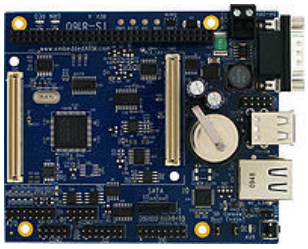
From Technologic Systems Manuals

TS-8160-4200

Contents

TS-8160-4200



Released Mar. 2011

Product Page (http://www.embeddedarm.com/products/board-detail.php?product=TS-8160-4200)

TS-4200 Documents

Schematic (http://www.embeddedarm.com/documentation/ts-4200-schematic.pdf)

Mechanical Drawing (http://www.embeddedarm.com/documentation/ts-4200-mechanical.pdf)

FTP Path (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4200-linux/)

TS-8160 Documents

Schematic (http://www.embeddedarm.com/documentation/ts-8160-schematic.pdf)

Mechanical Drawing (http://www.embeddedarm.com/documentation/ts-8160-mechanical.pdf)

FTP Path (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-8100/)

Processor

Atmel 396MHz AT91SAM9G20

CPU Series Website (http://www.atmel.com/devices/sam9g20.aspx)

CPU Datasheet (http://www.atmel.com/Images/doc6384.pdf)

RAM

64-128MB

FPGA

Actel A3P125

DIO

28

External Interfaces

PC/104 Bus

USB 2.0 1x host

1x 10/100 Ethernet

1x I2C/TWI

SPI

3x RS232

1x RS-232/Console Port

1-2x RS485 Port

Character Display

6x ADC

Internal Storage Media

1x SD

256MB XNAND

Power Requirements

5-28VDC

Operates around 2W

Operating Temperature

- -40C
- 85C

Mechanical

114.30mm X 100.00mm

Height 19.22mm

Weight 83.10 (approx)

1 Overview

The TS-8100 baseboard is an upgrade path from our TS-7350/TS-7370 series providing a PC104 bus, multiple serial ports, LCD, and a DIO header.

2 Getting Started

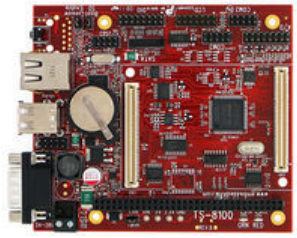
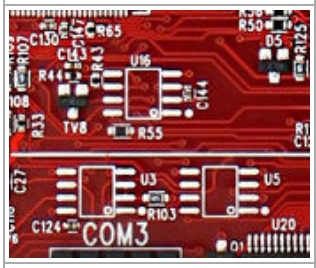
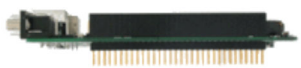
A Linux PC is recommended for development. For developers who use Windows, virtualized Linux using VMWare or virtualbox is recommended in order to make the full power of Linux available. The developer will need to be comfortable with Linux anyway in order to work with embedded Linux on the macrocontroller. 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.

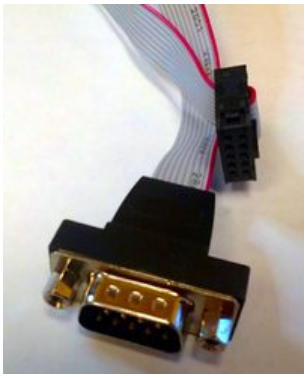
Once you have a development environment you should continue in the next few chapters for configuring the board and powering up the system.

2.1 TS-8100 Options and Accessories

TS-8100 Options

Item	Description
	The TS-8100 is a TS-Socket Baseboard (http://www.embeddedarm.com/products/ts-socket.php#ts-socket-baseboards) including a PC104 bus, DIO, RS232, RS485, and an optional second ethernet.
	The optional "OP-ETH" replaces the push button and instead populates a second ethernet port. This utilizes a USB ethernet which will work with any macrocontroller.
	The optional "OP-CAN2-485" includes the optional second #CAN and RS485 transceiver.
	The option "OP-STHRU-64" includes a 64 pin PC104 connector that extends the pins so the TS-8100 does not need to be the base of the PC104 stack. This option is not required for stacking PC104 peripehral on the TS-8100 as the peripherals include a stack through connector.

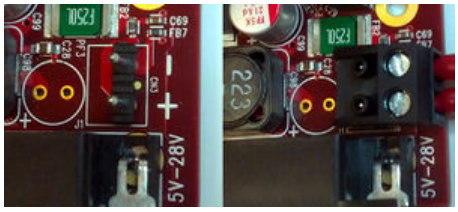
The other options include:

Item	Description
	The RC-DB9 brings out the 10 pin connection to a DB9. See the RC-DB9 page for the exact pinout. The red line indicates pin 1 and should be lined up with the white dot on the board.
CB-DB9-Y	CB-DB9-Y description
	The CB-USB-AF5P connects from a standard 5 pin 0.1" pitch header to a USB A host. This can be used to expose a single USB port while keeping the rest internal to your own enclosure.
	The CB7-05 is a 5 foot null modem cable. This is commonly used to connect to your workstation.
	The KPAD is a 4x4 keypad compatible with the DIO header on the TS-81XX.
LCD-LED	LCD-LED description
PS-12VDC-REG-2P2	PS-12VDC-REG-2P2 description
PS-5VDC-REG-2P1	PS-5VDC-REG-2P1 description
TS-ENC720-7200	TS-ENC720-7200 description
	The WIFI-N-USB is an ASUS 802.11N adapter. See the WIFI-N-USB page for more details.
KPAD-LCD	KPAD-LCD description

2.2 Booting up the board

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-8100-4200 accepts 5-28VDC input connected to the two terminal blocks.



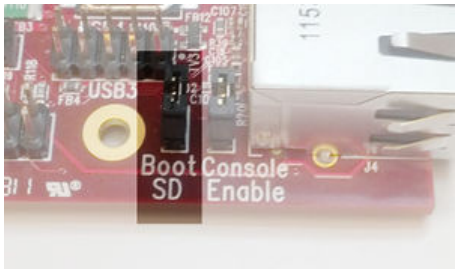
A typical power supply for just the TS-8100-4200 will allow around 1A, but a larger power supply may be needed depending on your peripherals.

Once you have applied power to your baseboard you should look for console output. Creating this connection is described more in the next chapter, but the first output is from the bootrom:

```
>> TS-BOOTROM - built Mar  9 2011 09:59:23
>> Copyright (c) 2011, Technologic Systems
>> Booting from SD card...
.
.
.
```

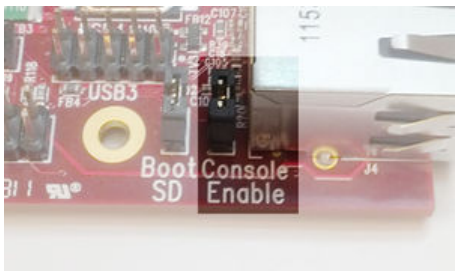
The 3 dots after indicate steps of the booting procedure. The first dot means the MBR was copied into memory and executed. The next two dots indicate that the MBR executed and the kernel and initrd were found and copied to memory.

The "Booting from XXXX...." message will indicate your boot media. On the TS-8100 this is controlled by the "SD Boot" jumper hear the ethernet connectors:



2.3 Get a Console

The TS-8100 console is an RS232 UART at 115200 baud, 8n1 (8 data bits 1 stop bit), and no flow control. On the TS-8100 you need to set the "Console Enable" jumper as pictured:



This will bring the console UART to both the DB9 Port, and the COM1 Header.

Note:	If DIO_9 is held low during boot until the red LED comes on (around 5 seconds), console will be redirected to XUART 0. On most baseboards where this is applicable, DIO_9 is an exposed button.

Console from Linux

There are many serial terminal applications for Linux, but 3 common implementations 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
```

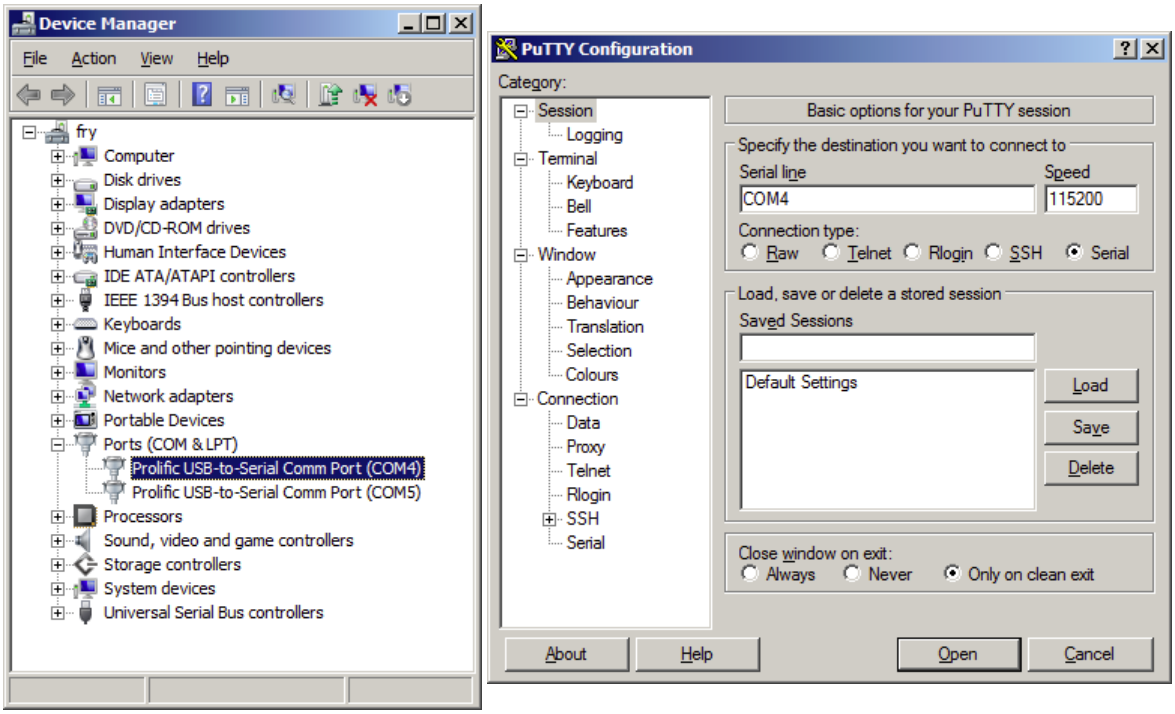
- Navigate to 'serial port setup'
- Type "a" and change location of serial device to '/dev/ttyUSB0' then hit "enter"
- If needed, modify the settings to match this and hit "esc" when done:

```
  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 (http://www.chiark.greenend.org.uk/~sgtatham/putty/) . Open up Device Manager to determine your console port. See the putty configuration image for more details.



