tom tweaks the algorithm. Currently, the code has an "#if" around the section to find the time constant using the spectral information. This bit has not been tested, and probably should be disabled until it has been verified.

Adding new sensor.

This is a bit of work. The best thing to do is examine the "async_in.c" code for the "SERIAL:" command.   Here is a list of things needed.

1) New data structure for sensor specicific information.
   This structure should be combined with the generic sensor structure "SENSOR".

2) New sensor routine. Copy, modify async_in.c for ease.

3) New command parsing routine. This has been placed in "feve.c" historically, but a paradigm might be to put this in the new sensor routine file. The jobs of the sensor parsing routine is to first read all the command information and make sure there are no mistakes. Then a new sensor structure is created, and the requesite information is placed there.

4) The sensor structures are linked lists. There is a general sensor head, and a specific head for each type of sensor. A new head will be needed for the new sensor.

5) The code to spawn the sensor task is located in a different section of the feve.c file. New code to spawn the new sensor will be needed.

6) Each sensor has a single character type assigned to it. These are

kept in data_struct.c. Some argument can be made to change the whole mess to an enum() statement, but this would be more work. Anyway the new sensor will require a new character type to identify it.

7) Several places in the feve.c file are marked with "^^^" in comments. These are places where the data from the sensor structure is needed for one reason or another. If the sensor is not too unorthodox, it should be possible to add the new type in fairly easily.

8) In addition to the sensor creation code, additional code is needed to free the memory required by the sensor. This is normally placed after the command parsing/creation code.

9) Code to do a taskDelete() on the sensor and hand off the structure deletion is also required. This is more to the front of feve.c

4. *VME / DCOM332 Addresses and Interrupt Vectors:*

--------------------- Update this from the text file !!! ----------------------

Addresses

Jumper W2	For DCOM332 setting address of 2MB dual ported SRAM.
0x800,000 -- 0xFFF,FFF	VME Standard Address Range. This is mapped from VME A24 space 0x000,000 -- 0x7FF,FFF. ie the DCOM sees it as 0x800,000 - 0x9FF,FFF for the 2MB (check)
0x7FF,E00 -- 0x7FF,FFF	68332 TPU, (used for serial, pulse inputs and system timer).
0x7FF,C00 -- 0x7FF,DFF	68332 Queued Serial Module
0x7FF,B00 -- 0x7FF,B3F	68332 Standby SRAM Control
0x7FF,A00 -- 0x7FF,A7F	68332 System Integration Module
0x7FF,000 -- 0x7FF,7FF	68332 2K Internal SRAM -- Used in EVE for TPU serial driver microcode.
0x700,000 -- 0x77F,FFF	EPROM space (Max of 512K or 27C4096 EPROM) Moved from Normal 000,000 on reset by changing CSBARBT to 0x7006
0x600,000 -- 0x60F,FFF	VMEbus Short I/O (A16) Space, ie address modifiers set accordingly and 16-bit address are set on the bus.
0x260,000 -- 0x260,0FF	VLX Bus Interface (Not currently used.
0x240,000 -- 0x240,00F	68692 DUART
0x200,000 -- 0x200,00F	MSM6242 Real-Time Clock Registers
0x000,000 -- 0x1FF,FFF	2 MB SRAM. Battery backed.

SRAM Address Space (2mB)

0x000,000 -- 0x17F,EFF	SRAM local address
0x17F,F00 -- 0x1FF,EFF	Battery Backed RAM disk 512K (0x80000 bytes in 1024 blocks of 512 each, using 1 logical track). See file bbram.c for init routine and the 'ramDevCreate' call which does this. 1024, 512 byte blocks in 1 logical track.
0x1FF,F00 -- 0x1FF,FFF	vxWorks non-volatile RAM boot-line

VMEbus Address Space

Note: DMEM20 is normally programmed to read the bottom half of VMEbus address space. Therefore, a board at 0x100,000 would be seen at 0x900,000 by the DCOM332. A board at 0x900,000 can't be addressed without toggling the VME A24 line on the DCOM332. See the DCOM332 manual for more information.

