

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

TS-7800

Contents

1 Overview
2 Getting Started
- 2.1 Get a Console
- 2.2 Initrd / Busybox / Fastboot
- 2.3 Boot Process
- 2.4 Jumper Configuration

3 Backup / Restore
- 3.1 SD Card
- 3.2 Onboard Flash

4 System Configuration
- 4.1 Configuring the Network
  - 4.1.1 Network Troubleshooting
- 4.2 Installing New Software
- 4.3 Setting up SSH
- 4.4 Starting Automatically

5 Software Development
- 5.1 Linux Kernel
  - 5.1.1 ATAGS
- 5.2 Compiling the Kernel

6 Features
- 6.1 CPU Functionality
  - 6.1.1 MMU
  - 6.1.2 Interrupts
  - 6.1.3 Onboard Flash
  - 6.1.4 USB Host
  - 6.1.5 SATA
  - 6.1.6 DDR RAM
  - 6.1.7 Onboard NAND Flash
- 6.2 FPGA Functionality
  - 6.2.1 FPGA Changelog
  - 6.2.2 SD
  - 6.2.3 Syscon
  - 6.2.4 TSUART
  - 6.2.5 PC104
    - 6.2.5.1 PC104 GPIO
    - 6.2.5.2 PC104 16550
  - 6.2.6 FPGA DMA
  - 6.2.7 FPGA IRQ Controller
- 6.3 AVR Functionality
  - 6.3.1 Sleep Mode
  - 6.3.2 ADC Sampling
- 6.4 CPLD Functionality
  - 6.4.1 LCD and DIO
- 6.5 Serial Ports
- 6.6 Realtime Clock
- 6.7 Temperature Sensor
- 6.8 Status LEDs
- 6.9 Jumpers

7 External Interfaces
- 7.1 DIO Header
- 7.2 LCD Header
- 7.3 Ethernet Port
- 7.4 SD Connectors
- 7.5 Power Supply Connector
- 7.6 DB9 Connector

TS-7800



Product Page

Product Images

Specifications

Documentation

Schematic

Mechanical Drawing

FTP Path

Processor

Marvell MV88F5182

500MHz Arm®v4TE Arm9™TDMI CPU

7.7 COM2 Header
7.8 COM3 Header
7.9 A/D Header
7.10 JTAG Header
7.11 PC104 Header
8 External Links
9 Product Notes
9.1 Errata
9.2 FCC Advisory
9.3 Limited Warranty
9.4 Trademarks

Overview

The TS-7800 is a RoHS compliant Single Board Computer (SBC) based on a Marvell MV88F5182 500MHz ARM9 CPU. An internal 32-bit PCI bus provides a standard set of high-end on-board peripherals such as 10/100/1000 ethernet, dual SATA and dual High-Speed host USB 2.0. The TS-7800 features 128MB of DDR-RAM and 512MB of high-speed NAND flash.


The TS-7800 running a LAMP stack hosting www.embeddedTS.com

Getting Started

A Linux PC is recommended for development. For developers who use Windows, virtualized Linux using VMWare or similar are recommended in order to make the full power of Linux available. The developer will need to be comfortable with Linux anyway in order to work with embedded Linux on the target platform. The main reasons that Linux is useful are:

- Linux filesystems on the microSD card can be accessed on the PC.
- More ARM cross-compilers are available.
- If recovery is needed, a bootable medium can be written.
- A network filesystem can be served.
- Builds such as Linux kernel, buildroot, yocto, distro-seed will not work from WSL1/2 on a case insensitive filesystem.

WARNING: Be sure to take appropriate Electrostatic Discharge (ESD) precautions. Disconnect the power source before moving, cabling, or performing any set up procedures. Inappropriate handling may cause damage to the board.

The TS-7800 receives power through the +5VDC power connector with a 1A minimum supply. If you have OP-SWITCHREG there will be a socket connector that will accept 8-30VDC.

Get a Console

The boot messages, by default, are all displayed on COM1 at 115200 baud. The TS-7800 board has Linux installed by default on onboard flash. Upon bootup, The board will boot within 1.1 seconds to a Linux prompt on UART #0 (/dev/ttyS0).

Use a null modem cable to connect the ARM system to your workstation. If you do not have a COM port on your system (as many newer systems do not), you can find a USB serial adapter that will bring out RS232.

Console from Linux

There are many serial clients for Linux, but 3 simple ones would be picocom, screen, and minicom. These examples assume that your COM device is /dev/ttyUSB0 (common for USB adapters), but replace them with the COM device on your workstation.

Linux has a few applications capable of connecting to the board over serial. You can use any of these clients that may be installed or available in your workstation's package manager:

Picocom is a very small and simple client.

```
picocom -b 115200 /dev/ttyUSB0
```

Screen is a terminal multiplexer which happens to have serial support.

```
screen /dev/ttyUSB0 115200
```

Or a very commonly used client is minicom which is quite powerful:

```
minicom -s
```

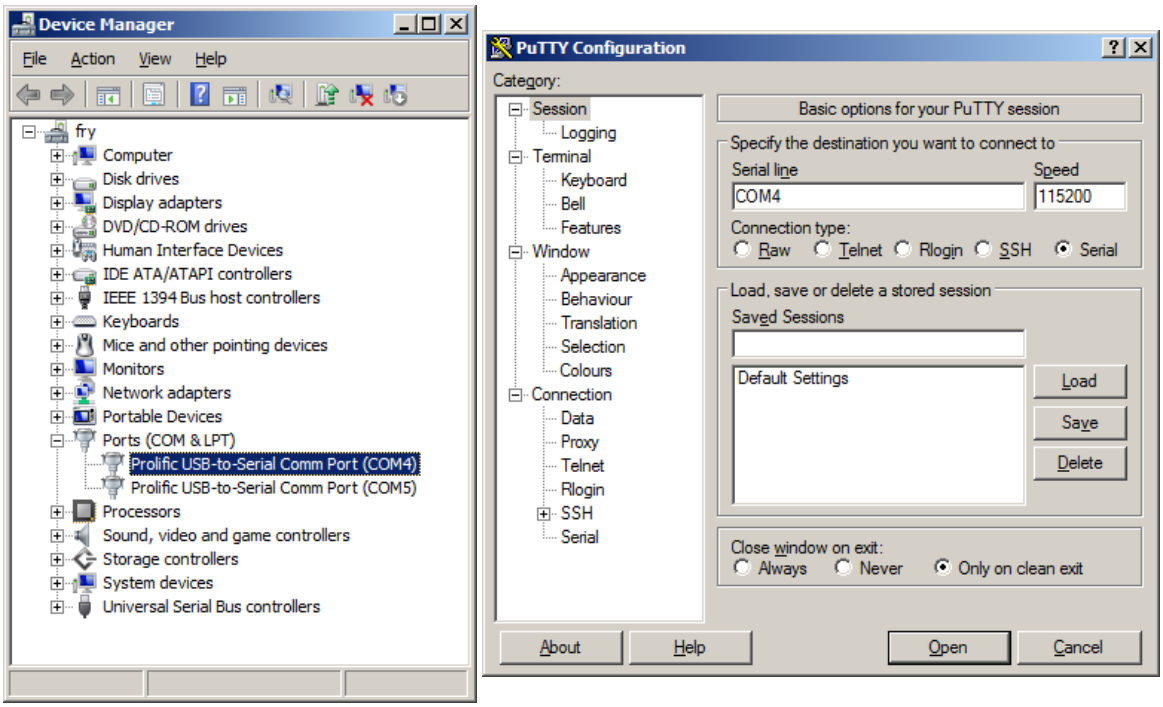
- Navigate to 'serial port setup'
- Type "a" and change location of serial device to '/dev/ttyUSB0' then hit "enter"
- If needed, modify the settings to match this and hit "esc" when done:

```
    E - Bps/Par/Bits          : 115200 8N1
    F - Hardware Flow Control : No
    G - Software Flow Control : No
```

- Navigate to 'Save setup as dfl', hit "enter", and then "esc"

Console from Windows

Putty is a small simple client available for download here. Open up Device Manager to determine your console port. See the putty configuration image for more details.



Initrd / Busybox / Fastboot

After the TS-7800 is first booted you will be at this shell:

```
>> TS-BOOTROM - built May 11 2011
>> Copyright (c) 2008, Technologic Systems
>> Booting from onboard NAND flash...
.
.
.
.
Finished booting in 0.78 seconds
Type 'tshelp' for help
/ #
```

This is a busybox shell which presents you with a very minimalistic system. This filesystem is loaded into memory, so none of the changes will be saved unless you type 'save', or mount a filesystem as read write. This can also provide a simple mechanism for running your application in an entirely read only environment. The linuxrc script will be the first thing executed as soon as the kernel is loaded. This sets the default IP address, starts the userspace ctl applications, and more. Read the linuxrc for more information.

While busybox itself doesn't contain much functionality, it does mount the Debian partition under /mnt/root/. It will also add common paths and load libraries from the Debian system. Many of the Debian applications will work by default. If an application relies on certain paths being in certain places, or running services, you should instead boot to Debian to run them.