2.4 Initrd / Busybox

When the board first boots you should see output similar to this:

```
>> TS-BOOTROM - built Mar  9 2011 09:59:23
>> Copyright (c) 2011, Technologic Systems
>> Booting from SD card...
.
.
.
Uncompressing Linux... done, booting the kernel.
>> Booted from: SD card                 Booted in: 1.40 seconds
>> SBC Model number: TS-4200            SBC Sub-model number: 0
>> CPU clock rate: 396 MHz              RAM size: 128MB
>> NAND Flash size: 256MB               NAND Flash Type: 0xdc20 (STMicro)
>> MAC number: 00:D0:69:44:3E:9B        SBC FPGA Version: 4
>> Temperature Sensor: 31.500 degC      MODE1 bootstrap: OFF
>> RTC present: YES                     Date and Time: Jan  1 2001 00:01:49
>> Base board type: TS-8200/Unknown     Base board FPGA Version: not present
>> MODE2 bootstrap: ON                  SD card size: 1886MB
>> Offboard SPI flash type: not present Offboard SPI flash size: not present
>> XUARTs detected: 0                   CAN present: NO
>> Linux kernel version: 2.6.36.2       Linux kernel date: Oct 7 2011
>> Bootrom date: unknown                INITRD date: Oct  7 2011
>> ts4200ctl date: Sep 27 2011          sdctl date: not present
>> canctl date: not present             nandctl date: Sep 26 2011
>> spiflashctl date: not present        xuartctl date: not present
>> dioctl date: not present             spictl date: not present
>> dmxctl date: not present             busybox date: Sep 26 2011 (v1.18.3)
>> ts4200.subr date: Sep 29 2011        daqctl date: not present
>> linuxrc date: Sep 27 2011            rootfs date: Oct  7 2011
>> MBR date: Sept 20 2011

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 the command

```
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. Whether or not a Debian application will work in fastboot needs to be judged per application. 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. Through any connection method you can relink the linuxrc to change it to boot by default to Debian.

The initrd has these boot scripts available:

Script	Function
linuxrc-fastboot (default)	Boots immediately to a shell in ramdisk. This will mount whichever boot medium you have selected to /mnt/root/. When you type 'exit', it will boot to that medium.
linuxrc-nandmount	Same as the linuxrc-fastboot script, but will mount and boot the debian partition from NAND.
linuxrc-sdmount	Same as the linuxrc-fastboot script, but will mount and boot the debian partition from SD.
linuxrc-sdroot	Boots immediately to the Debian stored on either SD or NAND depending on which device you have currently selected.
linuxrc-sdroot-readonly	Same as linuxrc-sdroot, except it will mount the Debian partition read only while creating a unionfs with a ramdisk. Changes will only happen in memory and not on disk.
linuxrc-usbroot	Mounts the first partition of the first detected USB mass storage device and boots there.

Note: Keep in mind the boot medium is selected by the pinout on your baseboard, not through software.

For example, to set the linuxrc to boot immediately to Debian on the SD you would run this:

```
rm linuxrc; ln -s /linuxrc-sdroot /linuxrc; save
```

The small default initrd is only 2Mbyte but there is space for approximately 300 Kbyte of additional user applications. The binaries on the initrd are dynamically linked against embedded Linux's "uclibc" library instead of the more common Linux C library "glibc". "uclibc" is a smaller version of the standard C library optimized for embedded systems and requires a different set of GCC compiler tools which are available here (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4200-linux/cross-toolchains/arm-uclibc-4.3.5-buildroot.tar.bz2) .

The compiled instance of busybox includes several internal commands listed below:

```
BusyBox v1.18.3 (2011-09-22 17:52:49 MST) multi-call binary.
Copyright (C) 1998-2009 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: busybox --list[-full]
   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:
        [, [[, ar, ash, basename, cat, chat, chgrp, chmod, chown, chroot, chrt,
        cmp, cp, cpio, cttyhack, cut, date, dc, dd, depmod, devmem, df,
        dirname, dmesg, dnsdomainname, du, echo, egrep, env, expr, false,
        fdisk, fgrep, find, free, grep, gunzip, gzip, halt, head, hostname,
        hush, hwclock, ifconfig, insmod, ipcrm, ipcs, kill, killall, ln, login,
        ls, lsmod, lsusb, md5sum, mdev, microcom, mkdir, mkfifo, mknod,
        modinfo, modprobe, more, mount, mv, netstat, nohup, ping, pivot_root,
        poweroff, printf, ps, pwd, rdate, reboot, rm, rmdir, rmmod, route, rx,
        sed, seq, setconsole, setsid, sh, sha1sum, sha256sum, sha512sum, sleep,
        stty, sync, sysctl, tail, tar, tee, telnetd, test, tftp, time, top,
        touch, tr, true, udhcpc, umount, uname, unxz, unzip, uptime, usleep,
        uudecode, uuencode, vi, watch, wget, xargs, xz, xzcat, yes, zcat
```

Also on the initrd are the TS specific applications: nandctl and ts4200ctl. We also provide the ts4200.subr which provides the following functions:

```
eth_off()
eth_on()
attime()
usb_off()
usb_on()
usbload()
sdload()
nandsave()
sdsave()
save()
sd2nand()
nand2sd()
setdiopin()
getdiopin()
getadc()
pc104on()
pc104off()
setupclk()
tshelp()
```

By default, linuxrc will not insert the necessary modules into the kernel to mount and use USB devices within the initrd/busybox environment if there is no USB device present upon bootup (USB support is enabled by default within the Debian environment). The quickest way to get a USB device (like a USB thumb drive) to mount in the initrd/busybox environment is to ensure that it is plugged in before the SBC is powered up. In order to get hot-swappable USB devices regardless of device presence at bootup time, you must "modprobe" the necessary modules. This has been done for you in the ts4200.subr file with the usbload() function.

3 Debian Configuration

For development it is recommended to go to Debian on the SD card 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. Out of the box the Debian distribution does not have a custom username/password set. You can use "root" as the username with no password to get access to the system. Keep in mind services such as ssh require a password set before they allow connection.

3.1 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
dhclient eth0
# Or if you're on a baseboard with a second ethernet port, you can use that as:
dhclient eth1
# You can configure all ethernet ports for a dhcp response with
dhclient
```

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
```

Note: During Debian's startup it will assign the interfaces eth0 and eth1 to the detected mac addresses in /etc/udev/rules.d/70-persistent-net.rules. If the system is imaged while this file exists it will assign the new interfaces as eth1 and eth2. This file is generated automatically on startup, and should be removed before your first software image is created. The initrd network configuration does not use this file.

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 (http://wiki.debian.org/Network) .

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.

3.2 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.

```
apt-get update
```

For example, lets say you wanted to install openjdk for Java support. You can use the apt-cache command to search the local cache of Debian's packages.

```
<user>@<hostname>:~# apt-cache search openjdk
icedtea-6-jre-cacao - Alternative JVM for OpenJDK, using Cacao
icedtea6-plugin - web browser plugin based on OpenJDK and IcedTea to execute Java applets
openjdk-6-dbg - Java runtime based on OpenJDK (debugging symbols)
openjdk-6-demo - Java runtime based on OpenJDK (demos and examples)
openjdk-6-doc - OpenJDK Development Kit (JDK) documentation
openjdk-6-jdk - OpenJDK Development Kit (JDK)
openjdk-6-jre-headless - OpenJDK Java runtime, using Hotspot Zero (headless)
openjdk-6-jre-lib - OpenJDK Java runtime (architecture independent libraries)
openjdk-6-jre-zero - Alternative JVM for OpenJDK, using Zero/Shark
openjdk-6-jre - OpenJDK Java runtime, using Hotspot Zero
openjdk-6-source - OpenJDK Development Kit (JDK) source files
openoffice.org - office productivity suite
freemind - Java Program for creating and viewing Mindmaps
default-jdk-doc - Standard Java or Java compatible Development Kit (documentation)
default-jdk - Standard Java or Java compatible Development Kit
default-jre-headless - Standard Java or Java compatible Runtime (headless)
default-jre - Standard Java or Java compatible Runtime
```

In this case you will likely want openjdk-6-jre to provide a runtime environment, and possibly openjdk-6-jdk to provide a development environment. You can often find the names of packages from Debian's wiki (http://wiki.debian.org/) 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 openjdk-6-jre
# You can also chain packages to be installed
apt-get install openjdk-6-jre nano vim mplayer
```

For more information on using apt-get refer to Debian's documentation here (http://wiki.debian.org/AptCLI) .

3.3 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 (http://www.chiark.greenend.org.uk/~sgtatham/putty/) .

Note: If your intended application does not have a DNS source on the target network, it can save login time to add "UseDNS no" in /etc/ssh /sshd_config.

3.4 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/ts4700ctl --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/ts4700ctl --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 this from 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.

4 Backup / Restore

If you are using a Windows workstation there is no support for writing directly to block devices. However, as long as one of your booting methods still can boot a kernel and the initrd you can rewrite everything by using a usb drive. This is also a good way to blast many stock boards when moving your product into production. You can find more information about this method with an example script here.

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

4.1 MicroSD 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.

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. Make sure you have any nbd devices unmounted before trying to restore new ones.

You can find the latest SD card image here (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4200-linux/binaries/ts-images/2gbsd-4200-latest.dd.bz2) . Make sure you decompress the image first before writing.

Note: Not all SD cards are created equally, over time they tend to shrink in size due to automatic retiring of bad blocks. All of Technologic System's images are 10% smaller than the target disc size. We STRONGLY recommend following that same practice on any mass-replicated images.

From Workstation

Backup

Entire SD card

```
dd if=/dev/mmcblk0 of=/path/to/backup.dd bs=32k && sync && sync
```

Note: Not all SD cards are created equally, over time they tend to shrink in size due to automatic retiring of bad blocks. All of Technologic System's images are 10% smaller than the target disc size. We STRONGLY recommend following that same practice on any mass-replicated images.

Kernel

```
dd if=/dev/mmcblk0p2 of=/path/to/zImage bs=32k && sync && sync
```

Initrd

```
dd if=/dev/mmcblk0p3 of=/path/to/initrd bs=32k && sync && sync
```

Restore

Entire SD card

```
dd if=/path/to/backup.dd of=/dev/mmcblk0 bs=32k && sync && sync
```

Kernel

```
dd if=/path/to/zImage bs=32k of=/dev/mmcblk0p2 && sync && sync
```

Initrd

```
dd if=/initrd bs=32k of=/dev/mmcblk0p3 && sync && sync
```

From SBC

Backup

Entire card

```
dd if=/dev/mmcblk0 of=/path/to/backup.dd && sync && sync
```

Kernel

```
dd if=/dev/mmcblk0p2 of=/path/to/backup.dd && sync && sync
```

Initrd

```
dd if=/dev/mmcblk0p3 of=/path/to/backup.dd && sync && sync
```

Restore

The entire card from SBC