0x200,000	DMEM20 PCMCIA A24 space. This space is addressed by opening a memory window to a PCMCIA card.

VMEbus I/O Space (Short A16 space)

Note: A board at short address space 0x1000, is seen at 0x601,000 by the DCOM332.

0xC000	PROPOSED VME RESET LOCATION

	C000 = reset
	C001 = clear watchdog
	C002 = toggle power circuit 1
	C003 = toggle power circuit 2
0x8000	Dynatem DLAN Adapter
0x4000-0x5FFF	DMEM20 Card Base (I/O space), includes:
0x4240	DMEM20 TCIC chip Base
0x2600	O+R VSER80 8 Channel Serial Card
0x1A00	O+R VADC21 A/D Board
0x000000	EEPROM,
	1st 4 bytes = Stack Pointer
	2nd 4 bytes = Reset Vector

Interrupt Vectors

Vector	Level	Device
0x18		Spurious Interrupt
== IRQ 0x19 to 0x1F are autovectors 1--7 ==		
0x19	1	N/A
0x1A	2	Mailbox
0x1B	3	N/A
0x1C	4	68692 DUART
0x1D	5	RTC, VLXbus (RTC is used by EVE, so VLX won't work)
0x1E	6	N/A
0x1F	7	Abort
0x40	5	68332 Serial
0x41		68332 SPI (reserved, but not currently used)
0x42	6	Clock tick (used for vxWorks auxClock and EVE A/D)
0x50	5	TPU chan 0 (EVE /tyCo/3 RX)
0x51	5	TPU chan 1 (EVE /tyCo/4 RX)
0x52	5	TPU chan 2 (EVE /tyCo/5 RX)
0x53	5	TPU chan 3 (EVE /tyCo/6 RX)
0x54	5	TPU chan 4 (EVE pulse 0)
0x55	5	TPU chan 5 (EVE pulse 1)
0x56	5	TPU chan 6 (EVE pulse 2)
0x57	5	TPU chan 7 (EVE pulse 3)
0x58	5	TPU chan 8 (EVE /tyCo/3 TX)
0x59	5	TPU chan 9 (EVE /tyCo/4 TX)
0x5A	5	TPU chan 10 (EVE /tyCo/5 TX)
0x5B	5	TPU chan 11 (EVE /tyCo/6 TX)
0x5C	5	TPU chan 12 (EVE /tyCo/3 RTS)
0x5D	5	TPU chan 13 (EVE /tyCo/4 RTS)
0x5E	5	TPU chan 14
0x5F	5	TPU chan 15 (vxWorks sysClk)
0x75	5	Dynatem DLAN Adapter
0x80	3	DMEM20 PCMCIA Board
0x81	3	DMEM20 PCMCIA Board

0x82	3	DMEM20 PCMCIA Board
0x83	3	DMEM20 PCMCIA Board
0x84	3	DMEM20 PCMCIA Board
0x85	3	DMEM20 PCMCIA Board
0x86	3	DMEM20 PCMCIA Board
0x87	3	DMEM20 PCMCIA Board
0x88	3	DMEM20 PCMCIA Board
0x89	3	DMEM20 PCMCIA Board
0x8A	3	DMEM20 PCMCIA Board
0x8B	3	DMEM20 PCMCIA Board (Card Insert/Remove IRQ)
0x8C	3	DMEM20 PCMCIA Board
0x8D	3	DMEM20 PCMCIA Board
0x8E	3	DMEM20 PCMCIA Board (ATA IRQ)
0x8F	3	DMEM20 PCMCIA Board
0xD0	3	VSER80 8 Channel Serial Card
0xD1	3	VSER80 Modem Interrupt
0xD2	3	VSER80 TX Interrupt
0xD3	3	VSER80 RX Interrupt
0xD7	3	VSER80 Bad RX Interrupt