This shell when started on the COM port is what is blocking a Debian boot. If you close it by typing 'exit', the boot process will continue. If you are connected through telnet, this will instead open up its own instance of the shell so typing 'exit' will only end that session. You would have to relink the linuxrc script to always boot to a specific partition from telnet. When logging into Debian, the default user is 'root' with no password. Technologic Systems strongly recommends setting a secure password for your system root account.

The compiled instance of busybox includes internal commands listed below:

```
/ # /bin/busybox --help
BusyBox v1.7.1 (2008-07-16 14:59:19 MST) multi-call binary
Copyright (C) 1998-2006  Erik Andersen, Rob Landley, and others.
Licensed under GPLv2.  See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: [function] [arguments]...

    BusyBox is a multi-call binary that combines many common Unix
    utilities into a single executable.  Most people will create a
    link to busybox for each function they wish to use and BusyBox
    will act like whatever it was invoked as!

Currently defined functions:
    [, [[, ash, basename, cal, catv, chgrp, chmod, chown,
    chroot, cksum, clear, comm, cp, cttyhack, cut, date, dd,
    df, dirname, dmesg, du, echo, egrep, env, eptime, expr,
    false, fdisk, fgrep, find, free, ftpget, ftpput, getopt,
    grep, gunzip, halt, head, hexdump, hostname, ifconfig,
    insmod, kill, killall, ln, ls, lsmod, md5sum, mdev, mkdir,
    mkfifo, mknod, modprobe, more, mount, mtdcp, mv, mvtime,
    netstat, nice, nohup, nslookup, peekpoke, ping, ping6,
    pivot_root, poweroff, printenv, printf, ps, pscan, pwd,
    reboot, rm, rmdir, rmmod, route, rx, sed, setconsole,
    setlogcons, sh, sha1sum, sleep, split, stat, stty, sum,
    sync, tail, tar, tee, telnetd, test, tftp, time, top,
    tr, traceroute, true, tty, udhcpc, umount, uname, unzip,
    uptime, usleep, uudecode, uuencode, vi, wget, which, xargs,
    zcat
```

Note: You may want to refer to the sources for some of these commands if they are not behaving as you would expect. Busybox implements them so they are incredibly small, but they don't always provide all of the functionality as is in the standard gnu/bash shell commands. Some commands will provide the same functionality like date, but may only work with very specific formats.

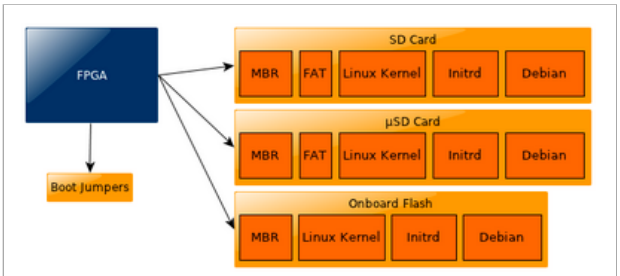
We also provide the ts7800.subr which provides the following functions:

```
bit_set()
bit_clr()
bit_get()
led0()
pc104on()
pc104off()
tshelp()
sdmount()
mtdmount()
help()
```

Boot Process

The MBR, kernel, and initrd can be loaded from MicroSD, SD, or the onboard flash depending on your jumper configuration. Once this is loaded you can boot to any other storage (SATA drive, USB, NFS, etc). JP1 will allow you to pick between booting between the SD card sockets and the onboard flash, but if both SD and MicroSD are populated preference will be given to MicroSD.

Once you are booted to the initrd, you can change the linuxrc script to boot into the full linux installation on any device. We provide several scripts that you can relink to linuxrc:



Boot Process

Script	Function
linuxrc-fastboot (default)	Boots to a shell. Once the shell is closed (type exit) it will boot to Debian on the full sized SD card.
linuxrc-mtdroot	Boots directly to Debian on the onboard flash. Uses a unionfs to mount the onboard flash in read only mode and commits writes to a tmpfs.
linuxrc-mtdroot	Boots directly to Debian on the onboard flash.
linuxrc-nfsroot	Attempts to NFS root to 192.168.0.1:/tsarm-nfsroot. You can edit this script to use your own server host and path.
linuxrc-sdroot	Boots directly to Debian on the full sized SD card.
linuxrc-sdroot-readonly	Boots directly to Debian on the full sized SD card. Uses a unionfs to mount the SD card in read only mode and commits writes to a tmpfs.

To relink to automatically boot to sd, you would use this command

```
rm linuxrc && ln -s linuxrc-sdroot linuxrc && save
```

To change the boot device to MicroSD card in these scripts, you can run a sed script on the linuxrc-fastboot or linuxrc-sdroot scripts:

```
sed --in-place 's/tssdcardb4/tssdcarda4/g' /linuxrc-sdroot
```

If you want to boot to USB or SATA you may want to modify the linuxrc-sdroot or linuxrc-fastboot to mount your storage device at /mnt/root/.

Jumper Configuration

Jumper	Function
JP1	Boot to SD (Otherwise nand)
JP2	User Configurable
JP3	Lowers CPU clock

Note: See the FPGA Register Map for reading the states of the jumpers in software

Note: While JP3 does lower the clock speed, it does not lower power consumption

Backup / Restore

SD Card

If backing up on a separate workstation, keep in mind windows does not have direct block device support needed to write these images. You will also need to determine the SD card device. You can usually find this in the output of 'dmesg' after inserting the SD card and you will typically see something like '/dev/sdb' as the block device and '/dev/sdb1' for the first partition. On some newer kernels you will see '/dev/mmcblk0' as the block device and '/dev/mmcblkop1' for the first partition. For these examples I will use the '/dev/mmcblk0' format. Both of the sd cards will use the same commands, except when backing up or restoring on the TS-7800 you would replace /dev/tssdcarda (MicroSD) or /dev/tssdcardb (SD) depending on which device you want to use. These examples will use /dev/tssdcarda.

You can find the latest image here

If you are backing up directly on the board you will likely need to use some kind of offboard storage like a thumbdrive or external hard drive.

From Workstation

Backup

Entire SD card

```
dd if=/dev/mmcblk0 of=/path/to/backup.dd bs=4M conv=fsync
```

Kernel

```
dd if=/dev/mmcblk0p2 of=/path/to/zImage bs=4M conv=fsync
```

Initrd

```
dd if=/dev/mmcblk0p3 of=/path/to/initrd bs=4M conv=fsync
```

Restore

Entire SD card

```
dd if=/path/to/backup.dd of=/dev/mmcblk0 bs=4M conv=fsync
```

Kernel

```
dd if=/path/to/zImage of=/dev/mmcblk0p2 bs=4M conv=fsync
```

Initrd

```
dd if=/path/to/initrd  of=/dev/mmcblk0p3 bs=4M conv=fsync
```

From TS-7800

Backup

Entire card

```
dd if=/dev/tssdcarda of=/path/to/backup.dd bs=4M conv=fsync
```

Kernel

```
dd if=/dev/tssdcarda2 of=/path/to/zImage bs=4M conv=fsync
```

Initrd

```
dd if=/dev/tssdcarda3 of=/path/to/initrd bs=4M conv=fsync
```

Restore

The entire card from SBC

```
dd if=/path/to/sdimage.dd of=/dev/tssdcarda bs=4M conv=fsync
```

Kernel

```
dd if=/path/to/zImage of=/dev/tssdcarda2 bs=4M conv=fsync
```

Initrd

```
dd if=/path/to/zImage of=/dev/tssdcarda3 bs=4M conv=fsync
```

Onboard Flash

To restore the onboard flash, you must first boot to a 512MB SD image. You can write that image to the onboard flash with this command:

```
createmtdboot && createmtdroot
```

This can take approximately 5 minutes.

WARNING: These scripts are only set up to copy the MBR that works with the 2.6.21 kernel image.

System Configuration

For development it is recommended to go boot to the full Debian where there is plenty of space for development work. Debian provides many more packages and a much more familiar environment for users already versed in Debian. Once here you can use apt-get to install/remove packages, configure the network, and perform other common tasks.

Configuring the Network

From almost any Linux system you can use "ip" or the ifconfig/route commands to initially set up the network. To configure the network interface manually you can use the same set of commands in the initrd or Debian.

```
# Bring up the CPU network interface
ifconfig eth0 up

# Or if you're on a baseboard with a second ethernet port, you can use that as:
ifconfig eth1 up

# Set an ip address (assumes 255.255.255.0 subnet mask)
ifconfig eth0 192.168.0.50

# Set a specific subnet
ifconfig eth0 192.168.0.50 netmask 255.255.0.0

# Configure your route.  This is the server that provides your internet connection.
route add default gw 192.168.0.1

# Edit /etc/resolv.conf for your DNS server
echo "nameserver 192.168.0.1" > /etc/resolv.conf
```

Most commonly networks will offer DHCP which can be set up with one command:

Configure DHCP in Debian:

```
# To setup the default CPU ethernet port
pump -i eth0
# Or if you're on a baseboard with a second ethernet port, you can use that as:
pump -i eth1
```

Configure DHCP in the initrd:

```
udhcpc -i eth0
# Or if you're on a baseboard with a second ethernet port, you can use that as:
udhcpc -i eth1
```

To make your network settings take effect on startup in Debian, edit /etc/network/interfaces:

```
# Used by ifup(8) and ifdown(8). See the interfaces(5) manpage or
# /usr/share/doc/ifupdown/examples for more information.

# We always want the loopback interface.
#
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
  address 192.168.0.50
  netmask 255.255.255.0
  gateway 192.168.0.1
auto eth1
iface eth1 inet dhcp
```

In this example eth0 is a static configuration and eth1 receives its configuration from the DHCP server. For more information on network configuration in Debian see their documentation here.