```
dd if=/path/to/2gbsd-4200-latest.dd of=/dev/mmcblk0 && sync && sync
```

Kernel

```
dd if=/mnt/root/zImage of=/dev/mmcblk0p2 && sync && sync
```

Initrd

```
dd if=/mnt/root/initrd of=/dev/mmcblk0p3 && sync && sync
```

Expected Partition Layout

Partition	Contents
1	FAT32 (empty)
2	kernel binary (0xda)
3	initrd (0xda)
4	Debian root filesystem (EXT3)

4.2 XNAND

This needs to be done directly on the SBC. Please note that all NBD partitions from the NAND card must be dismounted before attempting to image the NAND on the SBC.

WARNING: Since there is no locking mechanism, it is not safe to run multiple copies of nandctl on this board. Make sure you unmount your filesystems and stop running nandctl if you prefer to backup/write data with that. These examples use 'dd' which is safe to use while nandctl is running.

You can find the latest xnand image here (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4200-linux/binaries/ts-images/xnand-4200-latest.dd.gz) .

Backup

Entire Image

```
# Compressed
dd if=/dev/nbd0 bs=131072 count=2048 | bzip2 > backup.dd.bz2
# or uncompressed
dd if=/dev/nbd0 bs=131072 count=2048 of=backup.dd
```

Kernel

```
dd if=/dev/nbd1 bs=512 count=5119 of=/path/to/backup/zImage
```

Initrd

```
dd if=/dev/nbd2 bs=512 count=5120 of=/path/to/backup/initrd
```

Restore

Entire Image

```
dd if=xnand-4200-latest.dd bs=131072 count=2048 of=/dev/nbd0 conv=fsync
```

Kernel

```
dd of=/dev/nbd1 bs=512 count=5119 if=/path/to/backup/zImage conv=fsync
```

Initrd

```
dd of=/dev/nbd2 bs=512 count=5120 if=/path/to/backup/initrd conv=fsync
```

Expected Partition Layout

Partition	Contents
1	kernel binary (0xda)
2	initrd (0xda)
3	Debian root filesystem (EXT3)

5 Software Development

Most of our examples are going to be in C, but Debian will include support for many more programming languages. Including (but not limited to) C++, PERL, PHP, SH, Java, BASIC, TCL, and Python. Most of the functionality from our software examples can be done from using system calls to run our userspace utilities. For higher performance, you will need to either use C/C++ or find functionally equivalent ways to perform the same actions as our examples. Our userspace applications are all designed to go through a TCP interface. By looking at the source for these applications, you can learn our protocol for communicating with the hardware interfaces in any language.

The most common method of development is directly on the SBC. Since debian has space available on the SD card, we include the build-essentials package which comes with everything you need to do C/C++ development on the board.

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 (http://vimdoc.sourceforge.net/) .

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 (http://www.gnu.org/s/emacs/#Manuals) .

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 (http://www.nano-editor.org/docs.php) .

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 (http://gcc.gnu.org/onlinedocs/) . You can find a simple hello world tutorial for c++ with gcc here (http://www.network-theory.co.uk/docs/gccintro/gccintro_54.html) .

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 (http://www.gnu.org/software/make/manual/make.html#Introduction) .

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 (http://www.gnu.org/s/hello/manual/automake /Introduction.html#Introduction) .

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 (http://www.cmake.org/cmake/help/cmake_tutorial.html) .

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 (http://www.gnu.org/s/gdb/documentation/) .

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 (http://www.linuxmanpages.com/man1/strace.1.php) .

Ltrace will do the same thing with any generic library. You can find the manual page here (http://linux.die.net/man/1/ltrace) .

5.1 Cross Compiling

While you can develop entirely on the board itself, if you prefer to develop from another x86 compatible Linux system we have a cross compiler available. For this board you will want to use this toolchain (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4700-linux/cross-toolchains /arm-2008q3.tar.gz) . To compile your application, you only need to use the version of GCC in the cross toolchain instead of the version supplied with your distribution. The resulting binary will be for ARM.

```
[user@localhost]$ /opt/arm-2008q3/bin/arm-none-linux-gnueabi-gcc hello.c -o hello
[user@localhost]$ file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.14, not stripped
```

This is one of the simplest examples. If you want to work with a project, you will typically create a makefile. You can read more about makefiles

here (http://www.gnu.org/software/make/manual/make.html#Introduction) . Another common requirement is linking to third party libraries provided by Debian on the board. There is no exact set of steps you can take for every project, but the process will be very much the same. Find the headers, and the libraries. Sometimes you have to also copy over their binaries. In this example, I will link to sqlite from Debian (which will also work in the Ubuntu image).

Install the sqlite library and header on the board:

```
apt-get update && apt-get install -y libsqlite3-0 libsqlite-dev
```

This will fetch the binaries from the internet and install them. You can list the installed files with dpkg:

```
dpkg -L libsqlite3-0 libsqlite3-dev
```

The interesting files from this output will be the .so files, and the .h files. In this case you will need to copy these files to your project directory.

I have a sample example with libsqlite3 below. This is not intended to provide any functionality, but just call functions provided by sqlite.

```
#include <stdio.h>
#include <stdlib.h>
#include "sqlite3.h"

int main(int argc, char **argv)
{
        sqlite3 *db;
        char *zErrMsg = 0;
        int rc;
        printf("opening test.db\n");
        rc = sqlite3_open("test.db", &db);
        if(rc){
                fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
                sqlite3_close(db);
                exit(1);
        }
        if(rc!=SQLITE_OK){
                fprintf(stderr, "SQL error: %s\n", zErrMsg);
        }
        printf("closing test.db\n");
        sqlite3_close(db);
        return 0;
}
```

To build this with the external libraries I have the makefile below. This will have to be adjusted for your toolchain path. In this example I placed the headers in external/include and the library in external/lib.

```
CC=/opt/arm-2008q3/bin/arm-none-linux-gnueabi-gcc
CFLAGS=-c -Wall

all: sqlitetest

sqlitetest: sqlitetest.o
        $(CC) sqlitetest.o external/lib/libsqlite3.so.0 -o sqlitetest
sqlitetest.o: sqlitetest.c
        $(CC) $(CFLAGS) sqlitetest.c -Iexternal/include/

clean:
        rm -rf *o sqlitetest.o sqlitetest
```

You can then copy this directly to the board and execute it. There are many ways to transfer the compiled binaries to the board. Using a network filesystem such as sshfs or NFS will be the simplest to use if you are frequently updating data, but will require more setup. See your linux distribution's manual for more details. The simplest network method is using ssh/sftp. You can use winscp if from windows, or scp from linux. Make sure you set a password from debian for root. Otherwise the ssh server will deny connections. From winscp, enter the ip address of the SBC, the root username, and the password you have set. This will provide you with an explorer window you can drag files into.

For scp in linux, run:

```
#replace with your app name and your SBC IP address
scp sqlitetest root@192.168.0.50:/root/
```

After transferring the file to the board, execute it:

```
ts:~# ./sqlitetest
opening test.db
closing test.db
```

5.2 Kernel Compile Guide

WARNING:	Backup any important data on the board before continuing.

For adding new support to the kernel, or recompiling with more specific options you will need to have an X86 compatible linux host available that can handle the cross compiling. Compiling the kernel on the board is not supported or recommended. Before building the kernel you will need to install a few support libraries on your workstation:

Prerequisites

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.

Set up the Sources and Toolchain

```
# Download the cross compile toolchain (EABI)from Technologic Systems:
ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4200-linux/cross-toolchains/arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

# Extract to current working directory:
tar xvf arm-2008q3-72-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

# Download the Kernel sources
wget ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4200-linux/sources/linux-2.6.36-ts-src-latest.tar.gz

# Extract the Kernel Sources
gzip -dc linux-2.6.36-ts-src-latest.tar.gz | tar xf -

cd linux-2.6.36-release/
```

Configure the Sources

The kernel sources need a few variables to be exported.

```
# Set the CROSS_COMPILE variable to the absolute path to the toolchain.  This may be different for your system:
export CROSS_COMPILE=/opt/4200/arm-2008q3/bin/arm-none-linux-gnueabi-

# Normally, ARCH will be set based on your build hosts architecture.
export ARCH=arm
```

This sets up the default configuration that we ship with for the TS-4200

```
make ts4200_defconfig
```

This will bring up a graphical menu where you can edit the configuration to include support for new devices. For Example, to include support for a Prolific USB to serial adapter you would go to 'Device Drivers -> USB Support-> USB Serial Support' and then select 'USB Prolific 2303 Single Port Serial Driver'. Since the kernel only has a limited space try to build build drivers as modules whenever possible.

```
make menuconfig
```

Build the kernel Once you have it configured, start building. This usually takes a few minutes.

```
make && make modules
```

The new kernel will be at "arch/arm/boot" in a compressed format called zImage. The uncompressed version is simply called Image. With the default partitioning scheme it is REQUIRED that the kernel be < 2620416 bytes in size. If you need to shorten the size, try including your changes to the kernel as modules instead. Otherwise you will need to resize the kernel partition to account for the size difference.

Install the kernel Now that you have a kernel you can install it as you would our stock. See the #Backup / Restore section for examples on writing this to disk.

Install Modules Script to make directory and install modules

```
./build-module-bundles.sh
```

The build-module-bundles.sh script is meant to be run as a user (not root) and will create directories and install modules to them. The directory structure is created at /home/`whoami`/src/ts-4200/dist/<ts4200 kernel release number>/modules-install/. In that directory is initrd-modules/, lib/, and modules-<ts4200 kernel release number>.tgz.

initrd-modules/modules.tar.gz is a tarball that contains a minimal number of modules. This tarball needs to be copied to the initrd partition of the boot media. The boot process of the board will automatically un'tar this and insert any necessary modules.

Now the contents of lib/ can be copied to the root of the TS-4200. It is also possible to copy over modules-<ts4200 kernel release number>.tgz to the TS-4200 and unpack it in the root linux directory. You may want to remove any old modules on the board in /lib/modules/* before copying them to the board to rule out any incompatibilities. Once you boot up to the board, you need to run 'depmod' once to calculate module dependencies. You can then run 'modprobe' with the device drivers you've added. For the Prolific adapter added in the example, this would be:

```
modprobe pl2303
```

5.3 Getting Started with tsctl

First, download and install the latest version of tsctl as documented in the Getting Started Guide.

In the examples below you can follow along by typing the commands (the portion after the prompt) and expect to see the output below the prompt. Note that while the results should be similar, in some cases you might not see exactly the same results due to variations in execution.

Let's start the tsctl shell:

```
$ tsctl
tsctl 0.93-ts (Dec  7 2012 15:34:29)
Type "?" to get context-sensitive help.
tsctl>
```

Let's check that we really have a TS-4200.