5. VME BUS / Development Notes:

Here are a few things to keep track of when using and/or designing an interface for a VME system. First, a good reference to the VME bus is needed. For this access the Vita web site:
http://www.vita.com/
This site has pointers to manufacturers, documentation and more. A good reference is available from them:
The VME bus Handbook (A Users buide to the VME64 and VME64X bus specifications) by Wade D. Peterson
Tid-bits from that book:

- On an interface, put chips which have signal lines to the bus within 2” of the connector and put no more than 2 IC’s on 1 bus line.
- Bypass Capacitors of .01 or .1 f should be placed on all backplane terminators.
- Capacitive loading on the bus should be kept <17pf
- Bipolar TTL receivers need input clamping diodes on all lines: A1-15, AM0-AM5, DS0,DS1,Sysclk, Write, D0-D7

6. VME Bus Errors:

Errors of the type reported by VxWorks:

```
Bus Error
Invalid ESF type 0xc
Task: 0x133fa0 "tTermIn"
```

Can be caused by more than one type of error. Some are caused by failure of the VME bus when either the slave module fails to assert DTACK* low (data transfer acknowledge) or else drives the

BERR* (bus error) signal low or else the bus timer module on the cpu drives it low. Another is when a task overflows its stack allocation. These type messages are only reported to the system console, and not via the general purpose log message utility and as a result cannot be captured and recorded in the PAM system log file. They are generated by modules in the Vx Library:

"$PAM/pam3/vx/lib/libMC68010gnuvx.a"

as the result of the hardware induced error signals. When this happens the calling task is suspended by Vxworks and it cannot continue running until resumed. This can be observed by using the "i" command and noting that the condition of the task "0x133fa0 tTermIn" is indeed suspended. On the Elf development system (or one with the Vx monitor running) this task can be resumed via the command "taskResume 0x133fa0." On newer versions of EVE, it can be restarted with the 'entersys' command "tr 0x133fa0." Of course, on the example given above, the task which was suspended is the terminal input task, so on EVE the operator would be unable to talk to EVE to do this. On the bright side the remaining system tasks are able to run so data acquisition is generally not interrupted. However it is a good idea for an operator to check which task is suspended if that is possible an attempt to restart it. The pcmcia lock-up problem is sometimes caused by this type of error.
If a stack overflow error is suspected, the command "checkStack" will show which any particular task has suffered this type of error.

Signal Handlers
Special routines under VxWorks can be set up to catch and process these errors in a more graceful way using the **sigLib** software signal facility library. These require references to the signal.h definitions. For the Bus Error shown, this is defined as SIGSEGV. A signal handler is specific for the task it was setup for. That means potentially for bus errors generated by slave boards, a different handler will be needed for each. The basic appearance of this code (valid for VxWorks 5.1) is:

```
#include "signal.h"
#include "sigLib.h"
#include "setjmp.h"

static jmp_buf   goHere;
void sigHandler();
SIGVEC newSignal;


VOID test1()
{
   unsigned int   mem, val;
   short   *addr;
   unsigned char  ival;

   /* Setup for Exception Signal Handling.
    * Establish signal handler first.
    * Initial setjmp call returns 0, and represents the place
    * in the code where the execution environment is restored to.
    * longjmp from signal returns !=0 and conditional code is executed.
    */
   newSignal.sv_handler = sigHandler;
   newSignal.sv_mask = 0;
   newSignal.sv_flags = 0;
   sigvec( SIGSEGV, &newSignal, NULL); /* For Bus Error */
   if (setjmp(goHere) != 0)
   {   printf("\nSignal Return from longjump");
   }

   /* Read and process forever... */
   while( 1 )
   {
      here is the main task execution flow;
   }
}


void sigHandler()
{
   printf("\nThis Works! signal capture from sigHandler");
   longjmp(goHere,-1);
}
```