To make your changes permanent in the initrd you will need to edit the linuxrc script. Use the same commands you would use to manually configure it and place them over the current ifconfig calls.

Network Troubleshooting

One common error you might encounter is after changing to use linuxrc-sdroot, you might notice the Ethernet network interface fails to come up. The error will present itself during Debian's boot with several errors beginning in SIOCSIF. This is because udev is trying to assign eth0 to a MAC address that isn't present on your SBC. The fix is simple enough, edit or (re)move the configuration file telling udev to do something silly. Try this from Debian:

```
mv /etc/udev/rules.d/70-persistent-net.rules /etc/udev/rules.d/70-persistent-net.rules.old
```

Run sync and reboot. Now your preferences in /etc/network/interfaces will take effect.

Installing New Software

Debian provides the apt-get system which lets you manage prebuilt applications. Before you do this you need to update Debian's list of package versions and locations. This assumes you have a valid network connection to the internet.

Debian Etch has been moved to archive so you will need to update /etc/apt/sources.list to contain these two lines:

```
deb http://archive.debian.org/debian etch main
deb-src http://archive.debian.org/debian etch main
```

Now you can update the local cache of packages:

```
apt-get update
```

For example, if you wanted to install picocom you could use the apt-cache command to search the local cache of Debian's packages.

```
root@ts7000:~# apt-cache search picocom
picocom - minimal dumb-terminal emulation program
```

You can often find the names of packages from Debian's wiki or from just searching on google as well.

Once you have the package name you can use apt-get to install the package and any dependencies. This assumes you have a network connection to the internet.

```
apt-get install picocom
# You can also chain packages to be installed
apt-get install picocom nano vim
```

For more information on using apt-get refer to Debian's documentation here.

Setting up SSH

On our boards we include the Debian package for openssh-server, but we remove the automatically generated keys for security reasons. To regenerate these keys:

```
dpkg-reconfigure openssh-server
```

Make sure your board is configured properly on the network, and set a password for your remote user. SSH will not allow remote connections without a password or a shared key.

```
passwd root
```

You should now be able to connect from a remote Linux or OSX system using "ssh" or from Windows using a client such as putty.

Starting Automatically

From Debian the most straightforward way to add your application to startup is to create a startup script. This is an example simple startup script that will toggle the red led on during startup, and off during shutdown. In this case I'll name the file customstartup, but you can replace this with your application name as well.

Edit the file /etc/init.d/customstartup to contain this:

```
#! /bin/sh
# /etc/init.d/customstartup

case "$1" in
  start)
    /usr/local/bin/ts7800ctl --redledon
    ## If you are launching a daemon or other long running processes
    ## this should be started with
    # nohup /usr/local/bin/yourdaemon &
    ;;
  stop)
    /usr/local/bin/ts7800ctl --redledoff
    ;;
  *)
    echo "Usage: customstartup start|stop" >&2
    exit 3
    ;;
esac

exit 0
```

Note: The $PATH variable is not set up by default in init scripts so this will either need to be done manually or the full path to your application must be included.

To make this run during startup and shutdown:

```
update-rc.d customstartup defaults
```

To manually start and stop the script:

```
/etc/init.d/customstartup start
/etc/init.d/customstartup stop
```

To make your application startup from the initrd you only need to add the required lines (no need for the Debian init syntax) to the linuxrc script. Usually the best place to add in your application is right after /mnt/root/ is mounted so the Debian libraries and applications are available.

Software Development

Editors

Vim is a very common editor to use in Linux. While it isn't the most intuitive at a first glance, you can run 'vimtutor' to get a ~30 minute instruction on how to use this editor. Once you get past the initial learning curve it can make you very productive. You can find the vim documentation here.

Emacs is another very common editor. Similar to vim, it is difficult to learn but rewarding in productivity. You can find documentation on emacs here.

Nano while not as commonly used for development is the easiest. It doesn't have as many features to assist in code development, but is much simpler to begin using right away. If you've used 'edit' on Windows/DOS, this will be very familiar. You can find nano documentation here.

Compilers

We only recommend the gnu compiler collection. There are many other commercial compilers which can also be used, but will not be supported by us. You can install gcc on most boards in Debian by simply running 'apt-get update && apt-get install build-essential'. This will include everything needed for standard development in c/c++.

You can find the gcc documentation here. You can find a simple hello world tutorial for c++ with gcc here.

Build tools

When developing your application typing out the compiler commands with all of your arguments would take forever. The most common way to handle these build systems is using a make file. This lets you define your project sources, libraries, linking, and desired targets. You can read more about makefiles here.

If you are building an application intended to be more portable than on this one system, you can also look into the automake tools which are intended to help make that easier. You can find an introduction to the autotools here.

Cmake is another alternative which generates a makefile. This is generally simpler than using automake, but is not as mature as the automake tools. You can find a tutorial here.

Debuggers

Linux has a few tools which are very helpful for debugging code. The first of which is gdb (part of the gnu compiler collection). This lets you run your code with breakpoints, get backgraces, step forward or backward, and pick apart memory while your application executes. You can find documentation on gdb here.

Strace will allow you to watch how your application interacts with the running kernel which can be useful for diagnostics. You can find the manual page here.

Ltrace will do the same thing with any generic library. You can find the manual page here.

Linux Kernel

Marvell has strongly promoted running Linux on this chip and has done most of the legwork in creating a patch set to the Linux 2.6 kernels, but we have also had to modify the Linux Kernel (TS-Kernel) so it can support the on-board Flash chip (via mtd drivers) and the SD card driver. The TS-7800 SBC's are shipped standard with full-featured Debian linux operating system utilizing the Linux-boot-Linux boot loader installed in the on-board Flash memory. The supported TS-Kernel used is based upon the version 2.6.21, patched and compiled for the MV 88F5182 processor.

We also provide a 2.6.34 kernel as is. This will work on the SD card, but you will not be able to restore this SD image to the onboard nand without the correct MBR with the appropriate ATAGS. The vanilla Linux Kernel also has support for the TS-7800.

When you compile source code there are always lots of options for toolchains (soft float vs hard float...) but the two major categories customers really need to know about are version of gcc and version of glibc. The version of gcc is important when compiling the kernel because the kernel is literally millions of lines of code, so a lot of corner cases are encountered in the kernel. As gcc releases new versions, they sometimes change the way they handle corner cases. For example, in one version they may call a certain corner case a warning and let the corner case go by, but in another version they might call it an error and stop compiling. So you need to make sure you have the version of gcc that is compatible with your particular kernel or you may get an error where another version would have given you a warning. For the 7800 use "arm-none-linux-gnueabi-gcc" as the toolchain. The version of glibc is important for people compiling user space applications. libc is the C library where all the standard C functions are implemented.

In order to make things more efficient from a code size standpoint, Technologic Systems put one copy of the libc libraries in user space and let everyone share this one copy as opposed to giving everyone their own copy of the libc library. This is called dynamic linking. Static linking is where each binary has it's own copy of all the libraries it needs. When you compile a program you have to link against a version of the C libraries so the compiler can verify functions like printf() really do exist and you are passing the right arguments and lots of other details the compiler needs to know.

If you link against a version of glibc when you compile it is a really good idea to ensure you have that same version of glibc running on the board. In theory glibc is backwards compatible so if you link against an older version you should be able to run using a newer version, but this isn't always the case. In embedded systems where stability is one of your major concerns taking risks is not a good idea. For really large projects like php or apache it's usually a good idea to take a look at what version of gcc is recommended also, although for the most part this is not an issue for user space programs because the probability of them having code for one of the corner cases gcc may have changed is much lower.

ATAGS

During the initial release of the board we used an internal ATAG identifier that the linux kernel would use to recognize the kernel. When we created the 2.6.34 image we had one officially registered with the mainline kernel. This is set in the MBR, so you will have to use the appropriate MBR with the ATAG for your kernel or the kernel will not recognize the board.

Kernel 2.6.21 MBR

Kernel 2.6.34 (and above) MBR

In our official images these will already be written, but you can simply use dd to write these to first 446 bytes of either of the SD cards or mtdcopy for onboard flash. For example:

```
# This should be written to your SD block device, not a partition
dd if=mbr-2.6.34.dd bs=446 count=1 of=/dev/mmcblk0
```

Compiling the Kernel

WARNING: BACKUP YOUR DATA FIRST

Prerequisites

This guide is intended to run on an x86 compatible Linux workstation. While you may be able to compile the kernel on the board, we do not recommend it. A typical workstation compile will take several minutes. The same compile on the board will take several hours.

