

Embedded Systems

version 4.9

Kenneth Lundin

Typeset in \LaTeX from SGML source using the DOCBUILDER 3.0 Document System.

Contents

1	Embedded Systems User's Guide	1
1.1	Embedded Solaris	2
	Introduction	2
	Memory Usage	2
	Disk Space Usage	2
	Installation	3
	Starting Erlang	8
	Programs	9
	Other Issues	12
1.2	Windows NT	13
	Introduction	13
	Memory Usage	13
	Disk Space Usage	13
	Installation	13
	Starting Erlang	13
1.3	VxWorks	14
	Introduction	14
	Memory usage	14
	Disk usage	14
	Installation	14
	OS specific functionality/information	15
	Starting Erlang	15
	Card specific functionality/information	16
	List of Tables	17

Chapter 1

Embedded Systems User's Guide

This manual describes the issues that are specific for running Erlang on an embedded system. It describes the differences in installing and starting Erlang compared to how it is done in normal Erlang development system.

Note that this is a supplementary document, you still need the Development System Installation Guide.

There is also target architecture specific information in the top level README file of the Erlang distribution for that target.

1.1 Embedded Solaris

This chapter describes the OS specific parts of OTP which relate to embedded Solaris.

Introduction

The current status of embedded Solaris is that it reduces disk usage, but does not reduce the memory footprint of Solaris.

Memory Usage

Solaris takes about 17 Mbyte of RAM on a system with 64 Mbyte of total RAM. This leaves about 47 Mbyte for the applications. If the system utilizes swapping, these figures cannot be improved because unnecessary daemon processes are swapped out. However, if swapping is disabled, or if the swap space is of limited resource in the system, it becomes necessary to kill off unnecessary daemon processes.

The following start-scripts can be deleted to prevent unnecessary daemons from starting:

- `/etc/rc2.d/S72autoinstall`
- `/etc/rc2.d/S74autofs`
- `/etc/rc2.d/S76nscd`
- `/etc/rc2.d/S80PRESERVE`
- `/etc/rc2.d/S80lp`
- `/etc/rc2.d/S88sendmail`
- `/etc/rc2.d/S92volmgt`
- `/etc/rc2.d/S93cacheos.finish`
- `/etc/rc3.d/S15nfs.server`

More information is expected from Sun on how to modify the kernel in order to reduce the memory consumption. This will be performed by modifying the `/etc/system` file.

Disk Space Usage

The disk space required by Solaris can be minimized by using the Core User support installation. It requires about 80 Mbyte of disk space. This installs only the minimum software required to boot and run Solaris. The disk space can be further reduced by deleting unnecessary individual files. However, unless disk space is a critical resource the effort required and the risks involved may not be justified.

Installation

This section is about installing an Embedded Environment. Solaris 2.5.1 is the only UNIX operating system supported for embedded systems. The following topics are considered,

- Creation of user and installation directory,
- Installation of Embedded Environment,
- Configuration for automatic start at reboot,
- Making a hardware watchdog available,
- Changing permission for reboot,
- Patches for Solaris 2.5.1,
- Configuration of the `os_mon` application.

Several of the procedures described below require expert knowledge of the Solaris 2 operating system. For most of them super user privilege is needed.

Creation of user and installation directory

It is recommended that the Embedded Environment is run by an ordinary user, i.e. a user who does not have super user privileges.

Throughout this section we assume that the user name is `otpuser`, and that the home directory of that user is,

`/export/home/otpuser`

Furthermore, we assume that in the home directory of `otpuser`, there is a directory named `otp`, the full path of which is,

`/export/home/otpuser/otp`

This directory is the *installation directory* of the Embedded Environment.

Installation of Embedded Environment

The procedure for installation of an Embedded Environment does not differ from that of a Development Environment (see the chapter *Installation of Development Environment*), except for the following:

- the (compressed) tape archive file should be extracted in the installation directory as defined above, and,
- there is no need to link the start script to a standard directory like `/usr/local/bin`.

The details for extracting the tape archive file is not repeated here.

Configuration for Automatic Start at Boot

A true Embedded Environment has to start when the system boots. This section accounts for the necessary configurations needed to achieve that.