```
tsctl> System ModelId
16896
tsctl>
```

That doesn't look like 4200! The reason is that the ModelId (and BaseBoardId) commands return 0x4200 (hexadecimal 4200), but the tsctl shell defaults to decimal output. (Note that the command line defaults to hexadecimal!)

We can change to hexadecimal output using the mode command. There is no output from this command.

```
tsctl> Mode Hex
tsctl>
```

Now let's try again.

```
tsctl> System ModelId
0x00004200
tsctl>
```

In addition to the base the output is represented in, you can also change the general format. The tsctl shell defaults to "NoAssign" mode in which only the input or inputs are printed, each on a separate line. The "Assign" mode (which is the default for the command line) prints a descriptive *name=value* pair for each value output.

```
tsctl> Mode Assign
tsctl>
```

Now let's re-run the previous command. If your version of tsctl is linked against libreadline, you can use the up-arrow twice to pull the *System ModelId* command back instead of typing it out.

```
tsctl> System ModelId
System_ModelId_0=0x00004200
tsctl>
```

Let's run the command again.

```
tsctl> System ModelId
System_ModelId_1=0x00004200
tsctl>
```

Notice that the name changed slightly. The first part of the name is the class, the second it the field name (which is also the function name in cases where the value is directly returned at the C API level), and the third is a number. This number is the index of the number of times this class function has been called during the current invocation of tsctl.

Let's switch back to NoAssign mode. Although the field names can be useful if you aren't familiar with what the output fields mean, mostly Assign mode is meant for evaluating by the shell to set variables.

```
tsctl> Mode NoAssign
tsctl>
```

Now let's see what base board we have. Your output will differ depending on what you actually have installed.

```
tsctl> System BaseBoardId
0x00008200
tsctl>
```

Let's switch back to decimal output.

```
tsctl> Mode Dec
tsctl>
```

Now, let's enter the *System* class onto the command stack so that we don't have to type it repeatedly.

```
tsctl> System
tsctl System>
```

Note that our first command, "System" was incomplete, which caused tsctl to push it onto the command stack. This can be used to reduce typing when repetitive sequences start with the same partial command. To pop an element off the stack in the shell, enter an empty line.

One feature of libtsctl is the System Map, which contains name/value pairs. The name is any string (8-bit Array) and the value is an integer. First, let's see how many entries are stored in the table.

```
tsctl System> MapLength
458
tsctl System>
```

The entries in the table are stored sorted by name (case-insensitively). Entries are used to store DIO names, enumerated values, attributes, and user-defined name/value pairs. If you want to see the entire table you can get each entry one at a time by index number, starting at 1:

```
tsctl System> MapGet 1
AIO_ADC
1
tsctl System>
```

The first line contains the name, while the second contains the value.

To get several entries at once let's first put the function name on the command stack

```
tsctl System> MapGet
tsctl System MapGet>
```

One feature of tsctl is the ability to separate commands by a semi-colon. In the shell (e.g. bash) this requires quoting the semi-colon; in the tsctl shell it does not. Let's get the next ten entries:

```
tsctl System MapGet> 2;3;4;5;6;7;8;9;10;11
AIO_DAC
2
attrib.8200.Wire.Connector.1.0
1
attrib.8200.Wire.Connector.1.1
1
attrib.8200.Wire.Connector.10.0
```

```
2
attrib.8200.Wire.Connector.12.0
2
attrib.8200.Wire.Connector.12.1
2
attrib.8200.Wire.Connector.13.0
2
attrib.8200.Wire.Connector.13.1
2
attrib.8200.Wire.Connector.14.0
2
attrib.8200.Wire.Connector.15.0
2
tsctl System MapGet>
```

What happened here is that each number got appended as the parameter to the *System MapGet* function.

Hit enter on an empty line to pop the *MapGet* function off the command stack.

```
tsctl System MapGet>
tsctl System>
```

Let's look at some of the attributes available. We can find the name of a connector by number as follows:

```
tsctl System> MapLookupPartial attrib.Connector.Name. 2
CN2_
tsctl System>
```

How many connectors are there?

```
tsctl System> MapLookup attrib.Connector.Count
2
tsctl System>
```

Depending on your system, you may have more than this! How many pins does connector 1 have?

```
tsctl System> MapLookup attrib.Connector.1.Pins
100
tsctl System>
```

Most boards have a green and red LEDs, but the DIO number differs from board to board. Let's see what DIO numbers they are on this board. If the lookup fails, a negative value will be returned.

```
tsctl System> MapLookup;GREEN_LED;RED_LED;;
128
129
tsctl System>
```

Note that we used two semi-colons with nothing between them to pop the MapLookup function back off the stack.

What connector is the GREEN_LED on? We can determine this by searching the connector attribute for a value corresponding to the DIO number of GREEN_LED. In the above case, that value is 128. However we can also use GREEN_LED, as the tsctl text interface will automatically translate it to the correct value:

```
tsctl System> MapLookupPartial attrib.Connector. GREEN_LED
2.8
tsctl System>
```

Note that in a few rare cases the above lookup will conflict with another attribute and may not work. This is because MapLookupPartial looks for a name starting with the specified string, having the specified value. If we specified a value of "100" we might match "attrib.Connector.1.Pins", "attrib.Connector.2.Pins", or "attrib.Connector.1.8" as these all have a value of 100.

The interpretation of *2.8* is "Connector 2, Pin 8". This is also known as "CN2_8", by combining the name of the connector with the pin number on that connector.

```
tsctl System> MapLookup CN2_8
128
tsctl System>
```

NOTE: For widest cross-platform compatibility it is recommended to perform lookups based on connectors, rather than board specific DIO names.

How many DIO are on the board?

```
tsctl System> MapLookup attrib.DIO.Count
130
tsctl System>
```

If you have a peripheral board such as a baseboard or PC-104 board with supported DIO, you will see a higher number than this. Note that this number counts all raw, internally addressable DIO, regardless of whether or not they are brought out to pins. As such the actual number of usable DIO will frequently be lower than the number contained in this attribute.

Let's pop the System class from the command stack.

```
tsctl System>
tsctl>
```

What revision of the FPGA is on the board?

```
tsctl> System FPGARevision
4
tsctl>
```

We can read the I2C (TWI) temp sensor on the TS-4200 with tsctl. First let's switch to hexadecimal output for Arrays of bytes.

```
tsctl> Mode AHex
tsctl>
```

Next, we need to make sure that the pins that are used for TWI are correctly set up, as they are frequently multi-function pins. The easiest way to make sure the pins are set correctly is to lock the function you are going to use. As part of locking the pin will be initialized to the correct function. For some boards (notably the TS-4800) the TWI must always be locked during use as it uses an underlying operating system file to perform its functionaity.

```
tsctl> TWI Lock 0 0
1
tsctl>
```

Now verify that the temperature sensor is present at device address 0x49.

```
tsctl> TWI Read 0x49 1 7 2
TWISuccess
0x01:0x90
tsctl>
```

If the bytes read back are not 0x01:0x90 (the first value returned is the result code), then the temperature sensor is not present, or there is another problem with the TWI bus. Assuming we get the correct response back, we can next send the commands to start an aquisition, and read back the raw temperature data from the sensor.

```
tsctl> TWI Write;0x49 1 1 0x40:0x0;0x49 0 0 0x40:0x0;;Read 0x49 1 0 2;;
TWISuccess
TWISuccess
TWISuccess
0x12:0x60
tsctl>
```

The value returned can be converted to a temperature as follows:

tempC = (byte[0] * 256 + byte[1]) / 128
tempC = (0x12 * 256 + 0x60) / 128
tempC = 36.75C

Note: The ts8160ctl sample application provides the above TWI temp sensor reading functionality using the -t option.

Be sure to unlock any resource when you are done. Best practice is to hold a lock for the minimum amount of time necessary.

```
tsctl> TWI Unlock 0 0
1
tsctl>
```

There are also several timing based functions you can use. For instance, you can delay for a specific amount of time. You should see a delay of approximately 1 second (1,000,000 microseconds) when running the command below:

```
tsctl> Time;Delay 1000000
tsctl Time>
```

Note that the Delay function does not return a value. If you want to see the actual number of microseconds delayed, use the Wait function instead:

```
tsctl Time> Wait 1000000;;
1011557
tsctl>
```

The only difference between Wait and Delay is that the former returns the number of microseconds actually spend waiting, and the latter returns no value. This is an important nuance in the TCP classes, as in certain modes functions that return no data are not called until subsequent functions that do return data are called. This allows for things such as commanding a relay to cycle power on the board issuing the command (so long as it isn't the board interpreting the command), since otherwise the command to restore power would not be sent before power was cut.

Short delay times are generally only useful when using direct access from C. Otherwise, the overhead of parsing the command, sending it across TCP, and interpreting it on the server will be significant in comparison to the amount of time to delay.

6 Features

6.1 CPU

The TS-4200 features a AT91SAM9G20 400MHz ARM9. For more information on the processor and it's integrated peripherals, refer to the CPU manual (http://www.embeddedarm.com/documentation/third-party/atmel_at91sam9g20_manual_may2010.pdf) .

6.2 MicroSD Card Interface

The SD interface supports both MicroSD and MicroSDHC cards. There is a linux driver provided in the default TS-4200 kernel that allows you to access the SD card block device as /dev/mmcblk0, or /dev/mmcblk0p# for accessing a specific partition. By default the TS-4700 will use this layout:

Device	Contents
/dev/mmcblk0p1	Fat32 partition.
/dev/mmcblk0p2	Linux kernel.
/dev/mmcblk0p3	Initrd/Fastboot
/dev/mmcblk0p4	Debian Filesystem

The FAT32 partition isn't necessarily required, but unless Windows can recognize one of the partitions it will ask the user if they want to formt the disk. If you remove this partition you will need to modify the linuxrc scripts accordingly

6.3 XNAND

The XNAND is a custom block device abstraction which is designed to vastly increase the reliability of NAND access. This board includes a 512MB flash chip, but the XNAND algorithm will limit this to a usable 256MB from redundancy. The software layer to access the XNAND is implemented in userspace in conjunction with NBD (network block device). You may want to refer to the nandctl page which will show more advanced usage, but by default the linuxrc script will mount the sd card with the following layout:

```
/dev/nbd0 - whole disk device of XNAND drive
/dev/nbd1 - 1st partition (kernel partition)
/dev/nbd2 - 2nd partition (EXT2 initrd)
/dev/nbd3 - 3rd partition (~252MByte mini Debian EXT3 filesystem)
/dev/nbd4 - 4th partition (unused)
```

Note: NBD devices do not report size correctly. If you are formatting a partition or using dd you will need to specify the size.

6.4 Interrupts

We include a userspace IRQ patch in our kernels. This allows you to receive interrupts from your applications where you would normally have to write a kernel driver. This works by creating a file for each interrupt in '/proc/irq/<irqnum>/irq'. The new irq file allows you to block on a read on the file until an interrupt fires.

The original patch is documented here (http://lwn.net/Articles/127293/) .

This example below will work with any of our TS-Socket boards running Linux. This opens the IRQ number specified in the first argument and prints when it detects an IRQ.

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/stat.h>
#include <unistd.h>

int main(int argc, char **argv)
{
        char proc_irq[32];
        int ret, irqfd = 0;
        int buf; // Holds irq junk data
        fd_set fds;

        if(argc < 2) {
                printf("Usage: %s <irq number>\n", argv[0]);
                return 1;
        }

        snprintf(proc_irq, sizeof(proc_irq), "/proc/irq/%d/irq", atoi(argv[1]));
        irqfd = open(proc_irq, O_RDONLY| O_NONBLOCK, S_IREAD);

        if(irqfd == -1) {
                printf("Could not open IRQ %s\n", argv[1]);
                return 1;
        }

        while(1) {
                FD_SET(irqfd, &fds); //add the fd to the set
                // See if the IRQ has any data available to read
                ret = select(irqfd + 1, &fds, NULL, NULL, NULL);

                if(FD_ISSET(irqfd, &fds))
                {
                        FD_CLR(irqfd, &fds);  //Remove the filedes from set
                        printf("IRQ detected\n");

                        // Clear the junk data in the IRQ file
                        read(irqfd, &buf, sizeof(buf));
                }

                //Sleep, or do any other processing here
                usleep(10000);
        }

        return 0;
}
```

The TS-4200 has 4 IRQs that can be used by external devices. See page 30 of the CPU manual for a complete listing of all of the available IRQs.

IRQ #	Interrupt Condition	Name	Socket Location
31	Configurable [1]	PC14_IRQ2	CN2-91
192	High Level	IRQ5/DIO_00	CN1-93
193	High Level	IRQ6/DIO_01	CN1-91
194	High Level	IRQ7/DIO_02	CN1-89

1. ↑ See the CPU manual for details on customizing this IRQ. This supports operating as a posedge or negedge interrupt

Note: IRQs 192-194 require 'pc104on' to be run to MUX these IRQs. See the #PC104 chapter.

IRQ #	Interrupt Condition	Name	Location
192	High Level	IRQ5/DIO_00	#PC104 Header
193	High Level	IRQ6/DIO_01	#PC104 Header
194	High Level	IRQ7/DIO_02	#PC104 Header

Note: IRQs 192-194 require 'pc104on' to be run to MUX these IRQs. See the #PC104 chapter.

6.5 LEDs

On all of our baseboards we include 2 indicator LEDs which are under software control. You can manipulate these using "ts4200ctl --greenledon --redledon" or "ts4200ctl --greenledoff --redledoff". The LEDs have 4 behaviors from default software.

Green Behavior	Red behavior	Meaning
On	On for about .5s, off for 5s, on for 1s, off	After the red LED stays off the system has booted and is running.
On	Off	System is booted and running.
On	On for approximately 15s, then off	Once the system has booted the kernel and executed the startup script, it will check for a USB device and then determine if it is a mass storage device. This is used for updates/blasting through USB. Once it determines this is not a mass storage device the red LED will turn back off.
On for 10s, off for 100ms, and repeating	Turns on after Green turns off for 300ms, and then turns off for 10s	The watchdog is continuously resetting the board. This happens when the system cannot find a valid boot device, or the watchdog is otherwise not being fed. This is normally fed by ts4200ctl once a valid boot media has started. See the #Watchdog section for more details.
Off	Off	The FPGA is not able to start. Typically either the board is not being supplied with enough voltage, or the FPGA has been otherwise damaged. If using an off the shelf baseboard and a stable 5V is being provided and the supply is capable of providing at least 1A and 5V to the macrocontroller, an RMA (http://www.embeddedarm.com/about/rma.php) is suggested.

6.6 Ethernet Port

The Atmel Processor implements a 10/100 ethernet controller with support built into the Linux kernel. You can use standard Linux utilities such as ifconfig/ip to control this interface. See the #Configuring the Network section for more details. For the specifics of this network controller see the CPU manual (http://www.embeddedarm.com/documentation/third-party/atmel_at91sam9g20_manual_may2010.pdf) .

6.7 Baseboard ID

The TS-8100 has a baseboard ID of 7 which can be used to detect the baseboard in code. Implementations should refer to "ts4700ctl --baseboard" which detect the baseboard ID.

6.8 CAN

The TS-4200 does not support CAN.

6.9 USB

6.9.1 USB Host

The USB host port is a standard USB 2.0 at 480Mbps. The Linux kernel provides most of the USB support, and some devices may require a kernel recompile. Common devices such as keyboards, mice, wifi, and ethernet should mostly work out of the box.

The libusb (http://www.libusb.org/) project can also be used to communicate directly with USB peripherals from userspace.

6.10 TWI

TS-4200 TWI

6.11 SPI

The SPI controller is provided by the Atmel CPU. Documentation for this can be found in the CPU manual (http://www.embeddedarm.com /documentation/third-party/atmel_at91sam9g20_manual_may2010.pdf) in section 30 (page 391).

6.12 FPGA

The TS-4200 features a Low power Actel FPGA clocked by the CPU only when needed. This FPGA is not end user programmable, but you can use the MUXBUS to allow communication to an offboard FPGA. The TS-4200 provides access to the FPGA in an 8 bit region and a 16 bit region. The 8 bit base address is 0x10000000. The 16 bit base address is 0x30000000. All registers inside the TS-4200 FPGA are 16 bit registers and should be accessed via the 16 bit space. The 8 bit space is only needed for off-board 8 bit devices on the MUXBUS. To access hardware cores in the FPGA, add the offset in the table below to the base address.

This table shows our top level decode for the FPGA. Use the 16 or 8 bit address with these offsets to talk to our various FPGA cores.

Offset	Usage
0x0	Syscon registers
0x080	ADC registers (for off-board ADC)
0x100	NAND flash registers
0x200	NVRAM control registers
0x400	1KB MUXBUS space
0x800	2KB Direct blockram access (For NVRAM transfers)

6.13 Syscon

The registers listed below are all 16 bit registers and must be accessed with 16 bit reads and writes. This register block appears at base address 0x30000000.

Offset	Bits	Usage
0x0	15:0	Model ID: Reads 0x4200
0x2	15	Green LED (1 = on)
	14	Red LED (1 = on)
	13:8	Reserved
	7:4	Board Sub-model: reads 0x0 on standard unit
	3:0	FPGA revision
0x4	15:0	DIO direction for DIO 15(MSB)-0(LSB)
0x6	15:0	DIO output data for DIO 15(MSB)-0(LSB)
0x8	15:0	DIO input data for DIO 15(MSB)-0(LSB)
0xa	15:12	Reserved
	11	UART0 9 bit mode (so that TX_EN works correctly)
	10	1 = use DIO12 for uart0 TX_EN instead of DIO
	9	1 = use 512Hz for WDT instead of 200Hz
	8	1 = auto reboot when DIO 9 is driven low (enable reset button)
	7:6	Scratch Register (used by bootrom)
	5:4	Mode strapping inputs Mode2 and Mode1
	3	Reserved
	2	1 = Enable power to Ethernet PHY
	1	1 = Enable power to SD card
	0	Offboard reset signal (1 = reset)
0xc	15:0	32-bit 1MHz free running counter (16 LSBs)
0xe	15:0	32-bit 1MHz free running counter (16 MSBs)
0x10	15:0	Watchdog feed register
0x12	15:0	DIO direction for DIO 31(MSB) - 16(LSB)
0x14	15:0	DIO output data for DIO 31(MSB) - 16(LSB)
0x16	15:0	DIO input data for DIO 31(MSB) - 16(LSB)
0x18	15:0	Hardware RNG (16 LSB)
0x1a	15:0	Hardware RNG (16 MSB)
0x1c	15	Embedded FlashROM CLK
	14:8	Embedded FlashROM Address
	7:0	Embedded FlashROM Data Byte
0x1e	15:0	UART0 baud rate divisor (for UART0 TX_EN)
0x20	15:0	MUXBUS configuration register
0x22	15:0	IRQ register
0x24	15:0	IRQ mask register

The born-on date of the unit and the MAC address can be read by from the FPGA FlashROM. See ts4200ctl source code for sample usage.

6.14 ADC Core

The ADC core supports the MCP3428 which is an affordable 16-bit ADC board. This is available on several of the TS-Socket baseboards.

This core assumes the standard circuit which allows 2 differential channels and 4 single-ended channels. The single-ended channels are chosen using analog muxes controlled by the AN_SEL line. Since different base boards use a different pin for AN_SEL, a register is also provided to select the correct line. Channels 1 and 2 are differential channels with a range of -2.048V to +2.048V. Channels 3-6 are 0 to 10.24V.

Offset	Bits	Access	Usage
0x0	15-8	Read Only	Core ID register (0xad)
	7-6	Read Only	Reserved
	5-4	Read/Write	Analog Select 0 = Do not use an AN_SEL 1 = Use CN1 pin 77 for AN_SEL (TS-81X0) 2 = Use CN1 pin 74 for AN_SEL (TS-8390) 3 = Reserved
	3-2	Read/Write	Speed 0 = 240Hz, 12 bit resolution 1 = 60Hz, 14 bit resolution 2 = 15Hz, 16 bit resolution 3 = Reserved
	1-0	Read/Write	Programmable Gain 0 = No gain 1 = 2x gain 2 = 4x gain 3 = 8x gain
0x2	15-0	Read/Write	Channel Mask
0x4	15-0	Read Only	Channel 1 most recent conversion value
0x6	15-0	Read Only	Channel 2 most recent conversion value
0x8	15-0	Read Only	Channel 3 most recent conversion value
0xa	15-0	Read Only	Channel 4 most recent conversion value
0xc	15-0	Read Only	Channel 5 most recent conversion value
0xe	15-0	Read Only	Channel 6 most recent conversion value

The channel mask register controls which channels are enabled. Bits 0-5 enable channels 1-6 respectively. If a given channel is not enabled, (enable bit == 0) it will not be sampled and its conversion value register will contain an obsolete and meaningless value. The more channels that are enabled, the lower the sampling speed on each channel.