RHEL/Fedora/CentOS:

```
yum install ncurses-devel ncurses
yum groupinstall "Development Tools" "Development Libraries"
```

```
Ubuntu/Debian:
apt-get install build-essential libncurses5-dev libncursesw5-dev
```

For other distributions, please refer to their documentation to find equivalent tools. Next you need to set up the cross compiler and sources:

```
# Download the cross compile toolchain (OABI)from Technologic Systems:
wget https://files.embeddedTS.com/ts-arm-sbc/ts-7800-linux/cross-toolchains/ts7800-crosstool-linux-gnueabi-2005q3-2.tar.gz

#Extract to current working directory:
tar xvzf ts7800-crosstool-linux-gnueabi-2005q3-2.tar.gz

#Download the kernel sources
wget https://files.embeddedTS.com/ts-arm-sbc/ts-7800-linux/sources/linux-2.6.21-ts-src-latest.tar.gz

#Extract the source tarball.
tar xvzf linux-2.6.21-ts-src-latest.tar.gz

cd linux-2.6.21-ts
```

Export the CROSS_COMPILE variable with a path that points to the appropriate cross-compiler. If you followed the steps above exactly, the toolchain is extracted into the same directory as the kernel.

```
export CROSS_COMPILE=../arm-none-linux-gnueabi/bin/arm-none-linux-gnueabi-
export ARCH=arm
```

Now you will want to configure your settings. You can start with our default config by running:

```
make ts7800_defconfig
```

At this point you can configure any additional options with:

```
make menuconfig
```

For example, to include CIFS support use the arrow and Enter keys to navigate to Filesystems -> Network File Systems -> CIFS Support. Press "y" to include CIFS support into the kernel (alternatively, you could modularize the feature with "m" so you can enable or disable the module on demand which will also enable you to simply copy/paste the cifs.ko into the correct path in the kernel instead of copying the entire kernel (outlined below in appendix)). Keep hitting "exit" until you're prompted to save changes, choose "yes".

After you have made your selection, you can build your kernel and modules with:

```
make && make modules
```

Note: If you receive an error about PATH_MAX not being defined, you will need to edit scripts/mod/sumversion.c and add "#define PATH_MAX 1024".

The new kernel will be at "arch/arm/boot" in a compressed format called zImage (the uncompressed version is simply called Image, but it is REQUIRED that it be < 3.1MB (3145728 bytes in size). If it does not fit, you can go back in and change any changed options from included (*) to modules (m).

Once you have a kernel that fits you can install this to the kernel partition using dd.

```
# Assuming your SD card is /dev/sdb
dd if=arch/arm/boot/zImage of=/dev/sdb2 bs=4M conv=fsync
```

For more information on installing the kernel see the #Backup / Restore section.

You will need to install several of these modules to the initrd (third partition) of the SD card. Mount the third partition of the SD card and copy these files:

```
# Create a directory to mount this device
mkdir -p /mnt/sd/

# Note, your device may be something different.  Check 'dmesg' to get the correct name.  On my system, this is /dev/sdg.
# If you receive an error here, try restoring to the stock image first.  You may also need
# to load kernel modules for filesystem support.

mount /dev/sdg3 /mnt/sd

# This assumes you are still in the root of the kernel sources
cp ./drivers/mtd/nand/ts7800.ko /mnt/sd/
cp ./drivers/mmc/tssdcard.ko /mnt/sd
cp ./drivers/serial/tsuart1.ko /mnt/sd/
cp ./drivers/serial/tsuart7800.ko /mnt/sd/
cp ./drivers/usb/core/usbcore.ko /mnt/sd/
cp ./drivers/usb/host/ehci-hcd.ko /mnt/sd/

umount /mnt/sd && sync && sync
```

To install the modules in the Debian partition:

```
mkdir newmodules
INSTALL_MOD_PATH=newmodules make modules_install

# Again, your device may be something different.  Check 'dmesg' to get the correct name.  On my system, this is /dev/sdg.
mount /dev/sdg4 /mnt/sd

# Remove the old modules
rm -rf /mnt/sd/lib/modules/2.6.21-ts/

# Install the new modules
cp -r newmodules/lib/ /mnt/sd/

umount /mnt/sd && sync && sync
```

Once you boot up to the board to Debian, you need to run 'depmod' once to calculate module dependencies. This will not work from the fastboot as it cannot write to this directory. After this is run, you can 'modprobe <module>' for your specific device support.

While we strongly recommend the 2.6.21 kernel we also provide a 2.6.34 kernel sources here. Note that these are not provided with any support. You can follow the same steps as above for using this kernel, making sure that you use the updated #ATAGS, the different sources, and using this toolchain instead.

Features

CPU Functionality

The TS-7800 features a Marvell 88F5182. You may want to refer to the CPU user guide for more details on the advanced usage of the CPU features.

MMU

The 88F5182 features a Memory Management Unit, enabling high level operating systems such as Embedded Linux and Windows CE to run on the TS-7800. In the same way, the Linux TS-Kernel takes advantage of the MMU functionality. The MMU is controlled by page tables stored in system memory and is responsible for virtual address to physical address translation, memory protection through access permissions and domains, MMU cache and write buffer access. In doing so, software applications can access larger "virtual" memory space than the available physical memory size, allowing multiple programs to run and use the system memory simultaneously.

Interrupts

The 88F5182 has 64 interrupts available on chip as well as another 32 implemented in the Lattice FPGA by using the doorbell interrupt register. The CPU also has 26 GPIO pins that are configurable for interrupt use. The 88F5182 includes an interrupt controller that routes internal interrupt requests as well as external interrupt requests (GPIOs) to the Feroceon CPU core. The 88F5182 interrupt controller drives two interrupt signals to the Feroceon CPU core -- FIQ (high priority) and IRQ (regular priority). All interrupts are level sensitive. The interrupt is kept active as long as there is at least one non-masked cause bit set in the Interrupt Cause register. The 88F5182 can also be used as the interrupt controller for external devices generating interrupts to the Feroceon CPU core via GPIO inputs.

The interrupt controller can also receive interrupt messages from an external PCI Express device. The 88F5182 can also act as a PCI or PCI Express Endpoint. As such, it can generate the PCI Express INTA emulation message or the INTAn signal.

For more information about the interrupt functionalities, refer to the 88F5182 User's Guide.

Onboard Flash

The TS-7800 uses a 512MB NAND Flash chip for its on-board Flash resource. The physical address of the Flash chip is 0xe800_0800. The on-board flash is broken up into partitions and accessed through the Linux driver framework known as "MTD", or Memory Technology Device. The partitioning is dynamic and depends on the DOS-style MBR found at sector 0 of the flash. This MBR can be changed by using the "fdisk" command on the /dev/mtdblock/0 device, but doing so is not recommended.

USB Host

The USB Connector on the TS-7800 provide two high-speed USB interfaces for the user. These are directly connected to the MV88F5182 processor, which integrates an USB dual-port Open Host Controller Interface (OHCI), providing full-speed serial communications ports at a baud rate of 480 Mbits/sec. Up to 127 USB devices (printer, mouse, camera, keyboard, etc.) and USB hubs can be connected to the USB host in the USB "tiered-star" topology. This includes the following features: USB 2.0 compatible OHCI Rev 1.0 compliant USB device connections support at both low-speed (1.5 Mbps) full-speed (12 Mbps) and hi-speed (480 Mbps) Root HUB integrated with 2 downstream USB ports Transceiver buffers integrated, over-current protection on ports Supports power management Operates as a master on the bus

Additional non-volatile storage may be added with a USB flash drive. This device supplies additional non-volatile storage either for data or for a complete operation system distribution, such as Debian. A tar-file of Debian is available on the Technologic Systems FTP site.

SATA

The CPU provides two Marvell SATA 2.0 connections. These allow connection of mass storage devices in excess of 500GB. The drivers required to operate the SATA connectors in Sarge are thus: mv_sata, sd_mod, scsi_mod. Note while the TS-7800 cannot boot directly from a SATA device, chroot and pivot root can be used to change the root partition to one of these devices.

DDR RAM

The TS-7800 uses DDR RAM technology to provide 128 MB of high-speed volatile memory. The memory is soldered directly to the board, making the TS-7800 more reliable in high-vibration environments.

The TS-7800's RAM is not contiguous in the physical memory map of the 88F5182. But the MMU is programmed to remap the blocks of RAM to appear as a contiguous block of memory at the very beginning of the virtual memory map. Refer to the MMU section of this manual to understand how the physical memory is mapped and the virtual memory is translated.

> **Note:** It is not possible to use sizes of the DDR RAM other than the standard 128MB.

Onboard NAND Flash

The onboard flash is broken up into partitions and accessed through the Linux driver framework known as "MTD", or Memory Technology Device. The partitioning is dynamic and depends on the DOS-style MBR found at sector 0 of the flash. This MBR can be changed by using the "fdisk" command on the /dev/mtdblock/0 device, but doing so is not recommended.

/dev/mtdblock0 - Whole disk block device driver. /dev/mtdblock1 - First MBR partition (bootloader kernel binary) /dev/mtdblock2 - Second MBR partition (bootloader initrd) /dev/mtdblock3 - Third MBR partition (Linux YAFFS2 filesystem)

The TS-7800 uses a 512MB NAND Flash chip for its on-board Flash resource. The physical address of the Flash chip is 0xe800_0800. The on-board flash is broken up into partitions and accessed through the Linux driver framework known as "MTD", or Memory Technology Device. The partitioning is dynamic and depends on the DOS-style MBR found at sector 0 of the flash. This MBR can be changed by using the "fdisk" command on the /dev/mtdblock/0 device, but doing so is not recommended.