Bus Errors and the DMEM20 PCMCIA Board:
Bus signal timing is marginal for the dmem20 board and as a result it is not unusual to encounter bus errors with it. As an example, in one case the DMEM20 responded to the routine "boardPresent" properly while it was sitting in the VME chassis in it's normal postion between the CPU/IO and the VADC boards; however, when placed on the VME extender board at the end of the chain, it failed to respond properly and a bus error occurred. Other situations have produced the same type errors such as extremely cold operation (<-5-10C) in the temperature chamber and in the Arctic Sheba program. This is presumed to be caused by poor DTACK signal response timing to the CPU access (read/write is initiated by the cpu setting AS*, address-strobe low, when the VME address is setup and in response it requires the DMEM20 slave to reply with DTACK). It may be related to the slightly high system clock speed used on the DCOM332 although this is pure conjecture.
Note: why the signal handler in the "boardPresent" routine did not catch the occurance noted above needs to be determined.

7. VME Static RAM: bootup errors, 991123

Note: During testing of the 4 new DCOM332 and DMEM20 boards Steve Horan encountered an 'always' bootup failure using the EVE-PROM 990925 version (same as CASES99 with fastout, cardcopy, soft-reboot). EVE would never boot up, halting at the:

Setup DCOM332 serial ports
Look for VSER80 board

vicinity of the code. Steve reports that the only way he could get it to boot was by removing the SRAM battery, thereby flushing it totally, and then proceeding. With the battery in, the DCOM332 would not boot. My observation is that this is probably in the VME bus access portion where 'boardPresent' is called for the VSER80, VADC21, DMEM20, etc. That is based on my own experience with commenting that code out and seeing the soft reboot work. MORE INVESTIGATION NEEDED..... Sending temp. PROM to Steve with those boardPresent calls commented out.

8. SIZEOF storage allocation:

```
Size of int      = 4
Size of short    = 2
Size of long     = 4
Size of char     = 1
Size of unsigned = 4
Size of float    = 4
Size of double   = 8

Max. of int      = 2147483647
Min. of int      = -2147483648
Max. of short    = 32767
Min. of short    = -32768
Max. of long     = 2147483647
Min. of long     = -2147483648
Max. of char     = 127
Min. of char     = -128
Max. of unsigned = 4294967295
```

9. ***COVAR Accuracy, Negative Covars and Usage of COVARDP:***

Negative EVE Covars can be erroneously generated!
This situation occurred frequenctly during IHOP-02 at station #1 when covars for a CO2 sensor were being calculated; specifically the variance of co2'co2'. The problem can be encountered in 2 ways:

- At the next 'output' cycle after EVE does either a start/stop, or else when the sensor itself starts/stop its data output to EVE.
- When the scalar mean of the data parameter is large with respect to its fluxuations.

The problem is due to improper mathematical precision (round-off error);
i.e. Covars generated by Task: NONE COVAR COVARDP

$$\overline{c^2} - (\bar{c})^2$$

When

$$(\Delta c \ll c)$$

The problem can be simulated by sending EVE 'fake' sensor data

The solution to the problem is to use EVE double-precision covar calculations. EVE has 3 different methods for calculating covariances (see 'stats.c' code for where this is done)

COVAR	Single Precision
COVARDP	Double Precision
COVARXP	Mixed Precision.

Normally, single precision covar is sufficient except in the situations mentioned. In those cases, COVARDP overcomes this problem.There are as yet un-ivestigated issues involving residual noise in the normal covars. The relative magnitude of these is considered insignificant compared with the magnitude of the covars themselves.

The cost of using covar/covardp with EVE is in processing overhead. In measured approximate percentage of overall cpu cycles:

EVE-Task	No-Cov	COVAR	COVARDP
Fastout	3%	3%	3%
Sonic Ingest	20%	30%	40%
Sync	na	6%	8%
Interrupts	30%	30%	30%

These may be inflated numbers but represent the significant processing burden associated with generating approximately 13 covariances for a sonic anemometer:
uu,uv,uw,utc,vv,vw,vtc,ww,wtc,uco2,vco2,wco2,co2co2.

10. Scale and Bias of Calibrations:

scale = slope in Y=aX+b

When the term calibrated is used, it means that a data value is in engineering units. For instance a net radiation of 450 W/m**2 is a calibrated value. An uncalibrated value generally means a data parameter in some intermediate form. An example might be net radiation of 23.4 mV.