This example will sample the current value of all ADCs and output them in millivolts.

```
#include <stdio.h>
#include <stdint.h>
#include <fcntl.h>
#include <sys/mman.h>

volatile uint16_t *fpga = 0;
uint16_t peek16(uint16_t addr)
{
        uint16_t value;

        if(fpga == 0) {
                int mem = open("/dev/mem", O_RDWR|O_SYNC);
                fpga = mmap(0,
                        getpagesize(),
                        PROT_READ|PROT_WRITE,
                        MAP_SHARED,
                        mem,
                        0x30000000);
        }

        return fpga[addr/2];
}

void poke16(uint16_t addr, uint16_t value)
{
        if(fpga == 0) {
                int mem = open("/dev/mem", O_RDWR|O_SYNC);
                fpga = mmap(0,
                        getpagesize(),
                        PROT_READ|PROT_WRITE,
                        MAP_SHARED,
                        mem,
                        0x30000000);
        }
```

```
        fpga[addr/2] = value;
}

int main()
{
        int x, i;
        // Select TS-81XX, 15hz, 16-bit resolution with 0x gain
        poke16(0x80, 0x18);

        // Select TS-8390, 15hz, 16-bit resolution with 0x gain
        //poke16(0x80, 0x28);

        // unmaks to enable all 6 channels
        poke16(0x82, 0x3f);

        usleep(500000); // allow time for conversions
        for (i = 1; i <= 6; i++) {
                x = (signed short)peek16(0x82 + 2*i);
                if (i > 2) x = (x * 1006)/200;
                x = (x * 2048)/0x8000;
                printf("adc%d=%d\n", i, x);
        }

        return 0;
}
```

6.15 Watchdog

By default the watchdog is fed by ts4200ctl. This way if userspace, the kernel, or the FPGA communication has any issue the board will reboot. You can tailor this more specifically to your application by feeding the watchdog on your own criteria. The Watchdog feed register is write-only. Valid write values are:

Write Value	Effect
0	Feed watchdog for the next 0.5 sec
1	Feed watchdog for the next 2 sec
2	Feed watchdog for the next 16 sec
3	Disable watchdog

These watchdog timeout times assume that the 512Hz clock is being used as a watchdog clock. At power-up, the 200Hz clock is used, which means that the timeout times would actually be roughly 150% longer. The 512Hz clock signal is not turned on until a time is written to the RTC. The default linuxrc writes the RTC, which enables the 512Hz signal, and then switches the watchdog to the 512Hz signal.

If you would like to run your own watchdog you will need to kill ts4200ctl when switching to your own application. This below code is a minimalistic example for feeding the watchdog.

```
#include <stdio.h>
#include <stdint.h>
#include <fcntl.h>
#include <sys/mman.h>

int main()
{
        int mem;
        volatile uint16_t *syscon;

        mem = open("/dev/mem", O_RDWR|O_SYNC);
        syscon = mmap(0,
                      getpagesize(),
                      PROT_READ|PROT_WRITE,
                      MAP_SHARED,
                      mem,
                      0x30000000);

        for(;;) {
                // This feeds the watchdog for 16s.
                syscon[0x10/2] = 2;
                sleep(8);
        }

        return 0;
}
```

6.16 MUXBUS

The MUXBUS is the bus between the FPGA on the macrocontroller to communicate with the offboard CPLD. With the TS-8100, the CPLD includes all of the DIO and PC104. The MUXBUS config register in the #Syscon allows enabling and configuring the speed for this bus. The MUXBUS timing also influences the communication with PC104 peripherals.

For more advanced details on the MUXBUS, refer to the implementation details here.

Most applications can use one of two values

```
## Fast value
peekpoke 16 0x30000020 0x181

## Slow value (for older PC104 devices)
# peekpoke 0x30000020 0xf0ff
```

6.17 TS-81XX Register Map

All of these registers are intended for 16 bit access. You can find this range at 0x30000400-0x300007fe. You must first set the MUXBUS configuration register before accessing this range. For example, to read the board ID:

```
peekpoke 16 0x30000020 0x181 # Set MUXBUS configuration
peekpoke 16 0x30000400 # Read Board ID register (0x0)
```

Offset	Bits	Access	Description
0x0	15:0	Read Only	Board ID (0x8100)
0x2	3:0	Read Only	PLD revision
	7:4	Read/Write	Value to control PWM for LCD contrast
	8	Read/Write	TS-8100 USB Reset
	9	Read/Write	Controls ISA_RESET on the PC104 bus
	10	Read/Write	Enables a 14.3MHZ clock on the PC104 bus (B30) and the PLD (default 1)
	11	Read/Write	Enables the RS232 transceiver (default 1)
	12	Read/Write	Toggles 5V to the LCD header pin 1
	13	Read/Write	Enable CAN1 standby
	14	Read/Write	Enable CAN2 standby
	15	Read/Write	Enables the PWM output for the contrast value
0x4	7:0	Read/Write	DIO odd pins 15:1 output data
	13:8	Read/Write	PC104 header A21:A16 output data
	15:14	Read/Write	PC104 header B12:B11 output data
0x6	7:0	Read/Write	LCD pins 14:7 output data
	8	Read/Write	LCD Header pin 6 output data
	9	Read/Write	LCD Header pin 3 output data
	10	Read/Write	LCD Header pin 5 output data
	11	Read/Write	AVR MOSI
	12	Read/Write	AVR SCLK
	13	Read/Write	AVR RESET
	14:15	N/A	Reserved
0x8	7:0	Read/Write	DIO Header odd pins 15:1 data direction
	13:8	Read/Write	PC104 A21:A16 data direction
	15:14	Read/Write	PC104 B12:B11 data direction
0xa	7:0	Read/Write	LCD pins 14:7 data direction
	8	Read/Write	LCD Header pin 6 data direction
	9	Read/Write	LCD Header pin 3 data direction
	10	Read/Write	LCD Header pin 5 data direction
	15:11	N/A	Reserved
0xc	7:0	Read Only	DIO Header dd pins 15:1 input data
	13:8	Read Only	PC104 A21:A16 input data
	15:14	Read Only	PC104 B12:B11 input data
0xe	7:0	Read Only	LCD header pins 14-7 input data
	8	Read Only	LCD Write/Read (pin 6) input data
	9	Read Only	LCD Register Select (pin 3) input data
	10	Read Only	LCD Enable (pin 5) input data
	11	Read/Write	AVR MISO
	15:12	N/A	Reserved

6.18 DIO

The TS-8100 has up to 28 DIO available:

Signal	Location
8100_DIO_1	DIO header pin 1
8100_DIO_3	DIO header pin 3
8100_DIO_5	DIO header pin 5
8100_DIO_7	DIO header pin 7
8100_DIO_9	DIO header pin 9
8100_DIO_11	DIO header pin 11
8100_DIO_13	DIO header pin 13
8100_DIO_15	DIO header pin 15
8100_DIO_15	DIO header pin 15
8100_DIO_A16	PC104 header pin A16
8100_DIO_A17	PC104 header pin A17
8100_DIO_A18	PC104 header pin A18
8100_DIO_A19	PC104 header pin A19
8100_DIO_A20	PC104 header pin A20
8100_DIO_A21	PC104 header pin A21
8100_DIO_B11	PC104 header pin B11
8100_DIO_B12	PC104 header pin B12
8100_LCD_3	LCD header pin 3
8100_LCD_5	LCD header pin 5
8100_LCD_6	LCD header pin 6
8100_LCD_D1	LCD header pin 7
8100_LCD_D0	LCD header pin 8
8100_LCD_D3	LCD header pin 9
8100_LCD_D2	LCD header pin 10
8100_LCD_D5	LCD header pin 11
8100_LCD_D4	LCD header pin 12
8100_LCD_D7	LCD header pin 13
8100_LCD_D6	LCD header pin 14

These can be accessed using tsctl, or by interacting with the #TS-8100 Register Map.

6.19 UARTs

The TS-4200 brings out 7 UARTs at 0-3.3V levels, but typically the baseboard will include transceivers to bring this to RS232, RS485, or RS422. You can find resources for programming with serial ports in Linux here:

- Wikibook (http://en.wikibooks.org/wiki/Serial_Programming/Serial_Linux)
- Linux Documentation Project Serial Programming Guide (http://tldp.org/HOWTO/Serial-Programming-HOWTO/index.html)

When ttyS1 is connected to an RS485 transceiver the FPGA can be configured to autmoatically toggle TX Enable. This uses the FPGA to assert TXEN when transmitting and deassert when not. You can enable this by toggling a register in the #Syscon:

```
# Enables auto TX Enable in FPGA for ttyS1
peekpoke 16 0x3000000A 0x756
```

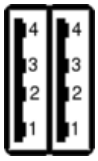
For UART4 (ttyS5) you will need to toggle the TX enable FPGA DIO14 manually.

Port	Type	RX (or 485 +)	TX (or 485 -)	Notes
ttyS0	RS232	DB9 pin 2, COM1 header pin 2	DB9 pin 3, COM1 header pin 3	Only with console enable jumper on
ttyS1	RS485	DB9 pin 1, COM1 header pin 1	DB9 pin 6, COM1 header pin 6	
ttyS2	RS232	DB9 pin 2, COM1 header pin 2	DB9 pin 3, COM1 header pin 3	Only with console enable jumper off
ttyS3	RS232	DB9 pin 8, COM1 header pin 8	DB9 pin 7, COM1 header pin 7	
ttyS4	RS232	COM2 header pin 2	COM2 header pin 3	
ttyS5	RS485	COM2 header pin 1	COM2 header pin 6	
ttyS6	RS232	COM3 header pin 2	COM3 header pin 3	CTS available on pin 8

7 External Interfaces

7.1 USB Port

The USB is available on two ports as a USB 2.0 host.



Header PIN	Name
1	USB_5V
2	USB -
3	USB +
4	GND

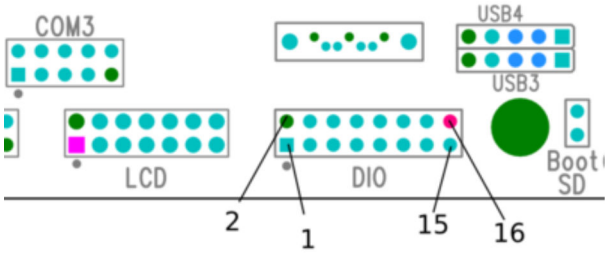
7.2 DIO header

The TS-8100 includes a 2x8 0.1" pitch header with 8 DIO, I2C, and SPI. Most DIO on this header are rated for 3.3V and are not tolerant of 5V IO. The only exception is SPI_MOSI which is 5V tolerant. The DIO on this baseboard can be accessed by manipulating the TS-8100 Register Map, or using tsctl.