The common usage of using the NAND flash is by first preparing an SD card with the desired image. Once this is written you can run the 'createmtdroot' command from the initrd which will copy the image on the SD card to the Onboard flash. This assumes you are using the default partitioning scheme we use on the SD card, and the partitions have not been extended past 512MB.

Linux uses a NAND flash filesystem called YAFFS2 for general purpose file storage. This filesystem is a log-structured filesystem which is safe against corruption caused by system crashes and power loss without the need for consistency checking on next boot. A normal PC cannot use this filesystem as it is specifically designed for NAND flash which a PC does not have.

FPGA Functionality

The 12,000 LUT Lattice FPGA is an integral part of the TS-7800 design. When the TS-7800 is powered on, the FPGA comes on first and it then boots the CPU, using boot code from a small EEPROM chip and from either the on-board flash or the SD card. After an operating system has booted, the FPGA provides the CPU access through its PCI bus to the NAND flash, the SD cards, and the PC/104 connector. 89 PC/104 pins are connected to the FPGA. The FPGA load allows these pins to be used either as a standard PC/104 bus or for GPIO.

The default FPGA has several cores available at 0xe8000000 as a 32-bit bus. Most of these are abstracted by existing drivers or utilities.

Offset	Description
0x00-0xff	Syscon and TSUARTs
0x100-0x1ff	SD card controller
0x200-0x2ff	IRQ controller
0x400-0x4ff	DMA channel interface
0x800-0x8ff	NAND controller

FPGA Changelog

FPGA Revision Log

Revision	Changes
1	Initial Release
2	Support for REV B boards with fixed PC104
3	Fixed SD writeprotect
4	RS485 half duplex ignores RX when TX
5	Fixed RS485 TXEN at bootup
6	Custom core update, No standard load changes
7	Fix for PC104 C13/C14
8	CTS3 added in register 0xc
9	Fixed regression in TS-UARTS from REV8
10	Samsung NAND support

SD

The SD card has a slightly different partition scheme than on the NAND to facilitate usage on a host PC. The micro-SD card shows up as /dev/tssdcarda while the full size SD card slot shows up as /dev/tssdcardb on the TS-7800. Both cards can be used together and the TS-7800 will attempt to boot from the micro-SD card before the full size SD card.

/dev/tssdcard(a-b)/1 - First MBR partition (Formatted FAT32, SD card bought from Technologic Systems contains the Eclipse IDE) /dev/tssdcard(a-b)/2 - Second MBR partition (bootloader kernel binary) /dev/tssdcard(a-b)/3 - Third MBR partition (bootloader initrd) /dev/tssdcard(a-b)/4 - Fourth MBR partition (Linux JFS filesystem)

Note that the MBR installed by default on the TS-7800 contains a 446 byte bootloader program that loads the initial power-on kernel and initrd from the first and second partitions. Replacing it with a MBR found on a PC would not work as a PC MBR contains an x86 code bootup program.

Technologic Systems has a full license for using the additional SD features which are reserved for members of the SD Card Association. This has allowed us to design both the hardware logic core and software specifically tuned to the capabilities of the TS-7800 CPU using the official SD specification documents. Since both a Linux driver module and an ARM9 object file containing OS-independent access routines are provided to customers purchasing the board hardware, customers do not have to seek SD licensing themselves.

SD Memory Card technology provides large capacity and fast access combined with a compact and slim profile, making it very appealing for a wide range of next generation products and applications. In addition, SD Cards feature content protection, planned capacity growth, high-speed data transfer, and a write protect switch. These devices supply additional non-volatile storage either for data or for a complete operation system distribution, such as Debian, to be used with the TS-7800 SBC.

The TS-7800 fully supports SDHC cards as well.

Syscon

By default the Lattice FPGA on the TS-7800 is loaded with a system controller core at base physical address 0xe8000000. Add that base to the offsets below to access these registers. With the exception of the UART control registers, these registers are all 32 bits wide and should therefore be accessed with 32 bit writes. The UART registers are 16 bit registers.

Offset	Bits	Access	Description
0x0	7-0	Read Only	FPGA Revision
	31-8	Read Only	Board ID: 0xb480
0x4	15-0	Read/Write	DIO header data in pins 16-1
	29-16	Read/Write	LCD header data in pins 14-1
	30	Read Only	JP1 status (1 = on)
	31	Read Only	JP2 status (1 = on)
0x8	15:0	Read/Write	DIO Header Data Output (pin 1:16)
	29:16	Read/Write	LCD header Data Output (pin 1:14)
	30	Read/Write	Green LED (1 is on)
	31	Read/Write	Temperature Sensor chipselect (1 selected)
0xc	12-0	Read/Write	PC104 Configuration
	13	Read/Write	UART RTS for COM1
	14	Read/Write	UART DTR for COM1
	15	Read/Write	1 = RS422 on COM2 using TSUART2 0 = dual RS485 using TSUART2 and TSUART3
	16	Read/Write	UART4 RTS on COM3
	31-17	Read Only	Reserved
0x10	31-0	Read/Write	PC104 row A GPIO data
0x14	31-0	Read/Write	PC104 row B GPIO data
0x18	19-0	Read/Write	PC104 row C GPIO data
	31-20	Read Only	Reserved
0x20	31-0	Read/Write	PC104 row A GPIO data direction
0x24	31-0	Read/Write	PC104 row B GPIO data direction
0x28	19-0	Read/Write	PC104 row C GPIO data direction
	31-20	Read Only	Reserved
0x2c	19-0	Reda/Write	PC104 row D GPIO data direction
	31-20	Read Only	Reserved
0x30	31-0	Read/Write	PC104 row A MUX function
0x34	31-0	Read/Write	PC104 row B MUX function
0x38	19-0	Read/Write	PC104 row C MUX function
	31-20	Read Only	Reserved
0x3c	19-0	Read/Write	PC104 row D MUX function
	31-20	Read Only	Reserved
0x40	31-0	Read Only	Free running microsecond counter
0x44	31-0	Read Only	32 bits of random data updated every second
0xC0-0xE6	15-0	Read/Write	TSUART control registers

TSUART

The TS-UART design used in the TS-7800 is fundamentally the same as the TS-UARTs in other TS boards. The TS-UART is a compact, two register design created to require minimal space in an FPGA. The two registers are STAT and DATA. The layout of the STAT register is:

Bits	Access	Name	Description
0	Read Only	TBRE	Transmit buffer not full
1	Read Only	DR	Receive data ready
2	Read Only	OERR	Receive overflow
3	Read Only	CTS	UART CTS
4	Read/Write	RTS	UART RTS
7:5	Read/Write	BAUD	**Value Rate** 0 115200 1 57600 2 38400 3 19200 4 9600 5 4800 6 2400 7 Off
8	Read/Write	NINEBIT	Enable 9-bit mode
9	Read Only	RBREAK	BREAK detected
10	Read/Write	TBREAK	Transmit Break
11	Read Only	DMXS	Reserved
12	Read Only	TRE	Transmitter empty
13	N/A	SLOW	Reserved
15:14	N/A	N/A	Reserved

Please consult the driver source code files tsuart1.c and tsuart-7800.c provided in the kernel sources for more information on implementing a device driver for the TS-UART on the TS-7800.

PC104

To access peripherals on the PC104 bus it is necessary to add the base address from the table below to the offset of the peripheral to get a memory address for accessing the peripheral. For example, for ISA 8-bit I/O address 0x100, add 0xEE000000 to 0x100 to get 0xEE000100.

	Memory	I/O
8-bit	0xEC000000	0xEE000000
16-bit	0xED000000	0xEF000000

IRQs 5, 6, and 7 on the PC104 bus from the TS-7800 are 64 + the PC104 IRQ number. These will be IRQs 69, 70, and 71.

WARNING: The Ethernet connector and the electrolytic capacitor near the PC/104 bus are near the edge of the PC/104 specification. It is advised to apply a piece of electrical tape on top of these components when using a PC/104 daughter board to help prevent any damage.

The TS-7800 provides control over some of the ISA parameters of the PC-104 bus through a 32-bit register located at address 0xE800000C, which is defined as follows:

Bit(s)	Function
5-0	ISA strobe length
9-6	ISA setup length
10	Honor ISA 0WS/ENDX signal (1=true)
11	TS special ISA pinout enable (1=true)
12	ISA oscillator select **Value** / **Function** 0 / high-jitter approximation of 14.318Mhz 1 / clean 25Mhz

(Other bits in this register should be masked out.)

The ISA strobe length and ISA setup length are both given as the number of extra 10ns periods.

The ISA strobe length is the amount of additional time that ISA_IOR, ISA_IOW, ISA_MEMR, and ISA_MEMW are held asserted. The minimum (when bits are zero) is 20ns. The default power-on value is 40, for a 420ns strobe length. If configured to honor the ISA 0WS/ENDX signal, the peripheral will skip the remaining strobe time for an early transaction end, allowing for faster devices than standard ISA allows.

The ISA setup length is the additional amount of time above 20ns that the address and data are held stable before asserting the strobe. The default is 14 (160ns).

There is an additional 20ns hold time at the end of the strobe where address and data are kept valid. The default total bus cycle length is then 160ns (setup) plus 420ns (strobe) plus 20ns (hold) for a total of 500ns (2Mhz). This is very conservative for modern hardware and most designs can actually run much faster.