The embedded environment and all the applications will start automatically if the script file shown below is added to the `/etc/rc3.d` directory. The file must be owned and readable by `root`, and its name cannot be arbitrarily assigned. The following name is recommended,

`S75otp.system`

For further details on initialization (and termination) scripts, and naming thereof, see the file `/etc/init.d/README` on a Solaris 2.5.1 system.

```
#!/bin/sh
#
# File name:  S75otp.system
# Purpose:    Automatically starts Erlang and applications when the
#             system starts
# Author:     janne@erlang.ericsson.se
# Resides in: /etc/rc3.d
#

if [ ! -d /usr/bin ]
then
    exit          # /usr not mounted
fi

killproc() {
    # kill the named process(es)
    pid='/usr/bin/ps -e |
        /usr/bin/grep -w $1 |
        /usr/bin/sed -e 's/^ *//' -e 's/ .*//''
    [ "$pid" != "" ] && kill $pid
}

# Start/stop processes required for Erlang

case "$1" in
'start')
    # Start the Erlang emulator
    #
    su - otpuser -c "/export/home/otpuser/otp/bin/start" &
    ;;
'stop')
    killproc beam
    ;;
*)
    echo "Usage: $0 { start | stop }"
    ;;
esac
```

The file `/export/home/otpuser/otp/bin/start` referred to in the above script, is precisely the script `start` described in the section *Starting an Embedded System*. The script variable `OTP_ROOT` in that `start` script corresponds to the example path

```
/export/home/otpuser/otp
```

used in this section. The `start` script should be edited accordingly.

Use of the `killproc` procedure in the above script could be combined with a call to `erl_call`, e.g.

```
$SOME_PATH/erl_call -n Node init stop
```

In order to take Erlang down gracefully see the `erl_call(1)` reference manual page for further details on the use of `erl_call`. That however requires that Erlang runs as a distributed node which is not always the case.

The `killproc` procedure should not be removed: the purpose is here to move from run level 3 (multi-user mode with networking resources) to run level 2 (multi-user mode without such resources), in which Erlang should not run.

Hardware Watchdog

For Solaris 2.5.1 running on VME boards from Force Computers, there is a possibility to activate the onboard hardware watchdog, provided a VME bus driver is added to the operating system. For further details see the *Embedded Systems* documentation.

See also the `heart(3)` reference manual page in *Kernel*.

Changing permissions for reboot

If the `HEART_COMMAND` environment variable is to be set in the `start` script in the section, *Starting an Embedded System*, and if the value shall be set to the path of the Solaris `reboot` command, i.e.

```
HEART_COMMAND=/usr/sbin/reboot
```

the ownership and file permissions for `/usr/sbin/reboot` must be changed as follows,

```
chown 0 /usr/sbin/reboot
chmod 4755 /usr/sbin/reboot
```

See also the `heart(3)` reference manual page in *Kernel*.

The TERM environment variable

When the Erlang system is automatically started from the `S75otp.system` script the `TERM` environment variable has to be set. The following is a minimal setting,

```
TERM=sun
```

which should be added to the `start` script described in the section.

Patches for Solaris 2.5.1

For proper functioning of flushing file system data to disk, the Solaris 2.5.1 specific patch with number 103640-02 must be added to the operating system.

Installation of module `os_sup` in application `os_mon`

The following four installation procedures require super user privilege.

Installation

1. *Make a copy the Solaris standard configuration file for `syslogd`.*

- Make a copy the Solaris standard configuration file for `syslogd`. This file is usually named `syslog.conf` and found in the `/etc` directory.
- The file name of the copy must be `syslog.conf.Orig` but the directory location is optional. Usually it is `/etc`.

A simple way to do this is to issue the command

```
cp /etc/syslog.conf /etc/syslog.conf.Orig
```

2. *Make an Erlang specific configuration file for `syslogd`.*

- Make an edited copy of the back-up copy previously made.
- The file name must be `syslog.conf.OTP` and the path must be the same as the back-up copy.
- The format of the configuration file is found in the man page for `syslog.conf(5)`, by issuing the command `man syslog.conf`.
- Usually a line is added which should state:
 - which types of information that will be supervised by Erlang,
 - the name of the file (actually a named pipe) that should receive the information.
- If e.g. only information originating from the unix-kernel should be supervised, the line should begin with `kern.LEVEL` (for the possible values of `LEVEL` see `syslog.conf(5)`).
- After at least one tab-character, the line added should contain the full name of the named pipe where `syslogd` writes its information. The path must be the same as for the `syslog.conf.Orig` and `syslog.conf.OTP` files. The file name must be `syslog.otp`.
- If the directory for the `syslog.conf.Orig` and `syslog.conf.OTP` files is `/etc` the line in `syslog.conf.OTP` will look like:

```
kern.LEVEL          /etc/syslog.otp
```