Pinout

Pin	Name	Notes
1	8100_DIO_1	Pulled high by R124
2	Ground	
3	8100_DIO_3	Pulled high by R123
4	I2C_CLK	
5	8100_DIO_5	Pulled high by R122
6	SPI_CS	
7	8100_DIO_7	pulled high by R121
8	I2C_DAT	
9	8100_DIO_9	Pulled high by R120
10	SPI_MISO	
11	8100_DIO_11	Pulled high by R119
12	SPI_MOSI	
13	8100_DIO_13	Pulled high by R118
14	SPI_CLK	
15	8100_DIO_15	Pulled high by R117
16	CPU_3.3V	

Header



This header is designed to connect to the KPAD accessory which uses the odd DIO on this header to scan a 4x4 keypad. This example scans the

KPAD and prints out the pressed character.

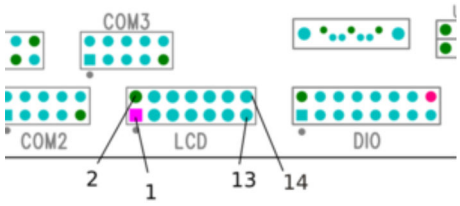
```
/* KPAD 4x4 keypad example code
 *
 * To compile, copy to the board and run:
 *      gcc kpad.c -o kpad  */

#include <stdio.h>
#include <stdint.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

// The mpeek/mpoke functions are specific to the TS-4200
volatile uint16_t *muxbus = 0;
uint16_t mpeek16(uint16_t addr)
{
        uint16_t value;

        if(muxbus == 0) {
                int mem = open("/dev/mem", O_RDWR|O_SYNC);
                muxbus = mmap(0,
                        getpagesize(),
                        PROT_READ|PROT_WRITE,
                        MAP_SHARED,
                        mem,
                        0x30000000);
        }

        return muxbus[(addr + 0x400)/2];
}

void mpoke16(uint16_t addr, uint16_t value)
{
        if(muxbus == 0) {
                int mem = open("/dev/mem", O_RDWR|O_SYNC);
                muxbus = mmap(0,
                        getpagesize(),
                        PROT_READ|PROT_WRITE,
                        MAP_SHARED,
                        mem,
                        0x30000000);
        }

        muxbus[(addr + 0x400)/2] = value;
}

int main()
{
    uint8_t ddr = 0xf0;
    uint8_t out = 0x0f;
    int row, col;

    char *keys[4][4] = {
        { "1", "2", "3", "UP" },
        { "4", "5", "6", "DOWN" },
        { "7", "8", "9", "2ND" },
        { "CLEAR", "0", "HELP", "ENTER" }
    };

    //set first 4 as outputs, last 4 as inputs
    mpoke16(0x8, ddr);
    mpoke16(0x4, out);

    while(1) {
        for(row = 0; row < 4; row++) {
            mpoke16(0x8, ddr | (1 << row));
            mpoke16(0x4, out | (1 << row));
            usleep(50000);
            uint16_t in = mpeek16(0xc);
            for(col = 4; col < 8; col++) {
                if(in & (1 << col)) {
                    // If we read it, sleep and read again to debounce
                    usleep(1000);
                    in = mpeek16(0xc);
                    if(in & (1 << col)) {
                        printf("%s\n", keys[row][col - 4]);
                        fflush(stdout);
                    }
                }
            }
        }
    }

    return 0;
}
```

7.3 LCD Header

The LCD header is designed around compatibility with the LCD-LED: Alphanumeric 2x24 LCD. These I/O are manipulated by accessing the register map. Connector CN8 is a 14 pin (2x7) 0.1" spacing header.

Pinout **Header**

Pin	Name
1	LCD_5V [1]
2	Ground
3	LCD_RS
4	LCD_BIAS
5	LCD_EN
6	LCD_WR#
7	LCD_D1
8	LCD_D0
9	LCD_D3
10	LCD_D2
11	LCD_D5
12	LCD_D4
13	LCD_D7
14	LCD_D6



1. ↑ Provides up to 1400mA

WARNING: LCD_D0 thru LCD_D7 are 5V tolerant. LCD_WR#, LCD_RS, and LCD_EN are not.

The LCD_5V pin can provide up to 1400mA, but this is a shared 5V rail and will depend on your power supply and what other devices are using that rail.

This example project allows you to pipe in data separated by newlines, or you can call the application with arguments to draw the two lines. For example:

```
./lcdmesg Technologic Systems
```

Will write this to the screen:


```
/* LCD 2x24 character example code
 *
 * To compile, copy to the board and run:
 *      gcc lcdmesg.c -o lcdmesg  */

#include<unistd.h>
#include<sys/types.h>
#include<sys/mman.h>
#include<stdio.h>
#include<fcntl.h>
#include<string.h>

#define FPGABASE 0x30000000
#define SYSCON (0x0a /sizeof(unsigned short))
#define ODR (0x406 /sizeof(unsigned short))
#define DDR (0x40A /sizeof(unsigned short))
#define IDR (0x40E /sizeof(unsigned short))

#define MUXBUS 0x20

#define LCD_WR 0x100
#define LCD_RS 0x200
#define LCD_EN 0x400

#define LCD_PWR 0x1000
```

```
// These delay values are calibrated for the TS-4200
#define SETUP   200
#define PULSE   400
#define HOLD    200

#define COUNTDOWN(x)    asm volatile ( \
  "1:\n"\
  "subs %1, %1, #1;\n"\
  "bne 1b;\n"\
  : "=r" ((x)) : "r" ((x)) \
);

volatile unsigned short *syscon, *odr, *ddr, *idr;

void command(unsigned int);
void writechars(unsigned char *);
unsigned int lcdwait(void);
void lcdinit(void);

/* This program takes lines from stdin and prints them to the
 * 2 line LCD connected to the TS-8100/TS-8160 LCD header.  e.g
 *
 *    echo "hello world" | lcdmesg
 *
 * It may need to be tweaked for different size displays
 */

int main(int argc, char **argv) {
        int i = 0;

        lcdinit();
        if (argc == 2) {
                writechars(argv[1]);
        }
        if (argc > 2) {
                writechars(argv[1]);
                lcdwait();
                command(0xa8); // set DDRAM addr to second row
                writechars(argv[2]);
        }
        if (argc >= 2) return 0;

        while(!feof(stdin)) {
                unsigned char buf[512];

                lcdwait();
                if (i) {
                        // XXX: this seek addr may be different for different
                        // LCD sizes!  -JO
                        command(0xa8); // set DDRAM addr to second row
                } else {
                        command(0x2); // return home
                }
                i = i ^ 0x1;
                if (fgets(buf, sizeof(buf), stdin) != NULL) {
                        unsigned int len;
                        buf[0x27] = 0;
                        len = strlen(buf);
                        if (buf[len - 1] == '\n') buf[len - 1] = 0;
                        writechars(buf);
                }
        }
        return 0;
}

void lcdinit(void) {
        int fd = open("/dev/mem", O_RDWR|O_SYNC);

        syscon = (unsigned short *)mmap(0, getpagesize(),
          PROT_READ|PROT_WRITE, MAP_SHARED, fd, FPGABASE);


        odr = &syscon[ODR];
        ddr = &syscon[DDR];
        idr = &syscon[IDR];
        syscon = &syscon[SYSCON];

        *syscon = *syscon | 0x1000;

        *ddr = *ddr & 0xFF00; // data lines to inputs
        *ddr = *ddr | LCD_WR | LCD_RS | LCD_EN; // control lines to outputs
        *odr &= ~(LCD_EN | LCD_RS);
        *odr |= LCD_WR;

        usleep(15000);
        command(0x38); // two rows, 5x7, 8 bit
        usleep(4100);
        command(0x38); // two rows, 5x7, 8 bit
        usleep(100);
        command(0x38); // two rows, 5x7, 8 bit
        command(0x6); // cursor increment mode
        lcdwait();
        command(0x1); // clear display
        lcdwait();
        command(0xc); // display on, blink off, cursor off
        lcdwait();
        command(0x2); // return home
}

unsigned int lcdwait(void) {
```

```
        int i, dat, tries = 0;
        unsigned short ctrl;

        *ddr &= 0xff00; // data lines to inputs
        ctrl = *odr;

        do {
                // step 1, apply RS & WR
                ctrl |= LCD_WR; // de-assert WR
                ctrl &= ~LCD_RS; // de-assert RS
                *odr = ctrl;

                // step 2, wait
                i = SETUP;
                COUNTDOWN(i);

                // step 3, assert EN
                ctrl |= LCD_EN;
                *odr = ctrl;

                // step 4, wait
                i = PULSE;
                COUNTDOWN(i);

                // step 5, de-assert EN, read result
                dat = *idr & 0xff;
                ctrl &= ~LCD_EN; // de-assert EN
                *odr = ctrl;

                // step 6, wait
                i = HOLD;
                COUNTDOWN(i);
        } while (dat & 0x80 && tries++ < 1000);
        return dat;
}

void command(unsigned int cmd) {
        int i;
        unsigned short ctrl;

        *ddr = *ddr | 0x00ff; // set port A to outputs
        ctrl = *odr;

        // step 1, apply RS & WR, send data

        ctrl = ctrl & 0xFF00;
        ctrl = ctrl | (cmd & 0xFF);
        ctrl &= ~(LCD_RS | LCD_WR); // de-assert RS, assert WR
        *odr = ctrl;


        // step 2, wait
        i = SETUP;
        COUNTDOWN(i);

        // step 3, assert EN
        ctrl = ctrl | LCD_EN;
        *odr = ctrl;

        // step 4, wait
        i = PULSE;
        COUNTDOWN(i);

        // step 5, de-assert EN
        ctrl = ctrl & ~LCD_EN; // de-assert EN
        *odr = ctrl;

        // step 6, wait
        i = HOLD;
        COUNTDOWN(i);
}

void writechars(unsigned char *dat) {
        int i;
        unsigned short ctrl = *odr;

        do {
                lcdwait();

                *ddr = *ddr | 0x00FF; // set data lines to outputs

                // step 1, apply RS & WR, send data

                ctrl = ctrl & 0xFF00;
                ctrl = ctrl | *dat++;
                ctrl = ctrl | LCD_RS; // assert RS
                ctrl = ctrl & ~LCD_WR; // assert WR
                *odr = ctrl;

                // step 2
                i = SETUP;
                COUNTDOWN(i);

                // step 3, assert EN
                ctrl = ctrl | LCD_EN;
                *odr = ctrl;

                // step 4, wait 800 nS
                i = PULSE;
                COUNTDOWN(i);
```

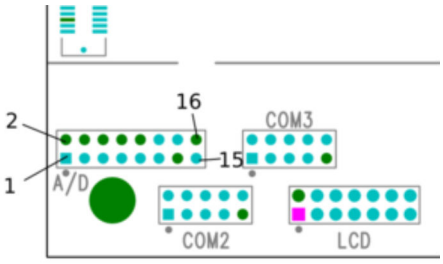
```
                // step 5, de-assert EN
                ctrl = ctrl & ~LCD_EN; // de-assert EN
                *odr = ctrl;

                // step 6, wait
                i = HOLD;
                COUNTDOWN(i);
        } while(*dat);
}
```

7.4 ADC Header

The Analog to Digital Converter consists of a 4-channel 16 bit sigma-delta converter and two, 2-channel analog switches. These are configured to allow input and conversion on two differential channels and 4 single ended channels. The 6-channel Analog to Digital signals are contained on connector HD5 which is a 16 pin (2x8) 0.1" spacing header. The connector layout and the signals carried by each pin are defined below. The input range for the differential input channels is 0- 2 VDC, and the input range on the single-ended channel is nominally 0-10 VDC.

Pinout Header

Pin	Type	Signal
1	Single ended	Channel 6
2	N/A	GND
3	Single ended	Channel 5
4	N/A	GND
5	Single ended	Channel 4
6	N/A	GND
7	Single ended	Channel 3
8	N/A	GND
9	N/A	Not connected
10	N/A	GND
11	Differential	Channel 2-
12	Differential	Channel 2+
13	N/A	GND
14	Differential	Channel 1-
15	Differential	Channel 1+
16	N/A	GND



Refer to the ADC Core section for example code.

7.5 COM Headers

The TS-8100 includes 3 2x5 0.1" pitch COM headers that feature RS232, RS485, and CAN ports. These follow a different pin layout which corresponds with a standard 10 pin header standard for UARTs. The RC-DB9 is available to convert these ports to a DB9.

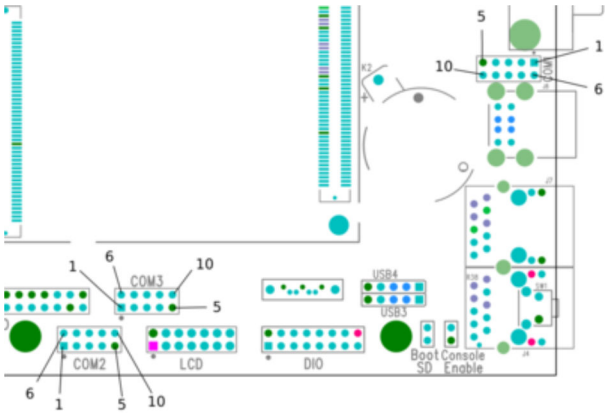
Pinout **Header**

COM1 header [1]

Pin	Name
1	ttyS1 RS485+
2	RS232 RXD [2]
3	RS232 TXD [3]
4	Unused
5	GND
6	ttyS1 RS485-
7	ttyS3 RS232 TXD
8	ttyS3 RS232 RXD
9	Unused
10	Unused

COM2 header

Pin	Name
1	ttyS5 RS485+
2	ttyS4 RS232 RXD
3	ttyS4 RS232 TXD
4	Unused
5	GND
6	ttyS5 RS485-
7	Unused
8	Unused
9	Unused



COM3 header

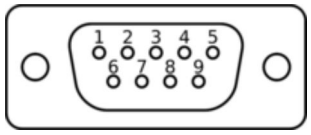
Pin	Name
1	Unused
2	ttyS6 RS232 RXD
3	ttyS6 RS232 TXD
4	Unused
5	GND
6	Unused
7	Unused
8	Unused
9	Unused
10	Unused

1. ↑ This header is brought out to both a 2x10 header and a #DB9.
2. ↑ CONSOLE RS232 RXD with "Console Enable" jumper on or ttyS2 RS232 RXD with "Console Enable" jumper off
3. ↑ CONSOLE RS232 TXD with "Console Enable" jumper on or ttyS2 RS232 TRXD with "Console Enable" jumper off

7.6 DB9

COM1 header [1]

Pin	Name
1	XUART0 RS485+
2	RS232 RXD [2]
3	RS232 TXD [3]
4	CAN_H
5	GND
6	XUART0 RS485-
7	XUART2 RS232 TXD
8	XUART2 RS232 RXD
9	CAN_L
10	Unused



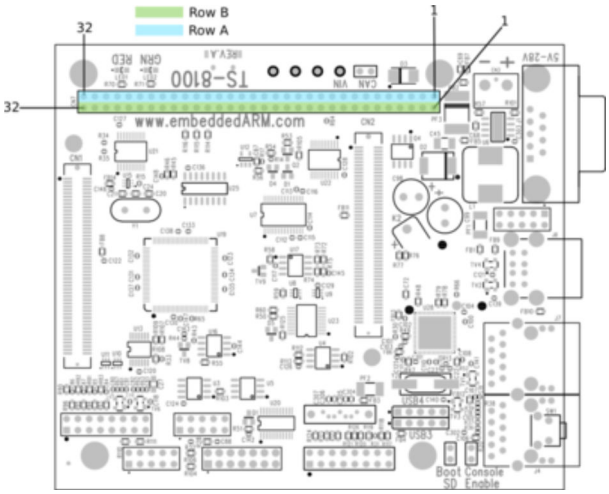
1. ↑ This header is brought out to both the DB9 and10 pin header.
2. ↑ CONSOLE RS232 RXD with "Console Enable" jumper on or XUART1 RS232 RXD with "Console Enable" jumper off
3. ↑ CONSOLE RS232 TXD with "Console Enable" jumper on or XUART1 RS232 TRXD with "Console Enable" jumper off

7.7 PC104 Header

The PC-104 connector consists of two rows of pins labeled A and B, the numbering of of which is shown below. The signals for the PC-104 are generated by the MAX240 PLD located on the baseboard. It converts the MUXBUS signals from the dual 100-pin Macrocontroller interface bus.

Any of the IO on this board labelled DIO_ can be controlled through manipulation of the TS-8100 registers.

You can also drive these DIO to manually manipulate the PC104 address to make peripherals usable that require a higher range of address than provided by the default address space of the MUXBUS.



Pin	Name	Pin	Name
A1	BUS_BHE#	B1	Ground
A2	AD_07	B2	ISA_RESET
A3	AD_06	B3	5V
A4	AD_05	B4	AD_08
A5	AD_04	B5	CPU_3.3V
A6	AD_03	B6	Not connected
A7	AD_02	B7	Not connected
A8	AD_01	B8	Not connected
A9	AD_D0	B9	VIN
A10	ISA_WAIT#	B10	Ground
A11	Ground	B11	DIO_B11
A12	Not connected	B12	DIO_B12
A13	Not connected	B13	ISA_LOW#
A14	Not connected	B14	ISA_IOR#
A15	Not connected	B15	Not connected
A16	DIO_A16	B16	Not connected
A17	DIO_A17	B17	AD_09
A18	DIO_A18	B18	AD_10
A19	DIO_A19	B19	Not connected
A20	DIO_A20	B20	AD_12
A21	DIO_A21	B21	ISA_IRQ7
A22	ISA_ADD9	B22	ISA_IRQ6
A23	ISA_ADD8	B23	ISA_IRQ5
A24	ISA_ADD7	B24	Ground
A25	ISA_ADD6	B25	AD_11
A26	ISA_ADD5	B26	AD_13
A27	ISA_ADD4	B27	AD_14
A28	ISA_ADD3	B28	AD_15
A29	ISA_ADD2	B29	5V
A30	ISA_ADD1	B30	ISA 14.3 MHZ
A31	ISA_ADD0	B31	Ground
A32	Ground	B32	Ground

WARNING: Most of the pins on the PC104 bus are only 3.3V tolerant. Refer to the schematic for more details.

7.8 SATA Connector

This controller does not support SATA.

7.9 Second Ethernet

The TS-8100 supports an optional second Ethernet port through a USB SMSC chip onboard. In linux this creates an eth1 interface. For more information see the #Network Configuration section.

Note: With the second ethernet adapter installed you cannot have the push button.

7.10 Push Button

The TS-8100 includes a push button which is connected to DIO 9 on the macrocontroller. By default this DIO is configured to reset the processor immediately when pushed. You can use this as a normal input by running:

```
ts4200ctl --resetswitchoff
```

From Debian you can use the ts4200.subr file to read this:

```
root@ts4200:~# source /initrd/ts4200.subr
root@ts4200:~# ts4200ctl --resetswitchoff
root@ts4200:~# getdiopin 9 # unpressed
1
root@ts4200:~# getdiopin 9 # pressed
0
```

8 Further Resources

For further support you can go to our Developer Forums here (http://forum.embeddedarm.com/) . You can also contact us for more information (http://www.embeddedarm.com/support/contact-us.php) .

We recommend reading our white papers if they are relevant to your project:

- Preventing Filesystem Corruption in Linux (http://www.embeddedarm.com/about/resource.php?item=459)
- XNAND (http://www.embeddedarm.com/about/resource.php?item=424)
- Meeting Real-Time Requirements with Technologic Systems Computers (http://www.embeddedarm.com/about/resource.php?item=566)

For learning more about Debian:

- The Debian Handbook (online or book) (http://debian-handbook.info/)
- Learning Debian GNU/Linux (book) (http://shop.oreilly.com/product/9781565927056.do)
- Debian Administration (online) (http://www.debian-administration.org/)

For Linux programming in general:

- The Linux Documentation Project (online) (http://tldp.org/)
- The Linux Programming Interface (book) (http://shop.oreilly.com/product/9781593272203.do)
- Linux System Programming (book) (http://shop.oreilly.com/product/9780596009588.do)
- Linux in a Nutshell (book) (http://shop.oreilly.com/product/9780596154493.do)

9 Product Notes

9.1 Errata

Synopsis	The REV A board had the white dot indicating pin 1 on the wrong side of the header.
Severity	Minor
Class	Software bug
Affected	TS-8100 REV A
Status	Fixed in REV B

Description:

On the REV A only board the white dot indicating pin 1 was on the wrong side. This was corrected in REV B, but on both boards the 5V pin is near the ethernet connector. Refer to the #USB Header section for further details.

9.2 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.

9.3 Limited Warranty

Technologic Systems warrants this product to be free of defects in material and workmanship for a period of one year from date of purchase. During this warranty period Technologic Systems will repair or replace the defective unit in accordance with the following process:

A copy of the original invoice must be included when returning the defective unit to Technologic Systems, Inc. This limited warranty does not cover damages resulting from lightning or other power surges, misuse, abuse, abnormal conditions of operation, or attempts to alter or modify the function of the product.

This warranty is limited to the repair or replacement of the defective unit. In no event shall Technologic Systems be liable or responsible for any loss or damages, including but not limited to any lost profits, incidental or consequential damages, loss of business, or anticipatory profits arising from the use or inability to use this product.

Repairs made after the expiration of the warranty period are subject to a repair charge and the cost of return shipping. Please, contact Technologic Systems to arrange for any repair service and to obtain repair charge information.