The TS special ISA pinout is designed to enable compatibility with products from TS that use the 16-bit bus on the 64-pin PC/104 header. For example the TS-ETH2 is a 16-bit peripheral that has a jumper labeled "ARM." When this jumper is shorted it will expect to use the TS special ISA pinout, if it is left open it will expect to use standard PC/104 16-bit pinout. Please note that disabling the special pinout is generally not required as both 8 and 16-bit bus cycles will still function normally when it is enabled. Disabling the special ISA mode will result in quirky behavior that may show double read strobes.

PC104 GPIO

The PC-104 connector can be multiplexed between PC-104 functionality and GPIO functionality. There are up to 89 general purpose digital I/O pins available. This corresponds to 104 pins total minus fifteen pins carrying power or ground rails.

Each GPIO pin has a two corresponding register bits, one in the GPIO data register, and one in the GPIO direction register. The direction register determines if the pin is an output (actively driven) or an input (tri-stated, driven externally). A high ("1") value sets a particular pin to be an output, while a low ("0") value sets it to be an input. The data register, when read, contains the current state of all pins in GPIO mode. When written, the value written will determine the state of all pins in GPIO mode, but only for pins which have their direction set to "output".

The GPIO register map is as follows:

Address	Row	Register
0xE8000010	A	Data
0xE8000014	B	Data
0xE8000018	C	Data
0xE800001C	D	Data
0xE8000020	A	Direction
0xE8000024	B	Direction
0xE8000028	C	Direction
0xE800002C	D	Direction

A simple command-line example to toggle pins on Row A is thus:

```
#connect an LED from ground (cathode) to Row A pin 2 (andode).
#set row A to all GPIO mode
peekpoke 32 0xe8000030 0x0
#set row A to all output:
peekpoke 32 0xe8000020 0xffffffff
#turn on the pin we attached an LED to with 0b000000010
peekpoke 32 0xe8000010 0x2
#turn off the pin
peekpoke 32 0xe8000010 0x0
#... or turn on the whole row:
peekpoke 32 0xe8000010 0xffffffff
```

Row A is the row nearest the edge of the board on the longer PC104 socket. Pin 1 is at the end nearest the DB9 connector. From outside (nearest to the edge of the board) to inside (closest to the center of the board) the pin row labels are D, C, A, B. For more pin descriptions and positions, see the table below in section 7.11.

PC104 16550

Many of our PC104 peripherals interface with the TS-7800 using a 16550 UART. To connect to these devices we provide a generic driver. In this example, I'm using a TS-SER4 with jumpers IRQ2 and IRQ4 on.

```
pc104on
modprobe ts7800_isa16550 irq=6 io=0x3e8
```

```
# On the TS-SER4, we have multiple 16550 devices.  To use
# these, you must remove the module and reinsert it with the
# options for the next port.  This will not remove the /dev/ entry for the original
# device, and to actually remove it you must reboot the board.
rmmod ts7800_isa16550
modprobe ts7800_isa16550 irq=6 io=0x2e8
```

After loading each driver, dmesg will indicate the new device name:

```
serial8250: **ttyS2** at MMIO 0xee0003e8 (irq = 70) is a 16550A
```

FPGA DMA

The TS-7800 supports DMA to transfer data between FPGA cores and the PCI interface. This is supported in our BSP with drivers/dma/ts7800dma.c and most users can use the provided drivers that already make use of the DMA cores. This is provided for driver implementation usage only.

The TS-7800 DMA core is at 0x200 (0xE8000200).

<table>
<tr><th>Offset</th><th>Bits</th><th>Description</th></tr>
<tr><td rowspan="10">0x0</td><td>31</td><td>DRQ2 input (FPGA peripheral is ready for DMA)</td></tr>
<tr><td>30</td><td>DRQ1 input (CPU PCI is ready for DMA)</td></tr>
<tr><td>29:21</td><td>Reserved</td></tr>
<tr><td>20:19</td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>1 byte</td></tr><tr><td>1</td><td>2 bytes</td></tr><tr><td>2</td><td>4 bytes</td></tr><tr><td>3</td><td>Reserved</td></tr></table></td></tr>
<tr><td>18:16</td><td>CPU PCI address increment per transfer</td></tr>
<tr><td>15:13</td><td>FPGA peripheral address increment per transfer</td></tr>
<tr><td>12</td><td>DMA direction (0 - CPU PCI reads, 1 - FPGA peripheral reads)</td></tr>
<tr><td>11:0</td><td>DMA length (number of transfers, not necessarily bytes)</td></tr>
</table>

When the DMA operation starts:

- The IRQ output goes to 0

When DMA operation is in progress:

- IRQ signal will still be 0
- DMA length (bits 11:0 in DMA control reg) will be decrementing to zero.
- wbm1 address and wbm2 address regs will be incrementing as configured in bits 18:13 of the DMA control reg

When DMA operation completes:

- The irq signal is 1
- DMA length (bits 11:0 in DMA control reg) will be zero.
- wbm1 address and wbm2 address regs will be pointing at the ending address plus 1 address increment (bits 18:13)

When the DMA is idle:

- The IRQ signal is 1

The IRQ is connected to the #FPGA IRQ Controller. IRQ0 high indicates the CPU DMA operation is complete, and IRQ1 indicates the FPGA DMA operation is complete.

FPGA IRQ Controller

The TS-7800 includes an IRQ controller which expands a CPU GPIO (MPP_7) into 32 other FPGA IRQs.

IRQ	Description
0	dma_cpu_pci_dack_o
1	dma_fpga_dack_o
2	SD Busy#
3	isa_irq3
4	isa_irq4
5	isa_irq5
6	isa_irq6
7	isa_irq7
8	Reserved
9	isa_irq9
10	isa_irq10
11	isa_irq11
12	isa_irq12
13	Reserved
14	isa_irq14
15	isa_irq15
16:25	tsuart_irqs
27:26	reserved
31:28	Reserved

AVR Functionality

The AVR controls power to the CPU. The CPU communicates with the AVR using two wire serial interface. (TWSI) The utility program ts7800ctl can be used to access features implemented in the AVR, including a watchdog timer, sleep mode, analog to digital inputs, and the on-board red LED. Run ts7800ctl with no arguments for an explanation of usage.

Sleep Mode

The TS-7800 implements a sleep mode that allows the board to go down to 200 microamps. This powers off the TS-7800 while the AVR waits for the specified time to wake up the board. You can sleep for 5 seconds by running this:

```
ts7800ctl -s 5
```

This only allows the board to sleep for seconds, not smaller units of time. You can however wake it up immediately from an external interface by driving DIO_04 or ISA_B32 high.

This sleep functionality will only work if power is fed in through the +5V connector on the TS-7800 itself. If you are using a PC104 device like the TS-POE100 or the TS-BAT10 then you would have to cut the +5V pins on PC104 and bring it down to the TS-7800 power connector in order to use the sleep functionality. Otherwise the board can be powered through PC104 with no modification.

ADC Sampling

WARNING: If you ordered your board before 09/2011 then you may want to update the AVR. See [Product Errata].

The TS-7800 provides an analog to digital capture function by way of an on-board AVR microcontroller. The microcontroller has six 10 bit 0-3.3V analog inputs and is configured to sample at a rate of 2 kHz. Five of the six analog input channels are brought out to the A/D header (see table below), the remaining channel (channel 6) is connected to Vcc of the optional switching regulator (8-30V). A voltage divider is used to divide Vcc / 9 in order to ensure the analog voltage doesn't exceed 3.3V.

The A/D header is laid out as follows:

GND	GND	GND	GND	GND
2	4	6	8	10
1	3	5	7	9
CH0	CH1	CH3	CH2	CH7

The ADC is implemented using ts7800ctl:

```
root@ts7800:root# ts7800ctl -S 0,1
[0x00000000, 0]=1023
[0x00000001, 1]=0810
[0x00000002, 0]=1023
[0x00000003, 1]=1023
[0x00000004, 0]=1023
[0x00000005, 1]=1023
```

This requests ts7800ctl to output ADC in string data. This returns:

```
 [Hex counter, Channel]=Value
```

If you prefer to implement this in your own code, you can get [ts7800ctl.c here].

Examples

```
# Capture 1000 raw binary samples from channel 0 and print the output to a file in a comma separated format.
ts7800ctl -r"0" | dd bs=2 count=1000 2>/dev/null | hexdump -v -e '1/2 "%u, "' > ch0_500_samples.csv

#Capture 500 samples from channel 0 and 500 samples from channel 1. Write the
#samples to a file where all channel 0 samples are in the first column and all
#channel 1 samples are in the second column.
ts7800ctl -r"0-1" | dd bs=2 count=1000 | hexdump -v -e '1/2 "0x%x\t" 1/2 "0x%x\n"' > ch0_ch1_1k_samples.dat

#Capture 1000 samples from channel 0, and then print the output to standard out in ASCII.
ts7800ctl -S"0,1" 2>/dev/null | dd bs=19 count=1000

#Capture 5 samples from channel 0 and 5 samples from channel 1, then print the output to standard out in ASCII.
ts7800ctl -S"0-1" 2>/dev/null | dd bs=19 count=10
```

CPLD Functionality

The Lattice CPLD on the TS-7800 allows the FPGA to control the DIO and LCD headers, COM3, and jumpers. See the respective sections for more information. The CPLD is 5V tolerant once it is powered on, however 0 to 3.3V logic levels on pins connected directly to the CPLD are still recommended. If 5V is applied to a CPLD pin before the CPLD has come up during the boot process, this can damage the CPLD.