3. *Check the file privileges of the configuration files.*

- The configuration files should have `rw-r--r--` file privileges and be owned by root.
- A simple way to do this is to issue the commands

```
chmod 644 /etc/syslog.conf
chmod 644 /etc/syslog.conf.Orig
chmod 644 /etc/syslog.conf.OTP
```

- *Note:* If the `syslog.conf.Orig` and `syslog.conf.OTP` files are not in the `/etc` directory, the file path in the second and third command must be modified.

4. *Modify file privileges and ownership of the `mod_syslog` utility.*

- The file privileges and ownership of the `mod_syslog` utility must be modified.
- The full name of the binary executable file is derived from the position of the `os_mon` application if the file system by adding `/priv/bin/mod_syslog`. The generic full name of the binary executable file is thus

`<OTP_ROOT>/lib/os_mon-<REV>/priv/bin/mod_syslog`

Example: If the path to the `otp-root` is `/usr/otp`, thus the path to the `os_mon` application is `/usr/otp/lib/os_mon-1.0` (assuming revision 1.0) and the full name of the binary executable file is `/usr/otp/lib/os_mon-1.0/priv/bin/mod_syslog`.

- The binary executable file must be owned by root, have `rwsr-xr-x` file privileges, in particular the `setuid` bit of user must be set.
- A simple way to do this is to issue the commands

```
cd <OTP_ROOT>/lib/os_mon-<REV>/priv/bin/mod_syslog
chmod 4755 mod_syslog
chown root mod_syslog
```

Testing the application configuration file The following procedure does not require root privilege.

- Ensure that the configuration parameters for the `os_sup` module in the `os_mon` application are correct.
- Browse the application configuration file (do *not* edit it). The full name of the application configuration file is derived from the position of the `os_mon`-application if the file system by adding `/ebin/os_mon.app`.

The generic full name of the file is thus

`<OTP_ROOT>/lib/os_mon-<REV>/ebin/os_mon.app`.

Example: If the path to the `otp-root` is `/usr/otp`, thus the path to the `os_mon` application is `/usr/otp/lib/os_mon-1.0` (assuming revision 1.0) and the full name of the binary executable file is `/usr/otp/lib/os_mon-1.0/ebin/os_mon.app`.

- Ensure that the following configuration parameters are bound to the correct values.

Parameter	Function	Standard value
<code>start_os_sup</code>	Specifies if <code>os_sup</code> will be started or not.	<code>true</code> for the first instance on the hardware; <code>false</code> for the other instances.
<code>os_sup_own</code>	The directory for (1)the back-up copy, (2) the Erlang specific configuration file for <code>syslogd</code> .	<code>"/etc"</code>
<code>os_sup_syslogconf</code>	The full name for the Solaris standard configuration file for <code>syslogd</code>	<code>"/etc/syslog.conf"</code>
<code>error_tag</code>	The tag for the messages that are sent to the error logger in the Erlang system.	<code>std_error</code>

Table 1.1: Configuration Parameters

If the values listed in the `os_mon.app` do not suit your needs, you should *not* edit that file. Instead you should *override* values in a *system configuration file*, the full pathname of which is given on the command line to `erl`.

Example: The following is an example of the contents of an application configuration file.

```
[{os_mon, [{start_os_sup, true}, {os_sup_own, "/etc"},  
{os_sup_syslogconf, "/etc/syslog.conf"}, {os_sup_errortag, std_error}]]].
```

Related documents See also the `os_mon(3)`, `application(3)` and `erl(1)` reference manual pages.

Installation Problems

The hardware watchdog timer which is controlled by the heart port program requires the `FORCEvme` package, which contains the VME bus driver, to be installed. This driver, however, may clash with the Sun `mcp` driver and cause the system to completely refuse to boot. To cure this problem, the following lines should be added to `/etc/system`:

- `exclude: drv/mcp`
- `exclude: drv/mcpzsa`
- `exclude: drv/mcpp`

Warning:

It is recommended that these lines be added to avoid the clash described, which may make it completely impossible to boot the system.