11. Signed and Unsigned quantities:

The terms “SIGNED” and “UNSIGNED” are used in the configuration file as parameters for creating the pseudo-ASCII values which are transmitted via GOES satellite. If a value is “SIGNED” then when converted to an integer, it must fit within the range of signed integer with the right number of bits. If it is “UNSIGNED” it must fit within the range of an unsigned integer with the right number of bits. When a floating point number is converted to an integer, EVE makes sure the integer is small enough to fit in the proper number of bits. If it is too big, then a special “BAD VALUE” is used instead. Here is an example to demonstrate how the parameters for conversion to an integer are calculated:

Temperature.
Desired range is -30 DEG C to +50 DEG C
Desired resolution is 0.02 DEG C
-Find the proper multiplier ‘a’ (or slope) for converting the calibrated temperature to an integer value.
 a = 1 / .02
 a = 50
-Find the proper size of pseudo-ASCII value for the conversion.
 1 byte = 6 bit integer = 0 to 63 unsigned or -31 to 31 signed
 2 bytes = 12 bit integer = 0 to 4095 unsigned or -2047 to 2047 signed
 3 bytes = 16 bit integer = 0 to 65535 unsigned or -32767 to 32767 signed
 (Note that 3 bytes can also be 18 bits in PAM III but in PAM II only
 16 bits were used.)

 Imax = (50 - (-30)) * a
 Imax = 80 * 50
 Imax = 4000

 So we can use the 2 byte value since it can go up to 4095.

 -Finally we need to find the offset ‘b’. We have two choices: either we can offset the integer value so it is always positive (unsigned), or we can offset the integer value so that it is either positive or negative (signed). The standard PAM method has been to use a signed value, so we will do that.

Negative MAX = -30 * a
Negative MAX = -30 * 50
Negative MAX = -1500
-2047 < -1500 so this is OK
Positive MAX = 50 * a
Positive MAX = 2500
2500 > 2047 this is not OK.
b = 2047 - 2500
b = -453

We can round off a little bit since we have some extra room
(4095 > 4000).
Thus we declare: b = -500

Temp Min = (-2047 - b) / a
Temp Min = (-2047 + 500)/50
Temp Min = -30.96
Temp Max = (2047 - b) / a
Temp Max = (2047 + 500) / a
Temp Max = 50.94

If the scaled and offset temperature value is less than -2047 (-30.96 DEG C) or greater than 2047 (50.94 DEG C) EVE will use the "BAD VALUE" number instead.

12. Status Parameters:

Although PAM III is currently transmitting about 30 of its status parameters, it can transmit up to 63 parameters. Some of these parameters are internal status parameters, others are simply pre-defined names, and the rest are available for anything else. Here is the list of predefined names including the internal ones. The parameters with their associated number postfixed with a '*' are internal and cannot be assigned from other parameters. The names which are not internal must be assigned from a sensor data or sensor status parameter. The other numbers from 26 to 63 (not shown in the table) can also be used for sensor data or sensor status parameters.

Name	Num	Description
BATT_I	16	Batttery load current.
BATT_V	8	Battery voltage.
BATT_TEMP	15	Battery box temperature.
BOX_TEMP	12	Electronic Box temperature.
C_ERRORS	1*	Code errors.
CHARGE_I	14	Charging current on battery.
CLEAN_MIRROR	11	Dirty mirror signal for dew-point hygrometer.
GIB_ERR	10	Historical PAM II G.I.B. (A/D board) errors.