LCD and DIO

Although the distinction is transparent to the user, the most of the pins on the DIO and LCD headers are driven by CPLD, which means that they are 5V tolerant. However logic levels of 0-3.3V are still recommended. Applying 5V to a pin that is connected to the CPLD before it has been initialized may damage the CPLD. The only pins that come directly from the FPGA are output pins for the SPI bus. The SPI_MISO pin goes through a separate buffer chip to get to the FPGA so it is also 5V tolerant.

The DIO and LCD headers are controlled by 32 bit registers at 0xe8000004 and 0xe8000008. Bits 15-0 control the DIO header and bits 29-16 control the LCD header. The register at 0xe8000008 is the output register. Writing a 0 drives the pin low but writing a 1 only tri-states. To use these pins for input, write a 1 to the output register and read the register at 0xe8000004. See the #LCD Header and #DIO Header sections for more details.

Serial Ports

The TS-7800 has eight UARTs. Two of these UARTs, which appear on the COM1 (DB9) header and the COM2 (10-pin) header are driven by the CPU. These ports show up in "/dev" as `ttyS0` and `ttyS1`.

The other six UARTs are TS-UARTs driven by the FPGA. These ports are available in "/dev" as `tttsX` where X is the UART number listed in the table below.

Please note that the pin numbering on the 10 pin COM headers is as shown here:

6	7	8	9	10
1	2	3	4	5

When some TS-UART ports are enabled, they override the meanings of other pins with their own meanings. When disabled, the pins revert to their original meaning. The table below describes the ten TS-UART ports. The "RS-" column indicates whether the port is RS-232, RS-485/422, or TTL. The "base adrs" column indicates the address of the STAT register; the DATA register is offset +2 from that. All TS-UART registers must be accessed through 16-bit memory accesses. The "header" column indicates which header the port is located on, and the "Tx", "Rx", "TxEn", "RTS", and "CTS" indicate the pin number of those respective signals on the header specified.

Number	Type	Base Address	Header	TX	RX	TXEN	RTS	CTS
0	RS232	0xE80000C0	COM1	7	8	N/A	N/A	N/A
1	RS232	0xE80000C4	COM1	4	1	N/A	N/A	N/A
2	RS485/RS422	0xE80000C8	COM2	+6 -1	+4 -9	N/A	N/A	N/A
3	RS485	0xE80000CC	COM2	+4 -9	N/A	N/A	N/A	N/A
4	RS232	0xE80000D0	COM3	3	2	N/A	7	8
5	RS232	0xE80000D4	COM3	7	8	N/A	N/A	N/A
6	TTL	0xE80000D8	DIO	13	15	11	N/A	N/A
7	TTL	0xE80000DC	LCD	13	14	12	N/A	N/A
8	TTL	0xE80000E0	PC104	C17	C18	C16	N/A	N/A
9	TTL	0xE80000E4	PC104	C14	C15	C13	N/A	N/A

Note: The RS485 ports are only available if the option OP-485-FD-14 was ordered with the board. Please submit an RMA for installation of this part.

You can select RS-422 on port 2 by setting bit 15 of register 0xe800000C to 1. If it is 0, it will default to RS485.

All TS-UARTs can be run at one of 7 programmable baud rates, accessed as baud rate indicies 0-6. These baud rates are programmable by accessing the 16-bit address for the index of interest. The baud rate index corresponds to the value written into the TS-UART STAT register's baud rate bits to select that speed. (Note: a baud rate index of 7 is reserved in the TS-UART to mean "disabled".)

Baud Index	Address
0	0xE80000E8
1	0xE80000EA
2	0xE80000EC
3	0xE80000EE
4	0xE80000F0
5	0xE80000F2
6	0xE80000F4

The value of the register can be computed from the following formula: 100,000,000 / (8 * baud_rate)

Realtime Clock

The TS-7800 optionally supports a Non-volatile Battery-backed real-time clock (RTC) which is soldered onto the board. You can select this during the order process as OP-BBRTC. This option uses an ST Micro M48T86PC1 module for the real-time clock function. This module contains the lithium battery, 32.768 kHz crystal, and a RTC chip with 114 bytes of battery-backed CMOS RAM. It will maintain clock operation for a minimum of 10 years in the absence of power.

The RTC is accessed using two registers. The write-only index register is located at physical address location 0xe800_0808 and the RTC data register is location at physical address location 0xe800_080c. These are byte-wide registers with the Index Register property of write only. The Data Register has a read/write property. Valid Index Register values are between 0 and 127, decimal. The first 14 index locations are used for accessing the RTC Time and Date registers. The next 114 locations are non-volatile RAM locations available to the user.

The TS-7800 now ships with an RTC driver, so the RTC can be accessed using the standard hwclock command.

Temperature Sensor

The TS-7800 can be ordered with an optional temperature sensor. Populated at U16 this sensor can provide enclosure-level temperature readings. Sample code for this optional function is available in tempSensor.c.

Status LEDs

The TS-7800 has to on-board LEDs, one red and one green, only the green is available for user software. This LED may be used for diagnostics, status messages, or simple output. When power is first applied to the TS-7800 the green LED will blink momentarily and then turn off, ready to be controlled by software. It is accessible at 0xe800_0008 bit 31, a Logical "1" turns on the LED.

The red LED can be controlled by the ts7800ctl utility. Run 'ts7800ctl --help' for more info.

Jumpers

There are three user settable jumpers located on the top side of the TS-7800 right next to the A/D header. Jumper 1 on forces the TS-7800 to boot from an SD card, this is useful if the board has been bricked, and allows the NAND flash to be reprogrammed. Jumper 2 on enables console output on COM1. Jumper 3 on scales down the processor to run at 333 Mhz which will also lower overall power consumption slightly.

External Interfaces

DIO Header

Access to the DIO header is described here. These pins can be driven low, otherwise they are inputs with pull-up resistors. The pull-ups are via 2.2k Ohms on the odd numbered pins 1-15. Pin 10 has a 4.7k pull-up and is a read-only input. The rest are pulled up by the CPLD through 20k-150k nominal resistance. The pinout for the DIO header is shown below.

There is no built-in SPI bus on the TS-7800, but SPI is easily implemented in software as demonstrated in our sample code.

DIO Header

GND	DIO_04	SPI_FRAME	DIO_08	SPI_MISO	SPI_MOSI	SPI_CLK	3.3V
2	4	6	8	10(RO)	12	14	16
1	3	5	7	9	11	13	15
DIO_01	DIO_03	DIO_05	DIO_07	DIO_09	DIO_11	DIO_13	DIO_15

LCD Header

This header is laid out to drive our LCD-LED product. The file tolcd.c in our samples directory demonstrates this. Like the DIO header, pins 7-14 can be driven low, otherwise they tri-state with a 2.2k Ohm pull-up. Pins 4 and 5 are pulled up by 470 Ohm and 51 Ohm inline resistance, respectively.

LCD Header

GND	BIAS	RW	DB0	DB2	DB4	DB6
2	4	6	8	10	12	14
1	3	5	7	9	11	13
+5V	RS	EN	DB1	DB3	DB5	DB7

Ethernet Port

The MV88F5182 Ethernet LAN controller incorporates all the logic needed to interface directly to any MII compatible Ethernet PHY chip. A low-power Marvell 88E1118 chip is used to implement the Ethernet PHY function and an integrated RJ-45 connector with built-in 10/100/1000 transformer and LED indicators completes the Ethernet sub-system.

The TS-7800 has both a LINK/ACTIVITY LED and a 10/100/1000 speed LED built into each RJ-45 connector that indicates the current Ethernet status. The LINK LED (right side of connector, green) is active when a valid Ethernet link is detected. This LED should be ON whenever the TS-7800 is powered and properly connected to a 10/100/1000BaseT Ethernet network. The LINK/ACTIVITY LED will blink to indicate network activity for either inbound or outbound data. The SPEED LED (left side of connector, green) will be on when a 1000Mb network is detected and off for a 10/100Mb network. Both of these LEDs are controlled by the 88E1118 and do not require any overhead by the processor.

Note: TS-Kernel provides all the software support to use the MV88F5182 10/100/1000 Ethernet core. For more details, find the TCP/IP configuration instructions on the Linux documentation.

SD Connectors

The SD Card socket (ALPS connector) at the back side of the TS-7800, and the Micro-SD card socket on the top of the TS-7800 enable SD Cards to be used as primary storage devices for the TS-7800. The hardware core implemented by Technologic Systems is integrated inside the on-board FPGA. Technologic Systems has written a binary Linux driver module and a set of generic, OS-independent read/write routines for accessing the SD flash inside of an ARM object (.ko) file. The linuxrc startup script assumes that partition 4 of the SD card contains a JFS filesystem. Both SD sockets are bootable and fully support SDHC cards.

Power Supply Connector

The TS-7800 requires regulated 5VDC at 1000 mA @ 500 MHz(maximum).