Starting Erlang

This section describes how an embedded system is started. There are four programs involved, and they all normally reside in the directory `<ERL_INSTALL_DIR>/bin`. The only exception is the program `start`, which may be located anywhere, and also is the only program that must be modified by the user.

In an embedded system there usually is no interactive shell. However, it is possible for an operator to attach to the Erlang system by giving the command `to_erl`. He is then connected to the Erlang shell, and may give ordinary Erlang commands. All interaction with the system through this shell is logged in a special directory.

Basically, the procedure is as follows. The program `start` is called when the machine is started. It calls `run_erl`, which sets things up so the operator can attach to the system. It calls `start_erl` which calls the correct version of `erlexec` (which is located in `<ERL_INSTALL_DIR>/erts-EVsn/bin`) with the correct boot and config files.

Programs

start

This program is called when the machine is started. It may be modified or re-written to suit a special system. By default, it must be called `start` and reside in `<ERL_INSTALL_DIR>/bin`. Another start program can be used, by using the configuration parameter `start_prg` in the application `sasl`.

The start program must call `run_erl` as shown below. It must also take an optional parameter which defaults to `<ERL_INSTALL_DIR>/bin/start_erl.data`.

This program should set static parameters and environment variables such as `-sname Name` and `HEART_COMMAND` to reboot the machine.

The `<RELDIR>` directory is where new release packets are installed, and where the release handler keeps information about releases. See `release_handler(3)` in the application `sasl` for further information.

The following script illustrates the default behaviour of the program.

```
#!/bin/sh
# Usage: start [DataFile]
#
ROOTDIR=/usr/local/otp

if [ -z "$RELDIR" ]
then
    RELDIR=$ROOTDIR/releases
fi

START_ERL_DATA=${1:-$RELDIR/start_erl.data}

$ROOTDIR/bin/run_erl /tmp/ $ROOTDIR/log "exec $ROOTDIR/bin/start_erl @@@
    $ROOTDIR $RELDIR $START_ERL_DATA" > /dev/null 2>&1 &
```

The following script illustrates a modification where the node is given the name `cp1`, and the environment variables `HEART_COMMAND` and `TERM` have been added to the above script.

```
#!/bin/sh
# Usage: start [DataFile]
#
HEART_COMMAND=/usr/sbin/reboot
TERM=sun
export HEART_COMMAND TERM

ROOTDIR=/usr/local/otp

if [ -z "$RELDIR" ]
then
    RELDIR=$ROOTDIR/releases
fi

START_ERL_DATA=${1:-$RELDIR/start_erl.data}

$ROOTDIR/bin/run_erl /tmp/ $ROOTDIR/log "exec $ROOTDIR/bin/start_erl @@@
    $ROOTDIR $RELDIR $START_ERL_DATA -heart -sname cp1" > /dev/null 2>&1 &
```

If a diskless and/or read-only client node is about to start the `start_erl.data` file is located in the client directory at the master node. Thus, the `START_ERL_DATA` line should look like:

```
CLIENTDIR=$ROOTDIR/clients/clientname
START_ERL_DATA=${1:-$CLIENTDIR/bin/start_erl.data}
```

run_erl

This program is used to start the emulator, but you will not be connected to the shell. `to_erl` is used to connect to the Erlang shell.

Usage: `run_erl pipe_dir/ log_dir "exec command [parameters ...]"`

Where `pipe_dir/` should be `/tmp/` (`to_erl` uses this name by default) and `log_dir` is where the log files are written. `command [parameters]` is executed, and everything written to `stdin` and `stdout` is logged in the `log_dir`.

In the `log_dir`, log files are written. Each logfile has a name of the form: `erlang.log.N` where `N` is a generation number, ranging from 1 to 5. Each logfile holds up to 100kB text. As time goes by the following logfiles will be found in the logfile directory

```
erlang.log.1
erlang.log.1, erlang.log.2
erlang.log.1, erlang.log.2, erlang.log.3
erlang.log.1, erlang.log.2, erlang.log.3, erlang.log.4
erlang.log.2, erlang.log.3, erlang.log.4, erlang.log.5
erlang.log.3, erlang.log.4, erlang.log.5, erlang.log.1
...
```

with the most recent logfile being the right most in each row of the above list. That is, the most recent file is the one with the highest number, or if there are already four files, the one before the skip.

When a logfile is opened (for appending or created) a time stamp is written to the file. If nothing has been written to the log files for 15 minutes, a record is inserted that says that we're still alive.

to_erl

This program is used to attach to a running Erlang system, started with `run_erl`.

Usage: `to_erl [pipe_name | pipe_dir]`

Where `pipe_name` defaults to `/tmp/erlang.pipe.N`.

To disconnect from the shell without exiting the Erlang system, type `Ctrl-D`.

start_erl

This program starts the Erlang emulator with parameters `-boot` and `-config` set. It reads data about where these files are located from a file called `start_erl.data` which is located in the `<RELDIR>`. Each new release introduces a new data file. This file is automatically generated by the release handler in Erlang.

The following script illustrates the behaviour of the program.

```
#!/bin/sh
#
# This program is called by run_erl. It starts
# the Erlang emulator and sets -boot and -config parameters.
# It should only be used at an embedded target system.
#
# Usage: start_erl RootDir RelDir DataFile [ErlFlags ...]
#
ROOTDIR=$1
shift
RELDIR=$1
shift
DataFile=$1
shift

ERTS_VSN='awk '{print $1}' $DataFile'
VSN='awk '{print $2}' $DataFile'

BINDIR=$ROOTDIR/erts-$ERTS_VSN/bin
EMU=beam
PROGNAME='echo $0 | sed 's/.*\\///''
export EMU
export ROOTDIR
export BINDIR
export PROGNAME
export RELDIR

exec $BINDIR/erlexec -boot $RELDIR/$VSN/start -config $RELDIR/$VSN/sys $*
```

If a diskless and/or read-only client node with the `sasl` configuration parameter `static_emulator` set to `true` is about to start the `-boot` and `-config` flags must be changed. As such a client cannot read a new `start_erl.data` file (the file is not possible to change dynamically) the boot and config files are always fetched from the same place (but with new contents if a new release has been installed). The `release_handler` copies this files to the `bin` directory in the client directory at the master nodes whenever a new release is made permanent.

Assuming the same `CLIENTDIR` as above the last line should look like:

```
exec $BINDIR/erlexec -boot $CLIENTDIR/bin/start @@@
    -config $CLIENTDIR/bin/sys $*
```


Other Issues

Future releases of OTP will include more information on how Solaris can be configured for use with embedded systems to get maximum performance. Issues which will be investigated include:

- how disabling swapping affects the system
- how locking processes in memory may yield performance benefits.

1.2 Windows NT

This chapter describes the OS specific parts of OTP which relate to Windows NT.

Introduction

We have at present no information from Microsoft on how to strip down a Windows NT installation to the minimum required for use as an embedded system.

Memory Usage

At the time of writing, we have no reliable information to put here. A system with less than 32 Mbytes of RAM is however not recommended.

Disk Space Usage

At the time of writing, we have no reliable information to put here.

Installation

This section is about how to configure a Windows NT system as an embedded system suitable for running OTP.

No documentation is currently available.

Hardware Watchdog

For Windows NT running on standard PCs with ISA and/or PCI bus there is a possibility to install an extension card with a hardware watchdog.

See also the `heart(3)` reference manual page in *Kernel*.

Starting Erlang

No documentation available yet.

1.3 VxWorks

This chapter describes the OS specific parts of OTP which relate to VxWorks.

Introduction

The Erlang/OTP distribution for VxWorks is limited to what Switchboard requires (Switchboard is a general purpose switching hardware developed by Ericsson). If you want us to include more parts of OTP in the distribution, contact us to discuss this. Please consult the README file, included at root level in the installation, for latest information on the distribution.

Memory usage

Memory usage is not known at the time of writing.

Disk usage

Disk usage is not known at the time of writing.

Installation

OTP/VxWorks is supplied in a distribution file named `<PREFIX>.tar.gz`; i.e. a tar archive that is compressed with gzip. `<PREFIX>` represents the name of the release, e.g. `otp_LXA12345_vxworks_cpu32_R3A`. Assuming you are installing to a Solaris file system, the installation is performed by following these steps: <

- Change to the directory where you want to install OTP/VxWorks (`<ROOTDIR>`): `cd <ROOTDIR>`
- Make a directory to put OTP/VxWorks in: `mkdir otp_vxworks_cpu32` (or whatever you want to call it)
- Change directory to the newly created one: `cd otp_vxworks_cpu32`
- Copy the distribution file there from where it is located (`<RELDIR>`): `cp <RELDIR>/<PREFIX>.tar.gz .`
- Unzip the distribution file: `gunzip <PREFIX>.tar.gz`
- Untar `<PREFIX>.tar`: `tar xvf <PREFIX>.tar`
- Create a bin directory: `mkdir bin`
- Copy the VxWorks Erlang/OTP start-up script to the bin directory: `cp erts-4.6/bin/erl bin/.`
- Copy the example start scripts to the bin directory: `cp releases/R3A/*.boot bin/.`

If you use VxWorks nfs mounting facility to mount the Solaris file system, this installation may be directly used. An other possibility is to copy the installation to a local VxWorks DOS file system, from where it is used.

OS specific functionality/information

There are a couple of files that are unique to the VxWorks distribution of Erlang/OTP, these files are described here.

- README - this files has some information on VxWorks specifics that you are advised to consult. This includes the latest information on what parts of OTP are included in the VxWorks distribution of Erlang/OTP. If you want us to include more parts, please contact us to discuss this.
- erts-4.6/bin/resolv.conf - A resolver configuration EXAMPLE file. You have to edit this file.
- erts-4.6/bin/erl - This is an EXAMPLE start script for VxWorks. You have to edit this file to suit your needs.
- erts-4.6/bin/erl_io - One possible solution to the problem of competing Erlang and VxWorks shell. Contains the function 'start_erl' called by the erl script. Also contains the function 'to_erl' to be used when connecting to the Erlang shell from VxWorks' shell.
- erts-4.6/bin/erl_exec - Rearranges command line arguments and starts Erlang.
- erts-4.6/bin/vxcall - Allows spawning of standard VxWorks shell functions (which is just about any function in the system...) from open_port/2. E.g. open_port({spawn, 'vxcall func arg1 arg2'}, []) will cause the output that 'func arg1, arg2' would have given in the shell to be received from the port.
- erts-4.6/bin/rdate - Set the time from a networked host, like the SunOS command. Nothing Erlang-specific, but nice if you want date/0 and time/0 to give meaningful values (you also need a TIMEZONE environment setting if GMT isn't acceptable). For example: putenv "TIMEZONE=CET: : -60:033002:102603" sets central european time.
- erts-4.6/src - Contains source for the above files, and additionally config.c, driver.h, preload.c and reclaim.h. Reclaim.h defines the interface to a simple mechanism for "resource reclamation" that is part of the Erlang system - may be useful to "port program" writers (and possibly others). Take careful note of the caveats listed in the file!

Starting Erlang

Start (and restart) of the system depends on what file system is used. To be able to start the system from a nfs mounted file system you can use VxWorks start script facility to run a start script similar to the example below. Note that the Erlang/OTP start-up script is run at the end of this script.

```
# start.script v1.0 1997/09/08 patrik
#
# File name:  start.script
# Purpose:    Starting the VxWorks/cpu32 erlang/OTP
# Author:     patrik@erix.ericsson.se
# Resides in: ~tornado/wind/target/config/ads360/
#
```

```

# Set shell prompt
#
shellPromptSet("sauron-> ")

#
# Set default gateway
#
hostAdd "router-20","150.236.20.251"
routeAdd "0","router-20"

#
# Mount /home from gandalf
#
hostAdd "gandalf","150.236.20.16"
usergroup=10
nfsAuthUnixSet("gandalf", 452, 10, 1, &usergroup)
nfsMount("gandalf", "/export/home", "/home")

#
# Load and run rdate.o to set correct date on the target
#
ld < /home/gandalf/tornado/wind/target/config/ads360/rdate.o
rdate("gandalf")

#
# Setup timezone information (Central European time)
#
putenv "TIMEZONE=CET::-60:033002:102603"

#
# Run the Erlang/OTP start script
#
cd "/home/gandalf/tornado/wind/target/erlang_cpu32_P3A/bin"
<erl

```

Card specific functionality/information

The only card Erlang/OTP R3A for VxWorks is tested on is Switchboard/mc68360. Other mc68360 based cards with VxWorks BSP support should work with none, or minimal effort.

List of Tables

Chapter 1: Embedded Systems User's Guide

1.1 Configuration Parameters 7