PRES_ERR	3	Pressure sensor errors.
PRES_ID	2	Pressure sensor ID.
T_RH_ERR	5	Temperature/humidity sensor errors.
T_RH_ID	4	Temperature/humidity sensor ID.
VERSION	9*	Configuration file checksum.
VISIT	13*	Internal Visit status bit-field.
WATCHDOG	7*	Internal Watchdog status (increments by 1 on output).
WIND_ERR	6	Wind sensor errors.
WIND_ID	17	Wind sensor ID.
FORMAT	18*	Configuration format ID.
STATION_ID	19*	Station ID.
YEAR	20*	Real-time year.
JDAY	21*	Real-time day of year.
HOUR	22*	Real-time hour.
MIN	23*	Real-time minute.
SEC	24*	Real-time second.
UNIX_TIME	25*	Unix style 32 bit time.

The EVE data system creates status type parameters for each sensor attached to it. Every serial type sensor has a possible \ID and a possible \ERROR value. The \ID is captured from the input message, and \ERROR is incremented if a message is not read properly. Analog input sensors and pulse input sensors have no \ID value, but they do have an \ERROR value which is incremented if an error occurs while reading data.

The EVE command "STATUS:" assigns the sensor status values to the status numbers up to 63. It also assigns the status priority queue to determine which status parameters are sent most often, and in what order.

13. Logic Families:

74F - Fast Logic:

74F logic is a general-purpose family of high-speed advanced bipolar logic. Provides functions, including gates, buffers/drivers, bus transceivers, flip-flops, latches, counters, multiplexers, and demultiplexers in the 74F logic family.

CD4000 - CMOS Logic
ABT - Advanced BiCMOS Technology
FB - Backplane Transceiver Logic
ABTE - Advanced BiCMOS Technology / Enhanced Transceiver Logic
FCT - Fast CMOS Technology
AC - Advanced CMOS Logic
GTL - Gunning Transceiver Logic
ACT - Advanced CMOS Logic
HC - High-Speed CMOS Logic

High-speed CMOS (HCMOS) offers low power and low noise at a low price. The HC family offers CMOS-compatible inputs and the HCT family offers TTL-compatible inputs.

While HCMOS can be used in most new designs, TI recommends Advanced High-speed CMOS (AHC) as your reliable and effortless migration path from the HC family. AHC delivers the same low noise as HC, with half the static power consumption of HC, at a competitive price.

AHC - Advanced High-Speed CMOS
HCT - High-Speed CMOS Logic
AHCT - Advanced High-Speed CMOS
HSTL - High Speed Transceiver Logic
ALB - Advanced Low-Voltage BiCMOS
JTAG - Boundary Scan Logic
ALS - Advanced Low-Power Schottky Logic

The ALS family provides a full spectrum of over 130 bipolar logic functions. This family, combined with the AS family, can be used to optimize systems through performance budgeting. By using AS in speed-critical paths and ALS where speed is less critical, designers can optimize speed and power performance.

The ALS family includes gates, flip-flops, counters, drivers, transceivers, registered transceivers, readback latches, clock drivers, register files, and multiplexers.

LS - Low-Power Schottky Logic

With a wide array of functions, the LS family is a mature, older, higher power technology. These classic line of devices were at the cutting edge of performance upon introduction and continue to deliver excellent value for many of today's designs.

ALVC - Advanced Low-Voltage CMOS Technology
LV - Low-Voltage CMOS Technology
ALVT - Advanced Low-Voltage BiCMOS Technology
LVC - Low Voltage CMOS Technology
AS - Advanced Schottky Logic:

The AS family of high-performance bipolar logic includes over 90 functions that offer high drive capabilities.

This family, combined with the ALS family, can be used to optimize system speed and power through performance budgeting. By using AS in speed-critical paths and ALS where speed is less critical, designers can optimize speed and power performance. The AS family includes gates, flip-flops, counters, drivers, transceivers, registered transceivers, readback latches, clock drivers, register files, and multiplexers.

LVT - Low-Voltage BiCMOS Technology
AVC - Advanced Very-Low-Voltage CMOS Logic
PCA - Personal Computer I2C Interface
BCT - BiCMOS Technology
lobS - Schottky Logic
CBT - Crossbar Technology
SSTL - Stub Series Terminated Logic
CBTLV - Low-Voltage Crossbar Technology

TTL - Transistor-Transistor Logic
TVC - Translation Voltage Clamp