There is an option available for the TS-7800, OP-SWITCHREG. This is a module added to the rear of the board (near the SD socket) that allows from 8-30VDC input to the TS-7800. The OP-SWITCHREG option does not interfere with any of the TS-7800's connectors.

WARNING: Supply voltages over 6 VDC may damage the TS-7800.

Be sure to use a regulated 5 VDC power supply, preferably with current limiting to 1 to 3 Amps. A current limited supply is very forgiving of common errors during development. A PC power supply that may be capable of supplying 20 Amps or more is not recommended. It is possible to do irreversible damage to the TS-7800 if the polarity on the power leads is reversed.

DB9 Connector

The DB9 connector brings out 1 CPU serial port (COM1, /dev/ttyS0, by default the Linux console) and 2 TSUARTs. Tx and Rx signals for COM1 are on pins 3 and 2. See section 6.5 for more information on TSUARTs. All signals are at RS-232 levels.

COM2 Header

The COM2 header brings out COM2, /dev/ttyS1, with Tx and Rx on pins 3 and 2 at RS-232 levels. If the board was purchased with the RS-485 option, then the RS-485 ports are also available on this header. See the #Serial Ports section for more details.

COM3 Header

The COM3 header brings out two TSUARTs at RS-232 levels. See the #Serial Ports section for more details.

A/D Header

The A/D header makes available 5 channels of 10 bit resolution A/D conversion. See the #ADC Sampling section for more details.

JTAG Header

The JTAG header is used at the factory for programming the FPGA and CPLD. It is not available for JTAG debugging on the CPU. Using the JTAG pins (pin 1-10) to program the FPGA is not supported and not recommended.

Pins 11-16 on this header are in parallel with the COM signals on the DB9 header.

PC104 Header

The PC-104 connector consists of pins in four rows labeled A, B, C, and D. The numbering of the pins in each row is shown below:

D					19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00								
C					19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00								
A	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01
B	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01

The PC-104 connector can be multiplexed between different functionalities including ISA bus and GPIO. The power-up default is GPIO mode, with all I/Os in a neutral state. To enable the PC-104 bus (ISA) signals, it is necessary to write the following values to the registers specified:

```
0x55555555 to address 0xE8000030
0x55555555 to address 0xE8000034
0x55555 to address 0xE8000038
0x55555 to address 0xE800003C
```

More specifically, the functionality of the PC-104 connector can be configured in a more fine-grained manner, two pins at a time. Each pin pair will have one of two functions:

Value	Description
0	GPIO
1	ISA
2	Reserved
3	Reserved

Setting the function of each pair of pins is done by writing the function number to the appropriate pair of bits in the register corresponding to the row in question. The table below shows the bit positions in each register on the top row, while the cells below in the same column give the corresponding pin numbers for each row which are programmed with those bits at the specified register address.

Row	Register bits	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
A	0xE8000030	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
B	0xE8000034	31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
C	0xE8000038							19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00
D	0xE800003C							19 18	17 16	15 14	13 12	11 10	09 08	07 06	05 04	03 02	01 00

For example, to set the function of pins B19 and B20, this the table above indicates to use bits [19:18] of the register at address 0xE8000034.

The function of the PC-104 connector pins are given in the table below. The "ISA" column gives the name of the pin signal when it is configured as ISA, while the "GPIO" column gives the name of the pin signal when it is configured as GPIO. To save space, there are two sets of columns in each table, whereby the pin name is listed first, followed by the ISA signal and then the GPIO signal, and then this order is repeated for the other set of pins on the same physical header.

The 64-pin connector is given first:

Pin	ISA	GPIO	Pin	ISA	GPIO
A1	IOCHK#	A[0]	B1	GND	GND
A2	D7	A[1]	B2	RESET	B[1]
A3	D6	A[2]	B3	+5V	+5V
A4	D5	A[3]	B4	IRQ9	B[3]
A5	D4	A[4]	B5	3.3V	3.3V
A6	D3	A[5]	B6	DRQ2	B[5]
A7	D2	A[6]	B7	NC	B[6]
A8	D1	A[7]	B8	ENDX#	B[7]
A9	D0	A[8]	B9	8V_30V	8V_30V
A10	IORDY	A[9]	B10	GND	GND
A11	AEN	A[10]	B11	MEMW#	B[10]
A12	A19	A[11]	B12	MEMR#	B[11]
A13	A18	A[12]	B13	IOW#	B[12]
A14	A17	A[13]	B14	IOR#	B[13]
A15	A16	A[14]	B15	DACK3#	B[14]
A16	A15	A[15]	B16	DRQ3	B[15]
A17	A14	A[16]	B17	DACK1#	B[16]
A18	A13	A[17]	B18	DRQ1	B[17]
A19	A12	A[18]	B19	RFRSH#	B[18]
A20	A11	A[19]	B20	BCLK	B[19]
A21	A10	A[20]	B21	IRQ7	B[20]
A22	A9	A[21]	B22	IRQ6	B[21]
A23	A8	A[22]	B23	IRQ5	B[22]
A24	A7	A[23]	B24	IRQ4	B[23]
A25	A6	A[24]	B25	IRQ3	B[24]
A26	A5	A[25]	B26	DACK2#	B[25]
A27	A4	A[26]	B27	TC	B[26]
A28	A3	A[27]	B28	BALE	B[27]
A29	A2	A[28]	B29	+5V	+5V
A30	A1	A[29]	B30	OSC	B[29]
A31	A0	A[30]	B31	GND	GND
A32	GND	GND	B32	ISA_B32	B[31]

Here are the pin assignments for the 40-pin connector:

Pin	ISA	GPIO	Pin	ISA	GPIO
C0	GND	GND	D0	GND	GND
C1	SBHE#	C[1]	D1	MEM16#	D[1]
C2	LA23	C[2]	D2	IO16#	D[2]
C3	LA22	C[3]	D3	IRQ10	D[3]
C4	LA21	C[4]	D4	IRQ11	D[4] (RO)
C5	LA20	C[5]	D5	IRQ12	D[5] (RO)
C6	LA19	C[6]	D6	IRQ15	D[6] (RO)
C7	LA18	C[7]	D7	IRQ14	D[7] (RO)
C8	LA17	C[8]	D8	3.3V	3.3V
C9	MEMR#	C[9]	D9	DRQ0	D[9]
C10	MEMW#	C[10]	D10	DACK5#	D[10]
C11	SD8	C[11]	D11	DRQ5	D[11]
C12	SD9	C[12]	D12	DACK6#	D[12]
C13	SD10	C[13]	D13	DRQ6	D[13]
C14	SD11	C[14]	D14	DACK7#	D[14]
C15	SD12	C[15]	D15	DRQ7	D[15]
C16	SD13	C[16]	D16	+5V	+5V
C17	SD14	C[17]	D17	MASTER#	D[17]
C18	SD15	C[18]	D18	GND	GND
C19	GND	GND	D19	GND	GND

Note: The GPIO nomenclature in these tables is such that, for example, "A[0]" means "Bit 0 of GPIO Register A", and in general "X[n]" means "Bit n of GPIO Register X" and "X[n:m]" means "Bits n through m of GPIO register X", where X is one of A, B, C, or D.

External Links

- IBM developerWorks: Build an embedded Linux distro from scratch

Product Notes

Errata

Synopsis	Multiple problems in AVR functionality
Severity	Medium
Class	Software
Affected	ll TS-7800s
Status	Workarounds available

Description:

The supervisory AVR microcontroller used for implementing the watchdog, red LED, 128 byte EEPROM storage, and analog input channels as well as the "ts7800ctl" userspace control application is prone to lock up, corrupt EEPROM, and spontaneously disable the watchdog upon manipulation of the red LED.

Workaround:

A AVR reflash package is available for download at https://files.embeddedTS.com/ts-arm-sbc/ts-7800-linux/binaries/ts-utils/7800_avr_reflash.tar.gz. Run the instructions contained in the README to update both the AVR as well as the "ts7800ctl" executable. Also, beware that some of the TWI protocol has changed to work around some protocol deficiencies as well as newly discovered Marvell CPU TWI bugs. Customer C code based on old versions of ts7800ctl.c should continue to work, but should not expect any improved behavior. It is suggested that any customer application that has used code taken from previous versions of the ts7800ctl utility be modified to use the new code in the above tar.gz package.

FCC Advisory

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used properly (that is, in strict accordance with the manufacturer's instructions), may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class A digital device in accordance with the specifications in Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the owner will be required to correct the interference at his own expense.

If this equipment does cause interference, which can be determined by turning the unit on and off, the user is encouraged to try the following measures to correct the interference:

Reorient the receiving antenna. Relocate the unit with respect to the receiver. Plug the unit into a different outlet so that the unit and receiver are on different branch circuits. Ensure that mounting screws and connector attachment screws are tightly secured. Ensure that good quality, shielded, and grounded cables are used for all data communications. If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The following booklets prepared by the Federal Communications Commission (FCC) may also prove helpful:

How to Identify and Resolve Radio-TV Interference Problems (Stock No. 004-000-000345-4) Interface Handbook (Stock No. 004-000-004505-7) These booklets may be purchased from the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402.

Limited Warranty

See our Terms and Conditions for more details.

Trademarks

Arm9 is a trademark, and Arm is a registered trademark, of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Retrieved from "http://docs.embeddedTS.com/index.php?title=TS-7800&oldid=16580"

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP