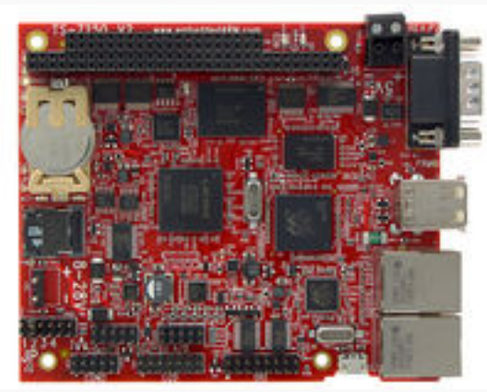
TS-7250-V2

From Technologic Systems Manuals

Contents

TS-7250-V2



Product Page (http://www.embeddedarm.com/products/board-detail.php?product=TS-7250-V2)

Schematic (http://www.embeddedarm.com/documentation/ts-7250-v2-schematic.pdf)

Mechanical Drawing (http://www.embeddedarm.com/documentation/ts-7250-v2-mechanical.pdf)

Processor

Marvell PXA166 or PXA168

800MHz ARM9 or 1066MHz ARM9

CPU Series Website (http://www.marvell.com/application-processors/armada-100/)

PXA16X Software Guide (http://www.marvell.com/application-processors/armada-100/assets/armada_16x_software_manual.pdf)

RAM

512MB DDR3

FPGA

Lattice LFXP2-8E

Reprogrammable with Opencore

DIO

28

External Interfaces

USB 2.0 2x host

2x 10/100 Ethernet

SPI

Internal Storage Media

1x SD, 2GB eMMC

Power Requirements

5VDC, or 8-28VDC

Operates around 2W

Operating Temperature

-40C

85C

Mechanical

113mm x 96.5

Height 19.3mm

Weight 109.4 (approx)

1 Overview

The TS-7250-V2 is a high performance board intended as an upgrade path to the TS-7250, TS-7260, and TS-7800. This board features a PC104 bus, 5V or 8-28V power input, and an integrated Marvell Switch allowing 2 separate Ethernet ports.

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 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.

This board accepts 5V near the DB9 cobnnector, or 8-28VDC near the MicroSD card.

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 Dec 21 2011 10:05:44
>> Copyright (c) 2011, Technologic Systems
.
.
.
```

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.

2.2 Get a Console

2.2.1 Option 1: Telnet

If your system is configured with zeroconf support (Avahi, Bonjour, etc) you can simply connect to the board with with:

```
telnet ts7250-<last 6 characters of the MAC address>.local
# You will need to use your TS-7250 MAC address, but
```

```
# for example if you mac is 00:d0:69:01:02:03
telnet ts7250-010203.local
```

Note: This is only available when soft JP1 is off. Debian can use port 2323 for a normal telnet connection if dhcp is preconfigured:

```
telnet ts7250-010203.local 2323
```

When the board first powers up it has two network interfaces. The first interface eth0 is configured to use IPv4LL, and eth0:0 is configured to use DHCP. The board broadcasts using multicast DNS advertising the _telnet._tcp service. You can use this to query all of the available TS-7250s on the network.

From Linux you can use the avahi commands to query for all telnet devices with:

```
avahi-browse _telnet._tcp
```

Which would return:

```
+   eth0 IPv4 TS-7250 console [4f47a5]                    Telnet Remote Terminal local
+   eth0 IPv4 TS-7250 console [4f471a]                    Telnet Remote Terminal local
```

This will show you the mac address you can use to resolve the board. In this case you can connect to either ts7250-4f47a5 or ts7250-4f47a5.

From Windows you can use Bonjour Print Services (http://support.apple.com/downloads/Bonjour_for_Windows) to get the dns-sd command. OSX also comes preinstalled with the same command. Once this is installed you can run:

```
dns-sd -B _telnet._tcp
```

Which will return:

```
Browsing for _telnet._tcp
Timestamp     A/R Flags if Domain                  Service Type             Instance Name
10:27:57.078  Add     3  2 local.                  _telnet._tcp.            TS-7250 console [4f47a5]
10:27:57.423  Add     3  2 local.                  _telnet._tcp.            TS-7250 console [4f47a5]
```

This will show you the mac address you can use to resolve the board. In this case you can connect to either ts7250-4f47a5.local or ts7250-4f47a5.local.

2.2.2 Option 2: Serial Console

The serial console is accessible through either the DB9 connector or the USB device connector. If jumper 2 is on, the console comes from the DB9 connector as RS232 on pins 2 and 3. If JP2 is off, the console goes through the USB port as CDC-ACM. The serial console is provided through this port at 115200 baud, 8n1, with no flow control.

The USB port uses a Cortex-M0 ARM microcontroller to create a CDC ACM serial device on a host PC.

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/ttyACM0 (this is what the USB device normally appears as), 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/ttyACM0
```

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

```
screen /dev/ttyACM0 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/ttyACM0' then hit "enter"

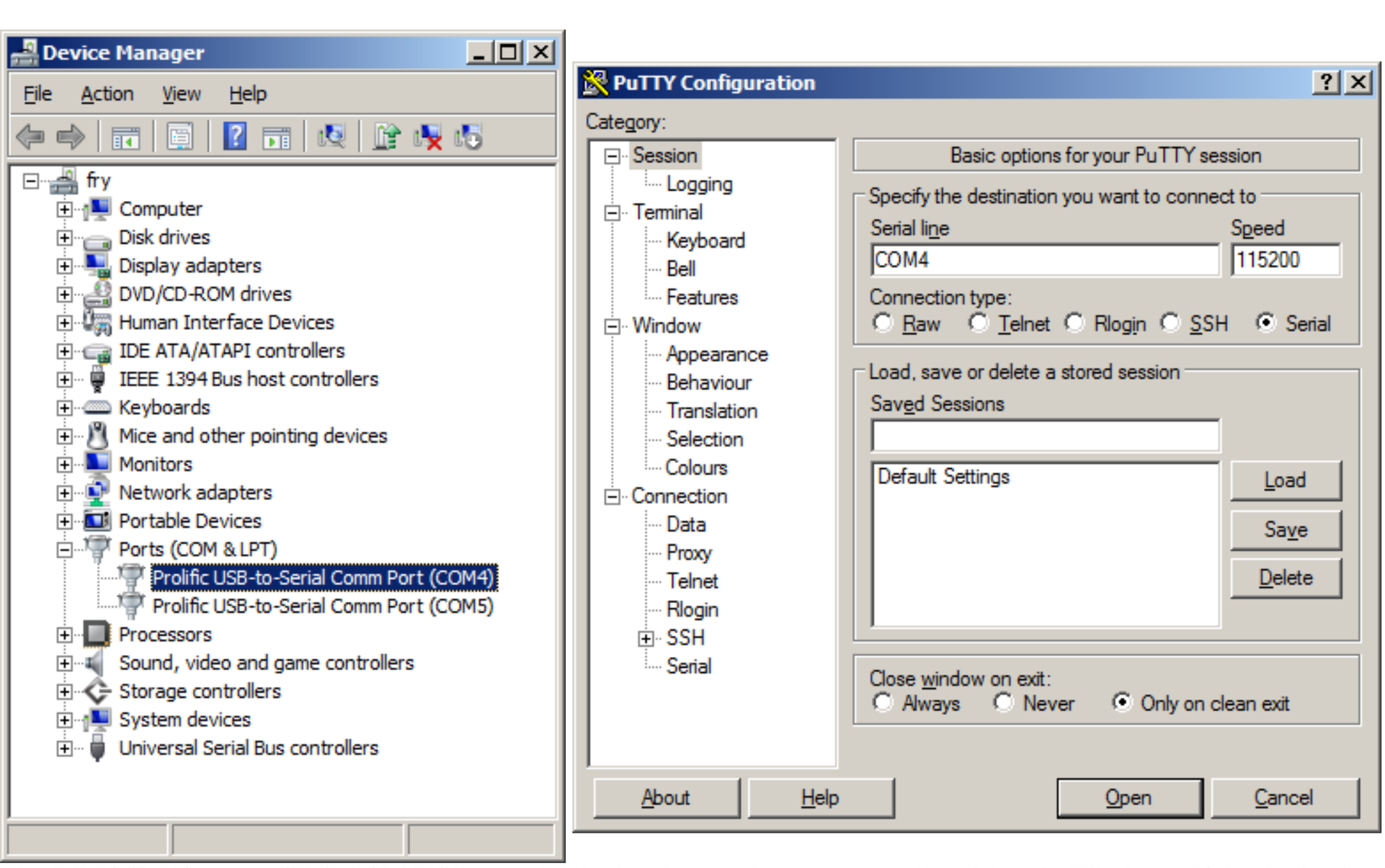
- If needed, modify the settings to match this and hit "esc" when done:

```
  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.



The TS-7250-V2 uses a CDC ACM USB to serial device that needs a corresponding driver in Windows. If the device is not detected correctly when its plugged in and Windows brings up the "Found new Hardware" wizard, download this zipfile (http://www.lpcware.com/content/blog/usb-cdcacm-drivers-windows) , and point the wizard to the unpacked folder. More recent windows will require driver signing to be disabled before this driver will work.

2.3 Initrd / Busybox

When the board first boots up you should have a console such as:

```
>> TS-BOOTROM - built Mar 14 2013 15:01:50
>> Copyright (c) 2012, Technologic Systems
.
.
Uncompressing Linux... done, booting the kernel.
Booted in 0.90s
Initramfs Web Interface: http://ts47XX-112233.local
```

This is a minimalistic initial ram filesystem that includes our specific utilities for the board, and is then used to bootstrap the Linux root. The initramfs is built into the kernel image so it cannot be modified without rebuilding the kernel, but it does include several bits for common configuration option we call soft jumpers.

Soft Jumpers

Jumper	Function
1	Boot automatically to Debian [1]
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Boot as fast as possible [2]
8	Skip full DRAM test on startup [3]

1. ↑ initramfs boot is default. Configure the network in Debian before setting this jumper if you do not have the serial console.
2. ↑ This option skips a significant amount of setup and will boot to a single SD card as fast as possible with no initialization. This mode will still execute /mnt/root/ts/init if it exists, or boot to Debian if jp1 is set.
3. ↑ The DRAM test can be used to verify the RAM, but adds approximately 20 seconds to the boot time. This should normally only be enabled when diagnosing problems.

There are 2 ways to manipulate soft jumpers on the board. The web interface at "http://ts<model>-<last 6 chars of the MAC>.local" has a list of checkboxes that will immediately change the values. You can also use tshwctl:

```
# Boot automatically to Debian:
tshwctl --setjp=1

# Or revert to the initramfs:
tshwctl --removejp=1
```

The Debian boot can also be inhibited by creating a file in /ts/fastboot in the Debian root. While this file exists the board will stop booting at the initramfs. If you do not have a serial console, **make sure you first configure Debian's network settings first *before* booting directly to Debian**. Once JP1 is enabled, the initramfs does not run ifplugd/udhcpc to configure the network.

Most development should be done in Debian, however many applications are capable of running from the initramfs. Utilities from Debian can be accessed under /mnt/root as read only, but for Debian services, or using apt-get a full boot into Debian should be performed. The initramfs itself cannot be easily modified, and it is not recommended to do so. The initramfs however has several hooks for applications to manipulate it's behavior.

/mnt/root/ts/init

For headless applications you can create a bash script with any initialization you require in /ts/init. This does not use the same $PATH as Debian, so you should enter the full path to any applications you intend to run from this environment. The init file does not exist by default and must be created:

```
#!/bin/sh

/path/to/your/application &
```

/mnt/root/ts/initramfs-xinit

Graphical applications should use /ts/initramfs-xinit. The xinit (https://wiki.debian.org/Xinitrc) file is used to start up a window manager and any applications. The default initramfs-xinit starts a webbrowser viewing localhost:

```
#!/bin/sh
# Causes .Xauthority and other temp files to be written to /root/ rather than default /
export HOME=/root/
# Disables icewm toolbars
export ICEWM_PRIVCFG=/mnt/root/root/.icewm/

# minimalistic window manager
icewm-lite &

# this loop verifies the window manager has successfully started
while ! xprop -root | grep -q _NET_SUPPORTING_WM_CHECK
do
    sleep 0.1
done

# This launches the fullscreen browser.    If the xinit script ever closes, x11 will close.  This is why the last
# command is the target application which is started with "exec" so it will replace the xinit process id.
exec /usr/bin/fullscreen-webkit http://localhost
```

/mnt/root/ts/config

This config file can be used to alter many details of the initramfs boot procedure.

```
## This file is included by the early init system to set various bootup settings.
## if $jp7 is enabled none of these settings will be used.

## Used to control whether the FPGA is reloaded through software.
```

```
## 1 to enable reloading (default)
## 0 to disable reloading
#CFG_FPGARELOAD="0"

## By default dns-sd is started which advertises the ts<model>-<last 6 of mac>
## telnet and http services using zeroconf.
## 1 to enable dns-sd (default)
## 0 to disable dns-sd
#CFG_DNSSD_EN="0"

## This is used to discover hosts and advertise this host over multicast DNS.
## 1 to enable mdns (default)
## 0 to disable mdns
#CFG_MDNS_EN="0"

## ifplugd is started in the initramfs to start udhcpc, and receive an ipv4ll
## address.
## 1 to enable ifplugd (default)
## 0 to disable ifplugd
#CFG_IFPLUGD_EN="0"

## By default telnet is started on port 2323.
## 1 to enable telnet (default)
## 0 to disable telnet
##CFG_TELNET_EN="0"

## The busybox webserver is used to display a diagnostic web interface that can
## be used for development tasks such as rewriting the SD or uploading new
## software
## 1 to enable (default)
## 0 to disable
##CFG_HTTPD_EN="0"

## This eanbles a reset switch on DIO 29 (TS-7700), or DIO 9 on all of the
## boards.  Pull low to reset the board immediately.
## 1 to enable the reset sw (default)
## 0 to disable
#CFG_RESETSW_EN="0"

## The console is forwarded through xuartctl which makes the cpu console available
## over telnet or serial console.
## 1 to enable network console (default)
## 0 to disable network console
#CFG_NETCONS_EN="0"

## By default Alsa will put the SGTL5000 chip into standby after 5 seconds of
## inactivity.  This is desirable in that it results in lower power consumption,
## but it can result in an audible popping noise.  This setting prevents
## standby so the pop is never heard.
## 1 to disable standby
## 0 to enable standby (default)
#CFG_SGTLNOSTBY="1"

## xuartctl is used to access the FPGA uarts.  By default it is configured to
## be IRQ driven which is optimized for best latency, but at the cost of
## additional CPU time.  You can reduce this by specifying a polling rate.
## The xuartctl process also binds to all network interfaces which can provide a
## simple network API to access serial ports remotely.  You can restrict this to
## the local network with the bind option.
## Configure XUART polling 100hz
## Default is IRQ driven
CFG_XUARGS="--irq=100hz"
## Configure xuartctl to bind on localhost
## Default binds on all interfaces
#CFG_XUARGS="--bind 127.0.0.1 --irq=100hz"
## For a full list of arguments, see the xuartctl documentation here:
## http://wiki.embeddedarm.com/wiki/Xuartctl#Usage

## By default the system will probe for up to 10s on USB for a mass storage device
## and mount the first partition.  If there is an executable /tsinit script in the
## root this will be executed.  This is intended for production or updates.
## 2 to enable USB init always (adds 10s or $CFG_USBTIME to startup)
## 1 to enable USB init when jp1=0 (default)
## 0 to disable USB init always
#CFG_USBINIT="2"

## The USB init script by default blocks for 10s to detect a thumb drive that
## contains the tsinit script.  Most flash media based drives can be detected
## in 3s or less.  Some spinning media drives can take 10s, or potentially longer.
## This options is the number of seconds to wait before giving up on the
## mass storage device.
#CFG_USBTIME="3"

### TS-8700
## Using the TS-8700 baseboard the board will by default initialze all of the
## ethernet ports as individual vlan ports, eg eth0.1, eth0.2, eth0,3, and eth0.4
## The alterantive option sets Port A to eth0.1, and Ports B-D to eth0.2, or
## you can configure all ethernet ports as a single eth0 port.
## See http://wiki.embeddedarm.com/wiki/TS-8700 for more information
## 2 disables any vlan and passes through all interfaces to eth0
## 1 enables "WLAN" mode setting "A" as eth0.1, and all others as eth0.2
## 0 enables "VLAN" mode for 4 individual ports (default)
#CFG_4ETH="1"

### TS-4712 / TS-4720
## These boards include an onboard switch with 2 external ports.  By default
## the switch will detect if it is on a known baseboard that supports the second
## ethernet switch port, and set up VLAN rules to define eth0.1 and eth0.2.  The
## other option is to configure the switch to pass through the packets to eth0
## regarless of port.
## 2 Disable VLAN and pass through to eth0
## 1 Enable VLAN on all baseboards
```

```
## 0 Enable VLAN on supported baseboards (Default)
#CFG_2ETH="1"
```

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

This board includes a Marvell switch chip which allows 2 separate networks using the same network interface. See the Ethernet port section for more information on the switch settings. When the switch is configured for 2 separate networks (as it is by default), the eth0 interface should not be directly configured. The switch will provide the eth0.1 and eth0.2 interfaces which can be configured. If the switch is configured to pass through, then the eth0 interface should be used as normal.

The board is initially configured to boot to the initramfs. While in this state ifplugd will automatically assign an IP address, and even if you type "exit" to boot to Debian it will retain the address it was assigned. If you need to boot to the full Debian, networking should first be set up in the /etc/network/interfaces file. As an example, to get dhcp from eth0.1: Open /etc/network/interfaces

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

auto eth0.1
iface eth0.1 inet dhcp
```

Once this file is set up, either reboot or "/etc/init.d/networking restart" for this to take effect.

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

```
# NOTE:  These are generic examples.  Be sure to read the entire networking section before trying any of these.
# Bring up the CPU network interface (for systems with only one Ethernet)
ifconfig eth0 up

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

# Or if you're on a TS-7250-V2...
ifconfig eth0.1 up
ifconfig eth0.2 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 initramfs:

```
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) .

3.1.1 WIFI Client

This board optionally supports 802.11 through the WIFI-N-USB-2 module using the ath9k_htc driver.

Scan for a network

```
ifconfig wlan0 up
```

```
# Scan for available networks
iwlist wlan0 scan
```

In this case I'm connecting to "default" which is an open network:

```
          Cell 03 - Address: c0:ff:ee:c0:ff:ee
                    Mode:Managed
                    ESSID:"default"
                    Channel:2
                    Encryption key:off
                    Bit Rates:9 Mb/s
```

To connect to this open network:

```
iwconfig wlan0 essid "default"
```

You can use the iwconfig command to determine if you have authenticated to an access point. Before connecting it will show something similar to this:

```
# iwconfig wlan0
wlan0     IEEE 802.11bgn  ESSID:"default"
          Mode:Managed  Frequency:2.417 GHz  Access Point: c0:ff:ee:c0:ff:ee
          Bit Rate=1 Mb/s   Tx-Power=20 dBm
          Retry  long limit:7   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
          Link Quality=70/70  Signal level=-34 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0
```

If you are connecting using WEP, you will need to define a network key:

```
iwconfig wlan0 essid "default" key "yourpassword"
```

If you are connecting to WPA you will need to use wpa_passphrase and wpa_supplicant:

```
wpa_passphrase the_essid the_password > /etc/wpa_supplicant.conf
```

Now that you have the configuration file, you will need to start the wpa_supplicant daemon:

```
wpa_supplicant -Dwext -iwlan0 -c/etc/wpa_supplicant.conf -B
```

Now you are connected to the network, but this would be close to the equivalent of connecting a network cable. To connect to the internet or talk to your internal network you will need to configure the interface. See the #Configuring the Network for more information, but commonly you can just run:

```
dhclient wlan0
```

Some older images did not include the "crda" and "iw" packages required to make a wireless connection. If

Note: you cannot get an ip address you may want to connect over ethernet and install these packages with "apt-get install crda iw -y".

3.1.2 Host a WIFI Access Point

The software image includes a build of compat-drivers from 3.8 so a large amount of wireless devices are supported. Some devices support AP/Master mode which can be used to host an access point. The WIFI-N-USB-2 module we provide also supports this mode.

First install hostapd to manage the access point:

```
apt-get update && apt-get install hostapd -y
```

Edit /etc/hostapd/hostapd.conf to include:

```
interface=wlan0
driver=nl80211
ssid=YourAPName
channel=1
```

Note: Refer to the kernel's hostapd documentation (http://wireless.kernel.org/en/users/Documentation/hostapd) for more wireless configuration options.

To start the access point launch hostapd:

```
hostapd /etc/hostapd/hostapd.conf &
```

This will create a valid wireless access point, however many devices will not be able to connect without either a static connection, or a DHCP server. Refer to Debian's documentation for more details on DHCP configuration (https://wiki.debian.org/DHCP_Server) .

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)
    /path/to/your/application
    ## If you are launching a daemon or other long running processes
    ## this should be started with
    # nohup /usr/local/bin/yourdaemon &
    ;;
  stop)
    # if you have anything that needs to run on shutdown
    /path/to/your/shutdown/scripts
    ;;
  *)
    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
```

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

Click to download the latest 4GB SD card image. (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-3x-latest.dd.bz2)

Using onboard web interface (recommended)

The initramfs contains a #Web interface that can be used to backup/restore the software image. From the main page, you can download a complete backup containing the MBR, Kernel, initramfs, and Debian filesystem by clicking "backup.dd". You can click "Choose File" and browse to a previous backup.dd, or the link above to rewrite the SD card.

Using another Linux workstation

If you do not have an SD card that can boot to the initramfs, you can download the sd card image (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-latest.dd.bz2) and rewrite this from a Linux workstation. A USB MicroSD adapter can be used to access the card. First, you must find out which /dev/ device corresponds with your USB reader/writer.

Step 1 Option 1 (lsblk)

Newer distributions include a utility called "lsblk" which allows simple identification of the intended card:

```
lsblk
```

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda      8:0    0   400G  0 disk
├─sda1   8:1    0   398G  0 part /
├─sda2   8:2    0     1K  0 part
└─sda5   8:5    0     2G  0 part [SWAP]
sr0     11:0    1  1024M  0 rom
sdc      8:32   1   3.9G  0 disk
├─sdc1   8:33   1   7.9M  0 part
├─sdc2   8:34   1     2M  0 part
├─sdc3   8:35   1     2M  0 part
└─sdc4   8:36   1   2.8G  0 part
```

In this case my SD card is 4GB, so sdc is the target device.

Step 1 Option 2 (dmesg)

After plugging in the device, you can use dmesg to list

```
scsi 9:0:0:0: Direct-Access     Generic  Storage Device   0.00 PQ: 0 ANSI: 2
sd 9:0:0:0: Attached scsi generic sg2 type 0
sd 9:0:0:0: [sdb] 7744512 512-byte logical blocks: (3.96 GB/3.69 GiB)
```

In this case, sdc is shown as a 3.96GB card.

Step 2

Once you have the target /dev/ device you can use "dd" to backup/restore the card. To restore the board to stock, or rewrite to the latest SD image:

```
wget ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-latest.dd.bz2
bzip2 -d 4gbsd-471x-latest.dd.bz2

# Specify your block device instead of /dev/sdc
# Note that this does not include a partition, so use /dev/sdc instead of
# using /dev/sdc1
dd if=4gbsd-471x-latest.dd conv=fsync bs=4M of=/dev/sdc
```

To take a backup of your entire SD card, you can switch the input file and the output file:

```
dd if=/dev/sdc conv=fsync bs=4M of=backup.dd
```

4.2 eMMC

Click to download the latest 2GB SLC eMMC image. (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gb-emmc-slc-471x-3x-latest.dd.bz2)

WARNING: Make sure the SD and eMMC filesystems are mounted read only while writing any images. If dd is used to read or write a disk while a filesystem on it is mounted read/write this will result in a corrupt image.

First boot to the initramfs. These steps should **not be run from the full Debian environment**, so to get back to the initramfs, first run:

```
tshwctl --removejp 1 && reboot
```

First, boot to the SD card. Once you are in the initramfs, mount a network share or usb drive containing the emmc image.

Write an Image to eMMC

```
# Check /mnt/root.  If it shows a file "root.version", you can proceed.
# If it shows nothing, you need to wait for the Debian filesystem
until [ -e /mnt/root/root.version ]; do sleep .1; done

# Plug in a thumbdrive formatted with ext2/3/4 or fat32
mkdir /mnt/usbdev
mount -o ro /dev/sda1 /mnt/usbdev
umount -l /mnt/root/

dd if=/mnt/usbdev/emmcimage.dd bs=4M of=/dev/nbd1
sync
reboot
```

Save an image from eMMC

```
# Check /mnt/root.  If it shows a file "root.version", you can proceed.
# If it shows nothing, you need to wait for the Debian filesystem
until [ -e /mnt/root/root.version ]; do sleep .1; done

# Plug in a thumbdrive formatted with ext2/3/4 or fat32
mkdir /mnt/usbdev
mount /dev/sda1 /mnt/usbdev
umount -l /mnt/root/

dd if=/dev/nbd1 bs=4M of=/mnt/usbdev/emmcimage.dd
umount /mnt/usbdev
sync
```

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 Accessing Hardware Registers

The standard assumption in Linux is that kernel drivers are required in order to control hardware. However, it is also possible to talk to hardware devices from user space. In doing so, one does not have to be aware of the Linux kernel development process. This is the recommended way of accessing hardware on a TS-SOCKET system. The special /dev/mem device implements a way to access the physical memory from the protected user space, allowing reading and writing to any specific memory register. Applications may be allowed temporary access through memory space windows granted by the mmap() system call applied to the /dev/mem device node.

The following C code is provided as an example of how to set up user space access to the SYSCON registers at base address 0x80004000:

```
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <assert.h>
static volatile unsigned short *syscon;
static unsigned short peek16(unsigned int adr) {
        return syscon[adr / 2];
}
static void poke16(unsigned int adr, unsigned short val) {
        syscon[adr / 2] = val;
}

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

        assert(devmem != -1);
        syscon = (unsigned short *) mmap(0, 4096,
          PROT_READ | PROT_WRITE, MAP_SHARED, devmem, 0x80004000);

        poke16(0x6, 0x3); // disable watchdog
        poke16(0x12, peek16(0x12) | 0x1800); // turn on both LEDs

        return 0;
}
```

Important Notes about the preceding example:

- The peek16 and poke16 wrapper functions make the code more readable due to how pointer arithmetic/array indexing works in C, since the same offsets from the register map appear in the code.
- Make sure to open using O_SYNC, otherwise you may get a cachable MMU mapping which, unless you know what you're doing, probably is not what you want when dealing with hardware registers.
- mmap() must be called only on pagesize (4096 byte) boundaries and size must at least have pagesize granularity.
- Only the root user can open '/dev/mem'. For testing, this just means the tester needs to be root, which is normal in embedded Linux. For deployment in the field under Debian, this can be an issue because the init process does not have root privileges. To get around this, make sure the binary is owned by root and has the setuid bit set. The command 'chmod +s mydriver' will set the setuid flag.
- The pointers into memory space should have the same bit width as the registers they are accessing. In the example above, the TS-4710 FPGA registers are 16 bits wide, so an unsigned short pointer is used. With very few exceptions, FPGA registers on TS-SOCKET macrocontrollers will be 16 bits wide and CPU registers will be 32 bits wide. Unsigned int, unsigned short, and unsigned char pointers should be used for 32, 16, and 8 bit registers, respectively.
- When compiling ARM code that emits 16 bit or 8 bit hardware register accesses, it is important to add the compiler switch -mcpu=arm9. Otherwise the wrong opcodes may be emitted by the compiler and unexpected behavior will occur.
- Pointers into memory space must be declared as volatile.

5.2 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
```

5.3 Compile the Kernel

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:

```
sudo apt-get install build-essential libncurses5-dev libncursesw5-dev git
## If you are on a 64-bit system then 32-bit libraries will be required for the toolchain
# sudo apt-get install ia32-libs
```

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:
wget ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4700-linux/cross-toolchains/arm-2008q3.tar.gz

# Extract the toolchain
tar xvf arm-2008q3.tar.gz

# Move arm-2008q3 to a permanent location, eg /opt/toolchains/
mkdir /opt/toolchains/
mv arm-2008q3 /opt/toolchains/

# Download the Kernel sources
git clone https://github.com/embeddedarm/linux-3.14-pxa16x.git

cd linux-3.14-pxa16x

# Set the CROSS_COMPILE variable to the absolute path to the toolchain.
export CROSS_COMPILE=/opt/toolchains/arm-2008q3/bin/arm-none-linux-gnueabi-
export ARCH=arm

# This sets up the default configuration that we ship with for the TS-471x
make ts471x_defconfig
```

Once you have the configuration ready you can make your changes to the kernel. Commonly a reason for recompiling is to add support that was not built into the standard image's kernel. You can get a menu to browse available options by running:

```
make menuconfig
```

You can use the "/" key to search for specific terms through the kernel.

Build the kernel

Once you have it configured you can begin building the kernel. This usually takes about 5-10 minutes.

```
make
```

The new kernel will be at "arch/arm/boot/Image".

Install the Kernel and Modules

Install the target SD card in your workstation, and mount the Debian partition. For example, if your workstation's SD card is /dev/sdc:

```
# Update this to point to your SD card block device
export DEV=/dev/sdc
sudo mkdir /mnt/sd/
sudo dd if=arch/arm/boot/zImage of="$DEV"1 conv=fsync
sudo mount "$DEV"2 /mnt/sd/
INSTALL_MOD_PATH=/mnt/sd/ sudo -E make modules_install
INSTALL_HDR_PATH=/mnt/sd/ sudo -E make headers_install
sudo umount /mnt/sd/
sync
```

6 Features

6.1 CPU

The TS-4710 supports two processors from Marvell's Armada 100 series. The common features will be described in other sections, but for more details see the CPU user guide (http://www.marvell.com/application-processors/armada-100/assets/armada_16x_software_manual.pdf) .

PXA16X Feature Comparison

Feature	PXA166 (88AP166)	PXA168 (88AP168)
Frequency	800MHz	1066MHz
Video Playback Acceleration (gstreamer)	Supported up to D1	Supported up to 720p
Maximum Framebuffer Fesolution	Up to WUXGA	Up to WUXGA
PCI-Express support	N/A	1x PCI-E 2.0

6.2 MicroSD Card Interface

This macrocontroller uses our SD controller implementation which supports MicroSD, MicroSDHC, and MicroSDXC cards. This controller has been tested with Sandisk Extreme SD cards which allow read speeds up to 20.5MB/s, and write speeds up to 21.5MB/s.

The support for the SD controller is provided by sdctl which serves up a /dev/nbd0 for the entire block device. The kernel also includes a module that will break this up into partitions. Our default software image contains 2 partitions:

Device	Contents
/dev/nbd0	SD Card block device
/dev/nbd0p1	Kernel and initramfs
/dev/nbd0p2	Full Linux Root

6.3 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/) .

The Linux kernel supports up to 16 IRQs from the FPGA. When the CPU receives an IRQ from the FPGA, it uses the IRQ register in the #Syscon to find out which IRQ on the MUX is triggering. Currently only three IRQs are used. Off-board IRQs 5, 6, and 7 correspond to FPGA IRQs 0, 1, and 2, respectively. FPGA IRQs 3 to 15 are reserved for future uses. If the DIO pins are not being used as IRQs, they can be masked out by writing 0 to the corresponding bit in the IRQ mask register.

> **Note:** Any of the EVGPIO can trigger the core interrupt if they are unmasked. See the #EVGPIO section for more information.

IRQ #	Name	Socket Location
49	Combined GPIO Interrupt	Any MFP pin
64	XUART IRQ	N/A
65	CAN1 IRQ	N/A
66	CAN 2 IRQ	N/A
67	IRQ5/DIO_00[1]	
68	IRQ6/DIO_01[1]	
69	IRQ7/DIO_02[1]	
78	EVGPIO #1	N/A
79	EVGPIO #2	N/A

1. ↑ [1.0] [1.1] [1.2] The PC/104 IRQs need to have the MUXBUS bus enabled in order to function as IRQs

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;
}
```

Any of the MFP pins can be repurposed to trigger IRQ 49. For example, to make MFP_46 (CN2_72) trigger on a rising edge:

```
# Enable rising edge detection on MFP_46
peekpoke 32 0xD4019034 0x4000

# Unmask MFP_46
peekpoke 32 0xD40190A0 0x4000

# to clear the interrupt after it has been triggered
peekpoke 32 0xD401904c 0x4000
```

See page 169 of the CPU manual (http://www.marvell.com/application-processors/armada-100/assets/armada_16x_software_manual.pdf) for more information on the interrupt controller.

6.4 RTC

The RTC is accessed using tshwctl. This is automatically retrieved on startup, but must be set manually.

```
# Save the running system clock to the RTC
tshwctl --setrtc

# Set the system clock from the RTC
tshwctl --getrtc
```

6.5 NVRAM

The RTC has an included 128 byte NVRAM which can be accessed using tshwctl.

```
tshwctl --nvram
```

This will return a format such as:

```
nvram0=0xf7f8a73e
nvram1=0x2fef5ae0
nvram2=0x48ca4278
...
nvram31=0x70544510
```

This breaks up the NVRAM into 32x 32-bit registers which can be accessed in bash. As this uses the name=value output, you can use "eval" for simple parsing:

```
eval `tshwctl --nvram`
echo $nvram2
```

From the above value, this would return 0x48ca4278. To set values, you can use environment variables:

```
nvram0=0x42 tshwctl --nvram
```

If you read back nvram0, this should now confirm the value is 0x42.

6.6 Temperature Sensor

This macrocontroller includes temperature sensors located on the CPU and RTC. Both of these can be read using tshwctl:

```
tshwctl --rtctemp
tshwctl --cputemp
```

Both of these will return the temperature in millivolts.

6.7 LEDs

On all of our baseboards we include 2 indicator LEDs which are under software control. You can manipulate these using "hwctl --greenledon --redledon" or "tshwctl --greenledoff --redledoff". The LEDs have 4 behaviors from default software. The LEDs are also controllable via the Syscon register at offset 0x12.

Green Behavior	Red behavior	Meaning
Solid On	Off	System is booted and running
Solid 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 tshwctl 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 a stable 5V is being provided and the supply is capable of providing at least 1A to the macrocontroller, an RMA (http://www.embeddedarm.com/about/rma.php) is suggested.
Blinking about 5ms on, about 10ms off.	Blinking about 5ms on, about 10ms off.	The board is receiving too little power, or something is drawing too much current from the macrocontroller's power rails.

6.8 Web Interface

This macrocontroller includes a web interface that can be used to simplify common tasks when working with our embedded systems.

Uploading files

On the main page you can select a file and upload. These have various functions depending on the file extensions:

Filename/Extension	Description
*.vme.bz2	Upload FPGA to be soft reloaded automatically on startup. This will be copied to "/ts/" path in the linux root filesystem.
ko.tar.bz2	While most kernel modules will be loaded automatically when needed, if you include a ko.tar.bz2 this will insmod each file in the archive automatically on startup. This will be copied to the "/ts/" path in the linux root filesystem.
init	If this file exists and the JP1 is not set, the board will boot to the initramfs and execute this script. This can be used to have an application automatically run on startup without proceeding with the Linux root filesystem's traditionally lengthy startup. This can have an application running within seconds after poweron. The $PATH variable is set up to be abel to resolve most applications in the linux root filesystem, and the libraries of the full distribution are available. As this does not run through the normal startup, any running services or network configuration will need to be started manually.
Image, zImage, kernel*.dd	This will automatically replace the first partition containing the Kernel.
root*.dd	This will completely replace the second partition with the uploaded dd file.
mbr.dd|mbr*.dd	Replace the MBR on the current boot image.
*.dd	Any file not caught by one of the previous "*.dd" filenames will entirely replace the SD image.
*.sh	Any file named *.sh will automatically be copied to /tmp, set as executable and run.
root*.tar	This will remove all data from the linux root filesystem and replace it with the contents of the uploaded root*.tar file.
src*.tar	This will extract the contents to the "/ts/" directory in the Linux rootfilesystem and if present, execute the "Makefile". This could be used to build a project, and automatically install it.
*.c *.cpp	Any uploaded c/cpp file will automatically be compiled and executed. The applications "stdout" will be printed out to the web page.
*	Any other files not captured by a previous pattern will be copied to the "/ts/" path in the Linux root filesystem.

Any uploaded file can be compressed with bzip2 or gzip before uploading. The file will be decompressed and then processed as normal as described in the above table.

Downloading Files

On the main page there is a download link for 4 files. Any downloaded file will be renamed to contain the date in the format "date -Iminutes".

Filename	Description
backup.dd	This is a backup containing the MBR, Kernel/initramfs, and Linux root filesystem.
root.dd	This is a backup of a complete dd of the Linux root filesystem.
root.tar	The root.tar contains a complete tar of the contents in the root filesystem.
kernel.dd	This file contains a copy of the kernel and initramfs.

Duplicating an SD card

This page can be used to either duplicate an SD card, or convert a software image to a single or dual Doublestore card configuration. When this page is loaded it copies the kernel/initramfs to ram. You will need to have the root.tar downloaded before continuing.

Once you have loaded this page and you have a copy of the root.tar, you can either remove the current SD card, or leave it in if you intend to convert it to DoubleStore. On step 2, you can select "Standard" to write a new sd card without doublestore, or you can create a single or dual card configuration. Click "Format card" after selecting either option.

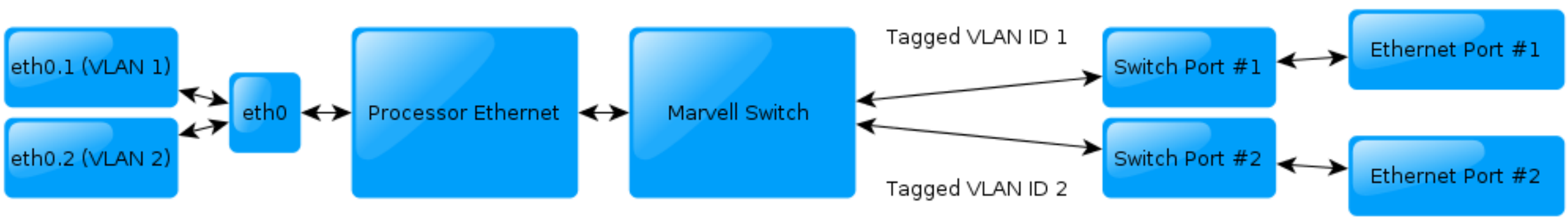
After being formatted you can upload the root*.tar file to reformat the rest of the card. Once this is completed, you can reboot to test out the card, or restart the procedure to create another card.

Find other TS41X boards

By default this board broadcasts itself using multicast DNS which can be used to detect all other similar boards on the network. This will print out the last 6 of the MAC address which can be used to uniquely identify each board.

6.9 Ethernet Port

The Marvell Processor implements a 10/100 ethernet controller with support built into the Linux kernel. The TS-7250-V2 includes an integrated Marvell Ethernet switch that allows multiple interfaces from one 10/100 port. This allows a total bandwidth of 100MB/s between both ports.



The default behavior will configure the ports to act as 2 individual ports. The port labelled "T1" near the USB connector is "eth0.2" and "T2" is "eth0.1". In this default mode traffic will not route between the two ports without external software doing so. While in this mode the eth0 interface should not be used directly, only the eth0.1 or eth0.2.

You can use standard Linux utilities such as ifconfig/ip to control eth0 and the vlan interfaces. See the #Configuring the Network section for more details. For the specifics of this interface see the CPU manual (http://www.marvell.com/application-processors/armada-100/assets/armada_16x_software_manual.pdf) .

The switch ports can also use tshwctl to detect link and the negotiated link speed:

```
root@ts7250-f7c0ff:~# tshwctl --ethinfo
switch_model=88E6020
switch_ports=a b
switchporta_link=0
switchporta_speed=10HD
switchportb_link=1
switchportb_speed=100FD
```

The switch can also be configured to pass through all traffic while also allowing communication to eth0. This is similar to the ethernet bridge functionality in Linux, but does not require any processing from Linux to pass through data. Unlike the software bridge modes, this does not support packet inspection or filtering. To use the bridge mode edit the file /ts/config, uncomment CFG_2ETH, and set it to "2":

```
CFG_2ETH="2"
```

See the /ts/config documentation here for more information.

6.10 DIO

6.10.1 EVGPIO

This board features the EVGPIO core (Event Driven GPIO) which allows a low bandwidth mechanism to monitor all DIO on 2 shared interrupts. All DIO are accessed atomically through two registers. The data register is used to read dio state changes, set output values, and data direction. The mask register is used to set which DIO will trigger the IRQ and provide state changes to the data register. We provide "evgpioctl" which can be used to access these DIO:

```
Usage: evgpioctl [OPTIONS] ...
EVGPIO utility

  -i, --getin  <dio>    Returns the input value of a DIO
  -s, --setout <dio>    Sets a DIO output value high
  -l, --clrout <dio>    Sets a DIO output value low
  -o, --ddrout <dio>    Set DIO to an output
  -d, --ddrin <dio>     Set DIO to an input
  -m, --setmask <dio>   Mask out a DIO so it does not provide input
                        event changes and trigger the shared IRQ
  -c, --clrmask <dio>   Clear the mask from a DIO so it provides input
                        event changes and trigger the shared IRQ
  -a, --maskall         Mask out all DIO so they do not provide input
                        event changes and trigger the shared IRQ
  -u, --unmaskall       Clear all masks so all DIO provide input event
                        changes and trigger the shared IRQ
  -w, --watch           Prints csv output when any unmasked DIO changes
                        <dio>,<1=high,0=low>
```

This provides a simple interface that can be used in scripts, or wrapped for higher level software access.

```
# Set DIO 31 to a high output
evgpioctl --ddrout 31 --setout 31

# Set DIO 31 to a low output
evgpioctl --setout 31

# Read the value of DIO 30
evgpioctl --ddrin 30 --getin 30

# The input return values are parsable and can be used easily in scripts:
eval $(evgpioctl --getin 30)
echo $dio30
```

The DIO mappings are found in the table below. The sources for this utility are available here:

- evgpio.c (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7700-linux/sources/evgpio.c)
- evgpio.h (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7700-linux/sources/evgpio.h)
- evgpioctl.c (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7700-linux/sources/evgpioctl.c)

You can also manipulate the EVGPIO data and mask registers directly in your application. There are 4 registers to control:

Offset	Bits	Usage
0x80004036	15:0	EVGPIO Data Core #0 (0-63)
0x80004038	15:0	EVGPIO Mask Core #0 (0-63)
0x8000403a	15:0	EVGPIO Data Core #1 (64-127)
0x8000403c	15:0	EVGPIO Mask Core #1 (64-127)

Setting a pin direction, output value, and reading input changes are accessed through the EVPGIO data register.

EVGPIO Data Register

Bits	Description
15:9	Reserved (Write 0)
8	Valid Read Data [1]
7	Value
6	Data/$\overline{\text{Data Direction}}$ [2]
5:0	DIO number

1. ↑ When writing, write 0. During a read this indicates if this read includes new valid changes. After an interrupt this register should be read until it returns 0.
2. ↑ When bit6 = 0, data direction of DIO is set to value (bit7 1 = output, 0 = input). When bit6 = 1, the data output of DIO is set to value

The second register is the IRQ mask. This is used to set which DIO will update the data register and trigger an IRQ on change.

EVGPIO Mask Register

Bits	Description
15:8	Reserved (Write 0)
7	Mask Set (0 = irq enabled, 1 = irq disabled)
6	Reserved (Write 0)
5:0	DIO number

This is a list of the DIO for using with evgpioctl or evgpio.c/h.

IO Number	Location	Alternate Function
0	PC104_B14	ISA_IOR
1	PC104_B13	ISA_IOW
2	PC104_B11	ISA_MEMW
3	PC104_B12	ISA_MEMR
4	PC104_B02	ISA_RESET
5	PC104_B30	ISA_14_3_MHZ [1]
6	PC104_B08	ISA_ENDX
7	PC104_A11	ISA_AEN
8	PC104_B19	ISA_REFRESH
9	PC104_A01	ISA_IOCHK
10	PC104_A10	ISA_IORDY
11	PC104_B16	ISA_DRQ3
12	PC104_B06	ISA_DRQ2
13	PC104_B15	ISA_DACK3
14	PC104_B23 [2]	ISA_IRQ5
15	PC104_B22 [2]	ISA_IRQ6
16	PC104_B21 [2]	ISA_IRQ7
17	PC104_B04	ISA_IRQ9
18	PC104_B17	ISA_DACK1
19	PC104_B18	ISA_DRQ1
20	PC104_B25	ISA_IRQ3
21	PC104_B20	ISA_BCLK
22	PC104_B26	ISA_DACK2
23	PC104_B27	ISA_TC
24	PC104_B28	ISA_BALE
25	PC104_A09	ISA_DAT_00 [3]
26	PC104_A08 [3]	ISA_DAT_01
27	PC104_A04 [3]	ISA_DAT_02
28	PC104_A07 [3]	ISA_DAT_03
29	PC104_A06 [3]	ISA_DAT_04
30	PC104_A05 [3]	ISA_DAT_05
31	PC104_A03 [3]	ISA_DAT_06
32	PC104_A02 [3]	ISA_DAT_07
33	PC104_C11	ISA_DAT_08
34	PC104_C12	ISA_DAT_09
35	PC104_C13	ISA_DAT_10
36	PC104_C14	ISA_DAT_11
37	PC104_C15	ISA_DAT_12
38	PC104_C16	ISA_DAT_13
39	PC104_C17	ISA_DAT_14
40	PC104_C18	ISA_DAT_15
41	PC104_A31	ISA_ADD_00
42	PC104_A30	ISA_ADD_01

43	PC104_A29	ISA_ADD_02
44	PC104_A28	ISA_ADD_03
45	PC104_A27	ISA_ADD_04
46	PC104_A26	ISA_ADD_05
47	PC104_A25	ISA_ADD_06
48	PC104_A24	ISA_ADD_07
49	PC104_A23	ISA_ADD_08
50	PC104_A22	ISA_ADD_09
51	PC104_A21	ISA_ADD_10
52	PC104_A20	ISA_ADD_11
53	PC104_A19	ISA_ADD_12
54	PC104_A18	ISA_ADD_13
55	PC104_A17	ISA_ADD_14
56	PC104_A16	ISA_ADD_15
57	PC104_A15	ISA_ADD_16
58	PC104_A14	ISA_ADD_17
59	PC104_A13	ISA_ADD_18
60	PC104_A12	ISA_ADD_19
61	ISA_DIR	Toggles DIR on U33
62	ISA_DATA_EN	Toggles OE on U33
63	ISA Memory Enable [4]	Controls ISA 16-bit/mem cycles [5]
64	LCD_05	LCD_EN
65	LCD_06	LCD_WR#
66	LCD_03	LCD_RS
67	LCD_04	LCD_BIAS
68	LCD_14	LCD_DATA6
69	LCD_13	LCD_DATA7
70	LCD_12	LCD_DATA4
71	LCD_11	LCD_DATA5
72	LCD_10	LCD_DATA2
73	LCD_09	LCD_DATA3
74	LCD_08	LCD_DATA0
75	LCD_07	LCD_DATA1
76	DIO_01 [6]	N/A
77	DIO_03	N/A
78	DIO_05	N/A
79	DIO_07	N/A
80	DIO_09	N/A
81	DIO_11	N/A
82	DIO_13	N/A
83	DIO_15	N/A
84	DIO_08	N/A
85	DIO_04	EN_DIO_FET
86	Reserved	N/A
87	PC104_D01	N/A
88	PC104_D02	N/A
89	PC104_D03	N/A
90	PC104_D04	N/A
91	PC104_D05	N/A
92	PC104_D06	N/A
93	PC104_D07	N/A
94	PC104_D08	N/A
95	PC104_D09	N/A

96	PC104_D10	N/A
97	PC104_D11	N/A
98	PC104_D12	N/A
99	PC104_D13	N/A
100	PC104_D14	N/A
101	PC104_D15	N/A
102	DB9_07 [7]	RTS
103	DB9_04 [7]	DTR
104	COM2_07 [7]	RTS
105	COM3_07 [7]	RTS
106	DB9_08 [2]	XUART5 CTS
107	DB9_06 [2]	DSR
108	DB9_01 [2]	DCD
109	COM2_08 [2]	CTS
110	COM3_08 [2]	CTS
111	JP2 [2]	Controls DB9 Console (on=cpu debug, off=xuart5)
112	JP3# [2]	Controls Boot device (on=SD, off=eMMC)
113	M0 spare In [2]	N/A
114	M0 spare Out [7]	N/A

1. ↑ This I/O is not usable as an input. Setting the I/O to 1 will output the 14.3MHz clock, and 0 will disable the clock.
2. ↑ 2.00 2.01 2.02 2.03 2.04 2.05 2.06 2.07 2.08 2.09 2.10 Input Only
3. ↑ 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7 These pins use a 245 buffer for 5V tolerance, but the direction must be manually set when using these pins as DIO. Toggle direction with #61.
4. ↑ This pin is not brought out, only used internal to the FPGA
5. ↑ When the output is 1, this will cause writes to the PC104 ranges to use MEMR/MEMR instead of IOR/IOW. IO mode is assumed by default. When the output enable is 1, this will allow 16-bit transactions on the 0x80008000 range, and 8-bit in 0x81008000 (default). When the output enable is low, 8-bit transactions work in the 0x80008000 range with their addresses doubled.
6. ↑ External signal brought low can wake from sleep mode
7. ↑ 7.0 7.1 7.2 7.3 7.4 Output Only

6.11 USB

6.11.1 USB OTG

This board features USB OTG which is wired to support a second USB host port. The Marvell OTG driver has a known issue where it does not properly support hotplugged USB devices. If a device is plugged in during boot it will work, but if it is plugged in later you must manually run:

```
echo 1 > /proc/driver/otg
```

6.11.2 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.

6.12 SPI

The SPI controller is implemented in the FPGA. This is commonly accessed using tsctl, or by accessing the registers directly. This core is found at 0x80004800, and should only be accessed using 16-bit reads/writes.

The table below is the register map for the SPI in the FPGA:

Offset	Access	Bit(s)	Description
0x0	Read Only	15	SPI MISO state
	Read/Write	14	SPI CLK state
	Read/Write	13:10	Speed - 0 (highest), 1 (1/2 speed), 2 (1/4 speed)...
	Read/Write	9:8	LUN (0-3 representing the 4 chip selects)
	Read/Write	7	CS (1 - CS# is asserted)
	N/A	6:1	Reserved
	Read/Write	0	Speed
0x2	Read Only	15:0	Previous SPI read data from last write
0x4	N/A	15:0	Reserved
0x6	N/A	15:0	Reserved
0x8	Read/Write	15:0	SPI read/write with CS# to stay asserted
0xa	Read Only	15:0	SPI pipelined read with CS# to stay asserted
0xc	Read/Write	15:0	SPI Read/Write with CS# to deassert post-op
0xe	N/A	15:0	Reserved

The SPI clk state register should be set when CS# is deasserted. Value 0 makes SPI rising edge (CPOL=0), 1 is falling edge (CPOL=1). This only applies to speed >= 1.

Where the base clock is 75Mhz (extended temp alters this to 50Mhz), speed settings break down as follows:

Value	Speed
0	75Mhz
1	37.5MHz
2	18.75MHz
3	12.5MHz
4	9.375MHz
5	7.5MHz
6	6.25MHz
7	5.36MHz
8	4.68MHz
9	4.17MHz
15	2.5MHz
19	1.97MHz
31	1.21MHz

The pipelined read register is for read bursts and will automatically start a subsequent SPI read upon completion of the requested SPI read. Reading from this register infers that another read will shortly follow and allows this SPI controller "a head start" on the next read for optimum read performance. This register should be accessed as long as there will be at least one more SPI read with CS# asserted to take place.

6.13 FPGA

All macrocontrollers feature an FPGA. Any external interfaces called for by the TS-SOCKET specification that are not provided by the CPU are implemented in the FPGA whenever possible. The FPGA is connected to the CPU by a static memory controller, and as a result the FPGA can provide registers in the CPU memory space.

While most common functionality is accessed through layers of software that are already written, some features may require talking directly to the FPGA. Access to the FPGA is done through either the 8-bit or 16-bit memory regions. Code should access 16-bit or 8-bit depending on the access designed for the specific hardware core. For example, the CAN core is 8 bit, the 8 bit MUXBUS space is 8 bit, and some 8 bit cycles are needed for the SPI core if you want to do 8 bit SPI transactions. To access hardware cores in the FPGA, add the offset in the table below to the base address.

Bit Width	Base Address
16	0x80000000
8	0x81000000

Offset	Usage	Bit Width
0x0000	16KB blockram access (for XUART buffer)	16
0x4000	Syscon registers	16
0x4400	ADC registers (for off-board ADC)	16
0x4800	SPI interface	16
0x4C00	CAN controller	8
0x4D00	2nd CAN controller	8
0x5000	Touchscreen registers	16
0x5400	XUART IO registers	16
0x8000	32KB MUXBUS space	16/8

6.13.1 FPGA Bitstreams

The FPGA has the capability to be reloaded on startup and reprogram itself with different configurations. The default bitstream is hardcoded into the FPGA, but the soft reloaded bitstreams can be placed in /ts/ts<model>-fpga.vme.gz on the Debian root to make the board load the bitstream on startup. If we do not have a configuration you need, you can build a new bitstream, or contact us (http://www.embeddedarm.com/support/contact-us.php) for our engineering services.

6.13.2 FPGA Programming

Note: The TS-7250-V2 does not yet have a released opencore. Send an email to support@embeddedarm.com to request a notification when this is released.

(work in progress, updates soon!!!)

6.14 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 0x80004000. For example, to identify the TS-7250-V2:

```
devmem 0x80004000 16
```

This will return 0x7250 to read back the model ID.

Note: This syscon is mostly shared between the TS-4700, TS-4710, TS-4712, TS-4720, TS-4740, and TS-7700, though DIO is handled differently between boards.

Many of the syscon options can be manipulated using tshwctl.

```
Usage: tshwctl [OPTION] ...
Technologic Systems TS-471x / TS-77XX FPGA manipulation.

General options:
  -g, --getmac            Display ethernet MAC address
  -s, --setmac=MAC        Set ethernet MAC address
  -R, --reboot            Reboot the board
  -t, --getrtc            Get system time from RTC time/date
  -S, --setrtc            Set RTC time/date from system time
  -F, --rtcinfo           Print RTC temperature, poweron/off time, etc
  -v, --nvram             Get/Set RTC NVRAM
  -i, --info              Display board FPGA info
  -e, --greenledon        Turn green LED on
  -b, --greenledoff       Turn green LED off
  -c, --redledon          Turn red LED on
  -d, --redledoff         Turn red LED off
  -D, --setdio=<pin>      Sets DDR and asserts a specified pin
  -O, --clrdio=<pin>      Sets DDR and deasserts a specified pin
  -G, --getdio=<pin>      Sets DDR and gets DIO pin input value
  -x, --random            Get 16-bit hardware random number
  -W, --watchdog          Daemonize and set up /dev/watchdog
  -n, --setrng            Seed the kernel random number generator
  -X, --resetswitchon     Enable reset switch
  -Y, --resetswitchoff    Disable reset switch
  -l, --loadfpga=FILE     Load FPGA bitstream from FILE
  -q, --cputemp           Display the CPU die temperature
  -U, --removejp=JP       Remove soft jumper numbered JP (1-8)
  -J, --setjp=JP          Set soft jumper numbered JP (1-8)
  -k, --txenon=XUART(s)   Enables the TX Enable for an XUART
  -K, --txenoff=XUART(s)  Disables a specified TX Enable
  -N, --canon=PORT(s)     Enables a CAN port
  -f, --canoff=PORT(s)    Disables a CAN port
  -h, --help              This help
  -E, --bbclk2on          Enables a 25MHz clock on DIO 34
  -I, --bbclk2off         Disables the 25MHz clock
  -r, --touchon           Turns the touchscreen controller on
  -T, --touchoff          Turns the touchscreen controller off
```

```
  -B, --baseboard         Display baseboard ID
  -a, --adc               Display MCP3428 ADC readings in millivolts
  -P, --ethvlan           Configures a network switch to split each port individually in a vlan
  -y, --ethswitch         Configures a network switch to switch all of the outside ports to one interface
  -e, --ethwlan           Configures the first network port (A) to its own VLAN, and all other ports to a shared switch
  -C, --ethinfo           Retrieves info on the onboard switch
```

Offset	Bits	Usage
0x00	15:0	Model ID: Reads 0x7250
0x02	15:9	Reserved
	8	Enable DB9 console (Overrides JP2)
	7:6	Scratch Register
	5	Mode2
	4	Mode1
	3:0	FPGA revision
0x04	15:0	Reserved
0x06	15:0	Watchdog feed register
0x08	15:0	Free running 1MHz counter LSB
0x0a	15:0	Free running 1MHz counter MSB
0x0c	15:0	Hardware RNG LSB
0x0e	15:0	Hardware RNG MSB
0x28	15:4	Reserved
	3:0	FPGA TAG memory access [1]
0x2a	15:0	Custom load ID register [2]
0x2c	15	EVGPIO Core 2
	14	EVGPIO Core 1
	13:6	Reserved
	5	Offboard IRQ 7
	4	Offboard IRQ 6
	3	Offboard IRQ 5
	2	CAN2 IRQ
	1	CAN IRQ
	0	XUART IRQ
0x2e	15:6	Reserved
	5	Offboard IRQ 7 mask (1 disabled, 0 on) [3]
	4	Offboard IRQ 6 mask (1 disabled, 0 on) [3]
	3	Offboard IRQ 5 mask (1 disabled, 0 on)[3]
	2	CAN2 IRQ mask (1 disabled, 0 on)[3]
	1	CAN IRQ mask (1 disabled, 0 on)[3]
	0	XUART IRQ mask (1 disabled, 0 on)[3]
0x34	15:7	Reserved
	6	Analog A2
	5	Analog A1
	4	Analog A0
	3	ADC SCL
	2	ADC SDA
	1	ADC SCL Input
	0	ADC SDA Input
0x36	15:0	EVGPIO Core #0
0x38	15:0	EVGPIO Mask Core #0
0x3a	15:0	EVGPIO Core #1
0x3c	15:0	EVGPIO Mask Core #1
0x3e	15:4	Reserved
	3:0	LCD PWM Duty Cycle (0x7 = 100%)

1. ↑ TAG memory stores persistent data on the FPGA such a the MAC address, CPU settings, and the born on date. Software using this data should instead use tshwctl rather than accessing this register manually.
2. ↑ Reads back 0 on default load. Used to identify customized bitstreams
3. ↑ 3.0 3.1 3.2 3.3 3.4 3.5 The IRQ masks are handled automatically by the kernel after an IRQ is requested. Under most circumstances these registers should not be manipulated.

6.15 Watchdog

By default there is a /dev/watchdog with the tshwctl daemon running at the highest possible priority to feed the watchdog. This is a pipe that is created in userspace, so for many applications this may provide enough functionality for the watchdog by verifying that userspace is still executing applications. If you would like to have the watchdog functionality more tightly integrated with your application you can specify various feed options.

At the lower level there are 3 valid watchdog feed values that are written to the watchdog register in the #Syscon:

Value	Result
0	feed watchdog for another .338s
1	feed watchdog for another 2.706s
2	feed watchdog for another 10.824s
3	disable watchdog

The watchdog is armed by default for 10s for the operating system to take over, after which the startup scripts autofeed the watchdog with:

```
echo a2 > /dev/watchdog
```

The /dev/watchdog fifo accepts 3 types of commands:

Value	Function
f<3 digits>	One time feed for a specified amount of time which uses the 3 digit number / 10. For example, "f456" would feed for 45.6 seconds.
"0", "1", "2", "3"	One time feed with the value in the above table.
a<num 0-3>	This value autofeeds with the value in the above table.

Most applications should use the f<3 digits> option to more tightly integrate this to their application. For example:

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

void do_some_work(int data) {
        /* The contract for sleep(int n) is that it will sleep for at least n
         * seconds, but not less.  If other kernel threads or processes require
         * more time sleep can take longer, but when your process has a high
         * priority this is usually measured in millseconds */
        sleep(5);
}

int read_some_io() {
        /* If this function (or do_some_work) misbehave and stall thee watchdog
         * will not be fed in the main loop and cause a reboot.  You can test
         * this by uncommenting the next line to force an infinite loop */
        // while (1) {}
        return 42;
}

int main(int argc, char **argv)
{
        int wdfd;
        /* In languages other than C/C++ this is still essentially the same, but
         * make sure you are opening the watchdog file synchronously so the writes
         * happen immediately.  Many languages will buffer writes together to make
         * them more efficient, but the watchdog needs the writes to be timed
         * precisely */
        wdfd = open("/dev/watchdog", O_SYNC|O_RDWR);

        while (1) {
                int data;
                /* This loop is expected to take about 5-6 seconds, but to allow some
                 * headroom for other applications, I will feed the watchdog for 10s. */
                write(wdfd, "f100", 4);

                data = read_some_io();
                do_some_work(data);
        }
}
```

6.16 PC104

To access peripherals on the PC104 bus it is necessary to add the base address from the table below to the offset of the peripheral to get a memory address for accessing the peripheral. For example, for ISA 8-bit I/O address 0x100, add 0x8108000 to 0x100 to get 0x81008100.

	Memory	I/O
8-bit	N/A	0x81008000-0x8100ffff
16-bit	N/A	0x80008000-0x8000ffff

Alternatively, if your peripheral requires more memory space you can switch to an alternate layout:

```
devmem 0x80004036 16 0x00bf
```

	Memory	I/O
8-bit	0x81008000-0x8100ffff	0x81007000-0x81007fff
16-bit	0x80008000-0x8000ffff	0x80007000-0x80007fff

Using either layout, IRQs 5, 6, and 7 on the PC104 bus from the TS-7250-V2 are 62 + the PC104 IRQ number. These will be IRQs 67, 68, and 69.

To enable "classic" 16 bit PC104 (for use only with non-TS 16 bit PC104 peripherals), set the alternate PC104 configuration register thus: `devmem 0x80004036 16 0xFE`

6.16.1 PC104 ISA16550

You can use the included ts4700_isa16550 driver to load support for various devices such as the TS-IRIDIUM, or TS-MULTI-104.

For example, to load a single device:

```
# Assumes COM1 and IRQ7 jumpers are set
modprobe ts4700_isa16550 com=0x3f8 irq=7
```

If you are loading multiple devices, you can specify the COM and IRQ in a single command. For example, to set up a TS-SER4 with only jumpers IRQ4, IRQ2, and COM1 set:

```
modprobe ts4700_isa16550 irq=6,6,6,6 com=0x3f8,0x2f8,0x3e8,0x2e8
```

This driver assumes the PC104 base is at 0x0 of the muxbus, but some baseboards such as the TS-8900 use another offset for PC104. This can be specified with the iobase argument:

```
modprobe ts4700_isa16550 com=0x3f8 irq=7 iobase=0x81008800
```

6.17 XUARTS

The XUARTs are ttl serial ports implemented in the FPGA. These communicate with the userspace driver xuartctl. Each XUART core in the FPGA can handle up to 8 XUARTs, though the default TS-4710 FPGA contains 6. The XUART serial ports have a single shared 4kByte receive FIFO which makes real time interrupt latency response less of a concern and in actual implementation, the serial ports are simply polled at 100Hz and don't even use an IRQ. Even with all 8 ports running at 230400 baud, it is not possible to overflow the receive FIFO in 1/100th of a second. The "xuartctl --server" daemon is started by default in the init scripts which sets up listening TCP/IP ports for all XUART channels on ports 7350-7357. An application may simply connect to these ports via localhost (or via the network) and use the serial ports as if they were network services.

The typical method for accessing xuarts is using the pts layer. For example:

```
eval $(xuartctl --server --port 3 --mode=8n1 --speed 9600 2>&1); ln -s $ttyname /dev/ttyxuart3
```

This will set up XUART port 3 to 9600 baud, 8n1, and symlink it to /dev/ttyxuart3. In your application you can open the /dev/ttyxuart3 and for most part you can access this just like any other uart. When using the PTS layer, there are several operations that are not supported. The mode and baud rate must be set up with xuartctl, and cannot be programatically changed with the standard ioctl.

The XUARTs can be managed with xuartctl. See the xuartctl page for more details on programming with XUARTs. See either of these links for more information on using serial ports in Linux:

- 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)

6.18 CAN

CAN is implemented in the TS-7250-V2 hardware in all currently shipping TS-7250-V2. Software support is pending internal review. This wiki entry is currently under construction, intended to assist those developers who require CAN before the officially supported image is released.

To enable CAN support, the kernel must be rebuilt to include the sja1000_isa CAN driver. Following the kernel build instructions above, during the "make menuconfig" step, set "M" through this tree: -> Networking support (NET [=y]) -> CAN bus subsystem support (CAN [=m]) -> CAN Device Drivers -> Platform CAN drivers with Netlink support (CAN_DEV [=m]) -> Philips/NXP SJA1000 devices (CAN_SJA1000 [=m])

At the bottom of this tree, be sure to set M in both "TS-CAN1 PC104 boards" and "ISA Bus based legacy SJA1000 driver"

During startup, you will need to include this command to load the driver:

```
modprobe sja1000_isa mem=0x81004c00 irq=65 clk=24000000
```

Before using the CAN network interface, you will need to bring it up in the normal manner:

```
ip link set can0 type can bitrate 125000

ifconfig can0 up
```

From there the interface is up and available for your usage.

6.19 Power Consumption

The TS-7250-V2's power consumption can vary a lot depending on the build and activity of the board. Most of the power savings happens automatically when the cpu is idle. These tests are performed with the Kernel 2.6 based image which uses dynamic ticks, whereas the Kernel 3.14 image is optimized for low latency and uses a 1000khz tick which will result power consumption with a smaller gap between idle and 100% cpu usage.

These tests do not include Ethernet or other peripherals unless otherwise stated. These tests are also performed using the 5V input.

TS-7250-V2-512-4096F-10S-I-DEV

Test	Max (W)	Average (W)	Min (W)
Idle CPU	2.424	1.64	1.41
100% CPU	2.58	2.27	1.97

TS-7250-V2-512-8S-I

Test	Max (W)	Average (W)	Min (W)
Idle CPU	1.73	1.21	1.00
100% CPU	1.86	1.65	1.49
Idle CPU, Ethernet connected	1.92	1.32	1.12

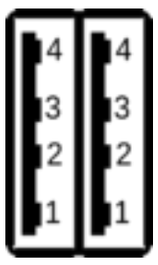
7 External Interfaces

7.1 Jumpers

Jumper	Function
1	Reserved
2	DB9 console (on=ttyS0, off=xuart4)
3	Boot device (on=SD, off=eMMC)
4	Reserved

7.2 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

Note: The PXA16X OTG port (bottom USB) does not support automatic enumeration when hotplugging devices. Refer to the #USB OTG section for more details.

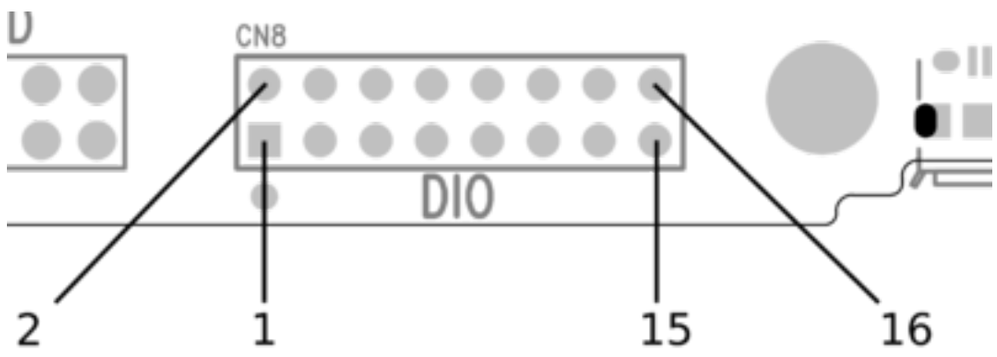
7.3 DIO header

The TS-7250-V2 includes a 2x8 0.1" pitch header with 8 DIO, #SPI, and a controllable large current sink. The DIO are accessed in software using #EVGPIO.

Pinout

Pin	Name	EVGPIO
1 [1]	DIO_01	76
2	Ground	N/A
3	DIO_03	77
4	40V_FET	85
5	DIO_05	78
6	SPI_CS#	N/A
7	DIO_07	79
8	DIO_08	84
9	DIO_09	80
10	SPI_MISO	N/A
11	DIO_11	81
12	SPI_MOSI	N/A
13	DIO_13	82
14	SPI_CLK	N/A
15	DIO_15	83
16	3.3V	N/A

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>
#include "evgpio.h"

#define DIO_01  76
#define DIO_03  77
#define DIO_05  78
#define DIO_07  79
#define DIO_09  80
#define DIO_11  81
#define DIO_13  82
#define DIO_15  83

void dio_out(uint8_t out)
{
    evsetdata(DIO_01, out & (1 << 0));
    evsetdata(DIO_03, out & (1 << 1));
    evsetdata(DIO_05, out & (1 << 2));
    evsetdata(DIO_07, out & (1 << 3));
    evsetdata(DIO_09, out & (1 << 4));
    evsetdata(DIO_11, out & (1 << 5));
    evsetdata(DIO_13, out & (1 << 6));
    evsetdata(DIO_15, out & (1 << 7));
}

void dio_ddr(uint8_t out)
{
    evsetddr(DIO_01, out & (1 << 0));
    evsetddr(DIO_03, out & (1 << 1));
    evsetddr(DIO_05, out & (1 << 2));
    evsetddr(DIO_07, out & (1 << 3));
    evsetddr(DIO_09, out & (1 << 4));
    evsetddr(DIO_11, out & (1 << 5));
    evsetddr(DIO_13, out & (1 << 6));
    evsetddr(DIO_15, out & (1 << 7));
}

uint8_t dio_read()
{
    uint8_t out = 0;
    out |= (evgetin(DIO_01) << 0);
    out |= (evgetin(DIO_03) << 1);
    out |= (evgetin(DIO_05) << 2);
    out |= (evgetin(DIO_07) << 3);
    out |= (evgetin(DIO_09) << 4);
    out |= (evgetin(DIO_11) << 5);
    out |= (evgetin(DIO_13) << 6);
    out |= (evgetin(DIO_15) << 7);
    return out;
}
```

```c
int main()
{
    uint8_t in;
    int row, col;
    char *keys[4][4] = {
        { "1", "2", "3", "UP" },
        { "4", "5", "6", "DOWN" },
        { "7", "8", "9", "2ND" },
        { "CLEAR", "0", "HELP", "ENTER" }
    };
    evgpioinit();

    dio_ddr(0x0f);

    while(1) {
        for(row = 0; row < 4; row++) {
            dio_out(~(1 << row));
            usleep(50000);
            in = dio_read();
            for(col = 4; col < 8; col++) {
                if(~in & (1 << col)) {
                    // If we read it, sleep and read again to debounce
                    usleep(1000);
                    in = dio_read();
                    if(~in & (1 << col)) {
                        printf("%s\n", keys[row][col - 4]);
                        fflush(stdout);
                    }
                }
            }
        }
    }

    return 0;
}
```

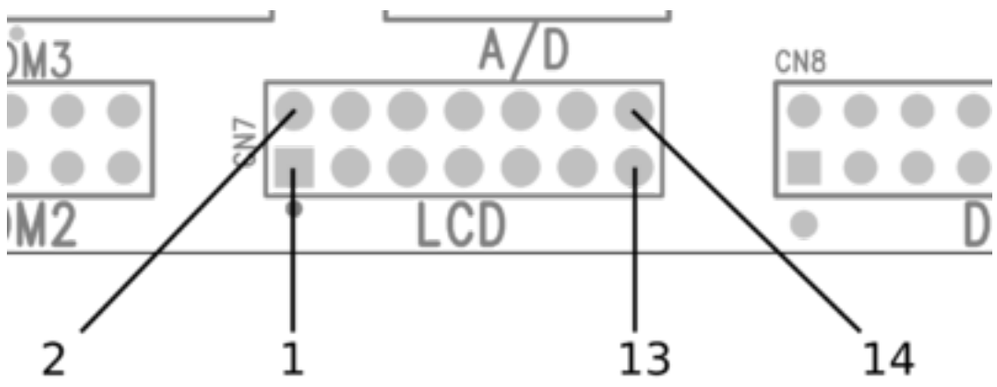
7.4 LCD Header

The LCD header is designed around compatibility with the common HD44780 LCD controller. We supply and support the low cost LCD-LED: Alphanumeric 2x24 LCD. These IO are accessed using #EVGPIO. Connector CN8 is a 14 pin (2x7) 0.1" spacing header.

Note: Other HD44780 LCD products may work with this header, but some of the control lines only go to 3.3V for compatibility with TTL lcd modules. While all of the LCD input pins are 5V tolerant, 5V CMOS modules will not be compatible.

Pinout **Header**

Pin	Name	EVGPIO Number
1	LCD_5V [2]	N/A
2	Ground	N/A
3	LCD_RS	66
4	LCD_BIAS	67
5	LCD_EN	64
6	LCD_WR#	65
7	LCD_D1	75
8	LCD_D0	74
9	LCD_D3	73
10	LCD_D2	72
11	LCD_D5	71
12	LCD_D4	70
13	LCD_D7	69
14	LCD_D6	68



1. ↑ When the Sleep mode is activated this pin can be asserted to wake up the board rather than waiting for the timed delay.
2. ↑ Provides up to 1400mA

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

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:


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

#include "evgpio.h"

void lcd_init(void);
void lcd_wait(void);
void lcd_command(uint16_t);
void lcd_writechars(unsigned char *dat);

// These are microsecond delays
#define SETUP   1
#define PULSE   2
#define HOLD    1

#define LCD_D0      74
#define LCD_D1      75
#define LCD_D2      72
#define LCD_D3      73
#define LCD_D4      70
#define LCD_D5      71
#define LCD_D6      68
#define LCD_D7      69
#define LCD_EN      64
#define LCD_WR      65
#define LCD_RS      66
#define LCD_BIAS    67

void lcd_write(uint8_t out)
{
    evsetdata(LCD_D0, out & (1 << 0));
    evsetdata(LCD_D1, out & (1 << 1));
    evsetdata(LCD_D2, out & (1 << 2));
    evsetdata(LCD_D3, out & (1 << 3));
    evsetdata(LCD_D4, out & (1 << 4));
    evsetdata(LCD_D5, out & (1 << 5));
    evsetdata(LCD_D6, out & (1 << 6));
    evsetdata(LCD_D7, out & (1 << 7));
}

void lcd_ddr(uint8_t out)
{
    evsetddr(LCD_D0, out & (1 << 0));
    evsetddr(LCD_D1, out & (1 << 1));
    evsetddr(LCD_D2, out & (1 << 2));
    evsetddr(LCD_D3, out & (1 << 3));
    evsetddr(LCD_D4, out & (1 << 4));
    evsetddr(LCD_D5, out & (1 << 5));
    evsetddr(LCD_D6, out & (1 << 6));
    evsetddr(LCD_D7, out & (1 << 7));
}

uint8_t lcd_read()
{
    uint8_t out = 0;
    out |= (evgetin(LCD_D0) << 0);
    out |= (evgetin(LCD_D1) << 1);
    out |= (evgetin(LCD_D2) << 2);
    out |= (evgetin(LCD_D3) << 3);
    out |= (evgetin(LCD_D4) << 4);
    out |= (evgetin(LCD_D5) << 5);
    out |= (evgetin(LCD_D6) << 6);
    out |= (evgetin(LCD_D7) << 7);
    return out;
}

void lcd_enpulse()
{
    usleep(SETUP);
    evsetdata(LCD_EN, 1);
    usleep(PULSE);
    evsetdata(LCD_EN, 0);
    usleep(HOLD);
}

void lcd_init(void)
{
    evgpioinit();
    // Data lines to inputs, control lines to outputs
    lcd_ddr(0x0);
    evsetddr(LCD_EN, 1);
    evsetddr(LCD_RS, 1);
    evsetddr(LCD_WR, 1);

    // Set LCD_EN and LCD_RS low
    evsetdata(LCD_EN, 0);
    evsetdata(LCD_RS, 0);
    // Set LCD_WR high
    evsetdata(LCD_WR, 1);
```

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

    evsetddr(LCD_BIAS, 1);
    evsetdata(LCD_BIAS, 0);
}

void lcd_wait(void)
{
    uint8_t in;
    int i, dat, tries = 0;
    lcd_ddr(0x0);
    do {
        // step 1, apply only RS & WR
        evsetdata(LCD_RS, 0);
        evsetdata(LCD_WR, 1); // low for write
        lcd_enpulse();
        usleep(1);
    } while (in & 0x80 && tries++ < 5);
}

void lcd_command(uint16_t cmd)
{
    lcd_ddr(0xff);
    lcd_write(cmd);
    evsetdata(LCD_WR, 0);
    evsetdata(LCD_RS, 0);

    lcd_enpulse();
}

void lcd_writechars(unsigned char *dat)
{
    int i;

    do {
        lcd_wait();
        lcd_ddr(0xff);
        evsetdata(LCD_RS, 1);
        evsetdata(LCD_WR, 0); // active low
        lcd_write(*dat++);
        lcd_enpulse();
     } while(*dat);
}

int main(int argc, char **argv)
{
    lcd_init();
    if (argc == 2) {
        lcd_writechars(argv[1]);
        return;
    }
    if (argc > 2) {
        lcd_writechars(argv[1]);
        lcd_wait();
        lcd_command(0xa8); // set DDRAM addr to second row
        lcd_writechars(argv[2]);
        return;
    }
    while(!feof(stdin)) {
        unsigned char buf[512];
        int i = 0;
        lcd_wait();
        if (i) {
            // XXX: this seek addr may be different for different
            // LCD sizes!  -JO
            lcd_command(0xa8); // set DDRAM addr to second row
        } else {
            lcd_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;
            lcd_writechars(buf);
        }
    }
    return 0;
}
```

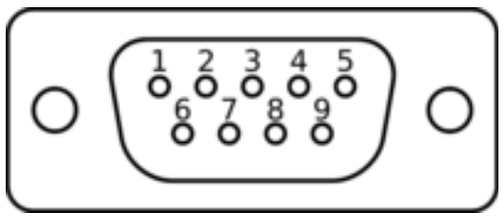
Compile this code with:

```
wget ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7700-linux/sources/evgpio.c
wget ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7700-linux/sources/evgpio.h
gcc evgpio.c lcdmesg.c -o lcdmesg
```

7.5 DB9

The DB9 connector brings out a single UART using either the XUART or CPU UART.

Pin	Name	Function or #EVGPIO
1	DCD	108
2	RXD	CONSOLE/XUART4 RXD [1]
3	TXD	CONSOLE/XUART4 TXD [1]
4	DTR	103
5	GND	N/A
6	DSR	107
7	RTS	102
8	CTS	106
9	RI	N/A



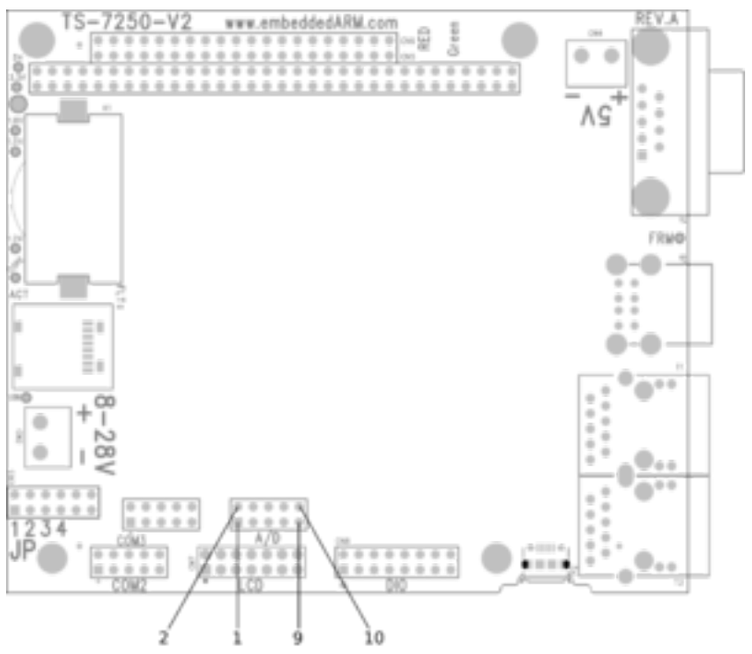
7.6 ADC Header

The Analog to Digital Converter header provides 5x 0-30V ADC inputs, or on 3 channels 4-20mA current sensors. This is a low speed ADC capable of about 3-4 samples per second, but with 16-bits of precision. These are available on CN10 which is a 10 pin (2x5) 0.1" spaced header. The connector layout and the signals carried by each pin are defined below.

Pinout

Pin	Function
1	adc0, 0-30V (default), 4-20mA (Toggle MFP_73 high)
2	GND
3	adc1, 0-30V (default), 4-20mA (Toggle MFP_74 high)
4	GND
5	adc2, 0-30V (default), 4-20mA (Toggle MFP_75 high)
6	GND
7	adc3, 0-30V
8	GND
9	adc4, 0-30V
10	GND

Header



7.7 COM Headers

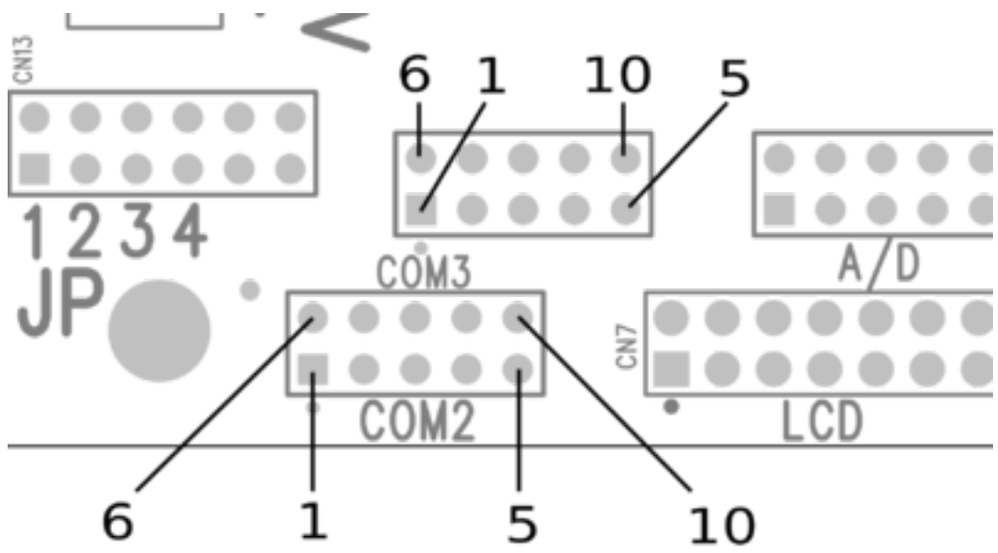
The TS-7250-V2 includes 2 2x5 0.1" pitch COM headers that feature RS232 and RS485/RS422 ports. These follow a different pin numbering which corresponds with a standard 10 pin header standard for UARTs. The RC-DB9 is available to convert these ports to a DB9.

Pinout

COM2 header

Pin	Name	#EVGPIO
1	XUART2 RS485+	N/A
2	XUART0 RS232 RXD	N/A
3	XUART0 RS232 TXD	N/A
4	XUART3 RS422+ RXD [2]	N/A
5	GND	N/A
6	XUART2 RS485-	N/A
7	COM2_RTS	104
8	COM2_CTS	109
9	XUART3 RS422- RXD [2]	N/A
10	Unused	N/A

Header



COM3 header

Pin	Name	#EVGPIO
1	Unused	N/A
2	XUART1 RS232 RXD	N/A
3	XUART1 RS232 TXD	N/A
4	Unused	N/A
5	GND	N/A
6	Unused	N/A
7	COM3_RTS	105
8	COM3_CTS	110
9	Unused	N/A
10	Unused	N/A

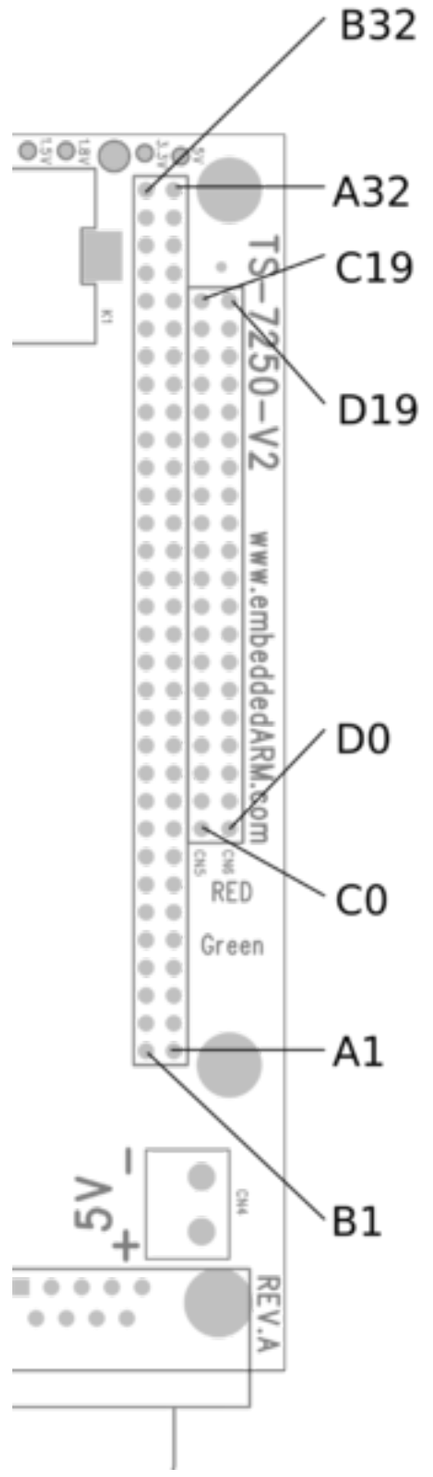
1. ↑ [1.0] [1.1] This either outputs XUART4, or the cpu's debug console depending on JP2, or the override in 0x2 in the #Syscon
2. ↑ [2.0] [2.1] This transceiver is only part of the FULL build

7.8 PC104 Header

The PC104 connector consists of four rows of pins labelled A-D. The PC104 standard implements an ISA bus over these headers, but we also include the ability for almost all of the pins to be used as DIO.

There is no mode switching required to use either function. The pins come up in the correct default state for PC104, and the appropriate pins involved in a bus cycle will be toggled if software initiates a read or write. The pins can be toggled any time using the #EVGPIO.

Pin	Name	EVGPIO	Pin	Name	EVGPIO	Pin	Name	EVGPIO	Pin	Name	EVGPIO
B32	GND	N/A	A32	GND	N/A						
B31	GND	N/A	A31	ISA_ADD_00	41						
B30	ISA_14_3_MHZ	5	A30	ISA_ADD_01	42						
B29	5V	N/A	A29	ISA_ADD_02	43						
B28	ISA_BALE	24	A28	ISA_ADD_03	44	C19	GND	N/A	D19	GND	N/A
B27	ISA_TC	23	A27	ISA_ADD_04	45	C18	ISA_DAT_15	40	D18	GND	N/A
B26	ISA_DACK2	22	A26	ISA_ADD_05	46	C17	ISA_DAT_14	39	D17	Unused	N/A
B25	ISA_IRQ3	20	A25	ISA_ADD_06	47	C16	ISA_DAT_13	38	D16	+5V	N/A
B24	GND	N/A	A24	ISA_ADD_07	48	C15	ISA_DAT_12	37	D15	ISA_DRQ7	101
B23	ISA_IRQ5	14	A23	ISA_ADD_08	49	C14	ISA_DAT_11	36	D14	ISA_DACK7	100
B22	ISA_IRQ6	15	A22	ISA_ADD_09	50	C13	ISA_DAT_10	35	D13	ISA_DRQ6	99
B21	ISA_IRQ7	16	A21	ISA_ADD_10	51	C12	ISA_DAT_09	34	D12	ISA_DACK6	98
B20	ISA_BCLK	21	A20	ISA_ADD_11	52	C11	ISA_DAT_08	33	D11	ISA_DRQ5	97
B19	ISA_REFRESH	8	A19	ISA_ADD_12	53	C10	Unused	N/A	D10	ISA_DACK5	96
B18	ISA_DRQ1	19	A18	ISA_ADD_13	54	C09	Unused	N/A	D09	ISA_DRQ0	95
B17	ISA_DACK1	18	A17	ISA_ADD_14	55	C08	Unused	N/A	D08	ISA_DACK0	94
B16	ISA_DRQ3	11	A16	ISA_ADD_15	56	C07	Unused	N/A	D07	ISA_IRQ14	93
B15	ISA_DACK3	13	A15	ISA_ADD_16	57	C06	Unused	N/A	D06	ISA_IRQ15	92
B14	ISA_IOR	0	A14	ISA_ADD_17	58	C05	N/A	Unused	D05	ISA_IRQ12	91
B13	ISA_IOW	1	A13	ISA_ADD_18	59	C04	Unused	N/A	D04	ISA_IRQ11	90
B12	ISA_MEMR	3	A12	ISA_ADD_19	60	C03	Unused	N/A	D03	ISA_IRQ10	89
B11	ISA_MEMW	2	A11	ISA_AEN	7	C02	Unused	N/A	D02	ISA_IO16	88
B10	GND	N/A	A10	ISA_IORDY	10	C01	Unused	N/A	D01	ISA_MEM16	87
B09	Unused	N/A	A09	ISA_DAT_00	25	C00	GND	N/A	D00	GND	N/A
B08	ISA_ENDX	6	A08	ISA_DAT_01	26						
B07	Unused	N/A	A07	ISA_DAT_03	28						
B06	ISA_DRQ2	12	A06	ISA_DAT_04	29						
B05	3.3V [1]	N/A	A05	ISA_DAT_05	30						
B04	ISA_IRQ9	17	A04	ISA_DAT_02	27						
B03	Unused	N/A	A03	ISA_DAT_06	31						
B02	ISA_RESET	4	A02	ISA_DAT_07	32						
B01	GND	N/A	A01	ISA_IOCHK	9						



1. ↑ The PC104 standard uses -5V here. If a third party device is used that might use this rail, FB17 should be removed.

8 Revisions and Changes

8.1 TS-7250-V2 PCB Revisions

Revision	Changes
A	▪ Initial Release
B	▪ Changed the uC from a NXP M0 to a SiLab. ▪ Instead of CDC-ACM this uses a signed windows driver CP210X which is preinstalled in most Linux distributsion. See here (https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx) for driver information. ▪ Added pull-down resistors to IRQ5-7 on the PC/104 bus ▪ Connected PC/104 pin B3 to 5V ▪ Connected PC/104 12V to VIN ▪ Changed full-size SD card to OST connector (180 degree reversed from old connector) ▪ D6 is not populated so USB does not power the entire board.

8.2 FPGA Changelog

FPGA Revision Log

Revision	Changes
9	▪ Connect RX line for XUART 4.
8	▪ Fix blockram arbiter for XUARTs.
7	▪ Workaround for TS-ADC16 ISA bus cycles (extra hold time on writes)
6	▪ At TS's special 16-bit ISA pin mapping on the PC104 64 pin connector and make it the default.
5	▪ Fix ISA 8-bit cycles at the 0x81xx_xxxx chip-select
4	▪ Mask 14mhz clk into EVGPIO core ▪ Make IO PC104 bus cycles the default, evgpio bit must be used to enable MEMR/W ▪ Fix red/green LED reg bit positions to be compatible with 47xx
3	▪ Pullups on LCD/DIO/PC104 pins ▪ Console enable on DB9 via JP2 ▪ Split IO/MEM PC104 region ▪ soft JP #7 now keeps PXA168s at 800Mhz ▪ IO bus cycles now drive upper address lines to 0 ▪ Inverted isa_16bit_en bit so that the poweron default is enabled
2	EVGPIO IRQ fixup
1	Fixed ISA databus
0	Initial release

Note: The opencore update is still pending release for this update to work. For updates please send an email to support@embeddedarm.com

You can update to the latest FPGA by booting to Debian and running:

```
cd /ts/
wget ftp://oz.embeddedarm.com/ts-arm-sbc/ts-7250-v2-linux/binaries/ts-bitstreams/ts7250-fpga-latest.vme.bz2
```

The FPGA is loaded in to the FPGA SRAM on every load, so this file will need to exist for all future boots.

8.3 Software Images

8.3.1 2.6 Debian Changelog

This is the changelog for the software image which is shared from the TS-4710, TS-4712, TS-4720, TS-4740, TS-7700, and the TS-7250-V2.

2.6.34 Based Image

Image File	Changelog	Known Issues
2gbsd-471x-20130221.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130221.dd.bz2)	▪ Initial release	

	▪ Updates to Debian Wheezy
	▪ TS specific utilities exported to Debian
	  ▪ Including tshwctl, xuartctl, sdctl
	  ▪ Includes busybox for vconfig and devmem
	▪ Includes more baseboard support
	  ▪ TX EN is automatically enabled on known baseboards
	▪ LCD support
	  ▪ TS-TPC-8900 support added
	▪ Touchscreen supported added	

2gbsd-471x-20130221.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130425.dd.bz2)	▪ TS-TPC-8900 support added ▪ Initial driver for SocketCAN support added ▪ ts4700ctl revised to tshwctl for consistency among other products ▪ tshwctl adds simple support for FPGA DIO with --getdio --setdio --clrdio options ▪ tsrf2cf support added ▪ ts4700_isa16550 support added for pc104 peripherals like TS-SER4 or TS-MULTI104 ▪ smsc95xx ethernet support added ▪ Icewm removed. ▪ fullscreen-webkit added ▪ Command is ./fullscreen-webkit <url> to run a full screen browser ▪ Replaces icewm as default desktop ▪ Fixed FPGA Reload support with release of opencore ▪ Added new soft jumpers. ▪ JP2 & JP3 are baseboard specific. ▪ Controls support such as ethernet switch config ▪ JP4 enables read only mode ▪ JP5 disables network autoconfig from initramfs. This can still be set from Debian	▪ CAN Controller currently inaccesible ▪ Second SD card not supported in sdctl ▪ smsc95xx periodically does not set MAC address correctly (already fixed in next image) ▪ mdnsd is not correctly started ▪ Debian Wheezy boot speed not yet optimized
2gbsd-471x-20130515.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130515.dd.bz2)	▪ Updated kernel. See git for the full changelog (https://github.com/embeddedarm/linux-2.6.34-ts471x/commits/master). ▪ TS-8700 support ▪ TS-8280 support ▪ Added SocketCAN support and example userspace utilities (cansend, candump, etc) ▪ Fixed FPGA reload to be reliable ▪ mdnsd started correctly ▪ fixed JP1 causing serial console to break	▪ USB OTG not yet supported ▪ Second SD card not supported in sdctl
2gbsd-471x-20130522.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130522.dd.bz2)	▪ Updated kernel. See git for the full changelog (https://github.com/embeddedarm/linux-2.6.34-ts471x/commits/master). ▪ Added basic support for TS-4740 ▪ Fixed USB OTG support ▪ Added PCIe support	▪ Second SD card not supported in sdctl ▪ No audio support ▪ PCIe breaks kernel register interface to FPGA
2gbsd-471x-20130531.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130531.dd.bz2)	▪ Disabled PCIe pending fixes for SMC FPGA interface ▪ Added TS-8150 support. ▪ Added TS-8920 support.	▪ Marvell switch chip drops fragmented packets (TS-4712 only) ▪ Set MTU to 1501 on eth0 as a workaround. ▪ PCIe disabled pending fix. ▪ No Audio support ▪ Second SD card not supported in sdctl
	▪ Implemented default splash screen, see /ts/splash ▪ Audio support (sgtl5000, wm8750, sii9022) ▪ Audio startup noise added, see /ts/startup.wav ▪ AutoStart X11 in the initramfs ▪ Configure started apps with /ts/initramfs-xinit ▪ Default x session changed to icewm-lite for faster boot time ▪ ifplugd is no longer run when jp1 is set due to race condition ▪ Configure the network in Debian once you are booting there ▪ check-usb-update implemented ▪ Plug in a USB drive with 1 partition containing /tsinit which will automatically run. Used primarily for production. ▪ sdctl updated to support dual SD ▪ Start "nbd-client 127.0.0.1 7501 /dev/nbd1" to access second card ▪ Doublestore fixes added	

2gbsd-471x-20130806.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130806.dd.bz2)	▪ PCIe fixes included, PCIe enabled by default on suported macros with pxa168
	▪ compat-drivers included
	  ▪ Includes common wireless drivers
	▪ Marvell Switch Chip fixes added
	▪ Wheezy updated to latest in repository
	▪ Added support for both onboard/offboard switches
	  ▪ Used in cases such as TS-4720 + TS-8700
	▪ Added initial TS-4740 support
	  ▪ Uses Xilinx FPGA for reload, added spiflash tools
	  ▪ spiflashctl added
	  ▪ bin2coe added
	  ▪ new tshwctl commands
	  ▪ eMMC support
	▪ TS-4720 support added
	▪ Generates random MAC address if none is programmed.
	  ▪ Should only be used for production
	▪ Added Kexec support
	  ▪ Allows support for NFS/http/ftp/etc kernel and rootfs loading
	▪ Xuartctl defaults to 100hz instead of IRQ driven
	  ▪ IRQ behavior is specifically tuned for best latency, but requires high CPU
	▪ Debian Wheezy configured to continue booting in the event of a fsck error unless it is completely unrepairable
	▪ ts-sendsigs-omit script fixes so multiple nbd-clients or xuartctls are not killed early in shutdown
	▪ Root filesystem is now always /dev/rootfs rather than /dev/nbd0p2
	  ▪ TS-4740/TS-4720 can boot to /dev/nbd1p2, so the actual rootfs is automatically symlinked to /dev/rootfs
		▪ Unionfs crashes with a kernel oops when JP4 is enabled.

2gbsd-471x-20130815.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gbsd-471x-20130815.dd.bz2)	▪ TS-8700 switch reset race condition fixed
	▪ Fixed FPGA reload for TS-4712/20
	▪ tshwctl minor fixes
	  ▪ ethinfo overflow fixed
	  ▪ tagmem is now only written if value actually changed with setjp/removejp/setmac
		▪ Unionfs crashes with a kernel oops when JP4 is enabled.

4gbsd-471x-20131004.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-20131004.dd.bz2)	▪ Switched to EXT3 by default
	  ▪ Doublestore users should disable journaling to reduce writes vs old ext2
	▪ Image sized for already shipping 4GB MicroSD cards
	▪ TS-8400 support completed
	▪ ifpulgd now starts on switch interfaces correctly
	▪ /ts/config file create to allow for further configuration of the initramfs. See this file for more information.
	▪ Soft Jumpers 2,3,4,5 have been removed and are implemented in the /ts/config file which allows more than 8 settings
	  ▪ The config file allows enabling and configuring utilities like ifplugd, xuartctl, mdnsd, and more.
	  ▪ Read only jumper 4 removed due to bugs with unionfs.
	  ▪ Behavior of JP1 and JP8 are not changed
	  ▪ JP7 added to minimize initramfs initialization for fastest boot.
	▪ Reloading the TS-4712 and TS-4720 now resets the model number correctly after a soft reload
	▪ X11 in Debian started with correct HOME variable so a valid .Xauthority file is created
	  ▪ This allows DISPLAY=:0 to work, and fixes some dns resolution issues
	▪ /etc/init.d/motd updated to include additional information for debugging
	▪ tshwctl updated
	  ▪ TS-7700 EVGPIO (getdio/setdio/clrdio) added
	  ▪ --getdio is fixed which would previously change the direction for all DIO in the first register when used (DIO 0-15)
	  ▪ --rtcinfo implemented which provides more status information on the RTC
	▪ Kernel updated
		▪ Unionfs support disabled

	▪ Included commonly requested features in default config ▪ ▪ Bluetooth, ipv6, netfilter (iptables), bridging, and more device support. ▪ wm8750 support added for TS-8400 ▪ sgtl5000 supports an option to disable standby to prevent audible pops ▪ See git for the full changelog (https://github.com/embeddedarm/linux-2.6.34-ts471x/commits/master) .	
4gbsd-471x-20140306.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4720-linux/binaries/ts-images/4gbsd-471x-20140306.dd.bz2)	▪ Fixed rare reboot lockup ▪ Added Marvell switch chip errata fix for 10Mb/s ethernet ▪ Added minimum packet size for TS-4712/TS-4720 to work around corrupt FCS from switch chip ▪ Added TS-4720 support ▪ Added TS-4740 support	
4gbsd-471x-20140430.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4720-linux/binaries/ts-images/4gbsd-471x-20140430.dd.bz2)	▪ Initial TS-7250-V2 support ▪ Initramfs now prints boot device on TS-7250-V2 and TS-4720 ▪ Initramfs zeroconf/ipv4ll IP is now only used as a fallback if dhcp fails ▪ Fixes mdns resolving zeroconf to a system with no route to it ▪ Updated tshwctl with rtcinfo, and 7250v2 dio/adc support ▪ Updated sdctl for 7250v2 ▪ Debian updated with empty killprocs script ▪ Normal killprocs ignores debians list of processes not to kill which kills the nbd-client ▪ Empty killprocs will succeed these updates, but not restart processes	▪ USB WiFi modules not compiled with the kernel. USB WiFi will not work, and requires a kernel compile and rebuilding the modules as outlined in the Compile the Kernel section.
* 4gbsd-471x-20140724.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-20140724.dd.bz2)	▪ Fixed TS-7250-V2 ADCs ▪ Fixed 16550 driver for 7250v2 ▪ Fixed USB WIFI module regression ▪ eMMC on 7250v2 is now converted to SLC. ▪ new image available 2gb-emmc-slc-471x-latest.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gb-emmc-slc-471x-latest.dd.bz2) ▪ Enabled /dev/i2c interface in the default kernel ▪ Added new CFG options ▪ CFG_MUTE_CPU_UART=1 will completely disable output on /dev/ttyS0 after the bootrom messages ▪ CFG_XUART_CONSOLE_EN=<0-7> will use an XUART for console instead of /dev/ttyS0	* EVGPIO IRQ #2 requires a build from the latest kernel sources
* 4gbsd-471x-20140924.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-20140924.dd.bz2)	▪ Using older images if multiple copies of tshwctl are run at the same time which access tagmem, this can corrupt the mac address and the soft jumpers. In some cases this could also break the touchscreen loading random data as calibration. This new tshwctl includes locking that will prevent multiple copies from accessing tagmem simultaneously. ▪ Removed udev rule that allocates eth# to a mac address	
* 4gbsd-471x-20141013.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-20141013.dd.bz2)	▪ Switched sdctl to using Unix Domain Sockets instead of localhost tcp. ▪ This gets around a rare bug that would cause the kernel to drop packets on localhost under heavy congestion. ▪ This changes only involves a new kernel/initramfs and does not impact the debian filesystem. ▪ Fixed touchscreen regression ▪ On older images the calibration was slightly off in the bottom right corner of the TS-TPC-8390 ▪ Fixed in ts_lcd.c in the kernel ▪ Updated Debian for shellshock vulnerability. ▪ Older images can just "apt-get update && apt-get dist-upgrade"	
▪ 4gbsd-471x-20141027.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-		

4710-linux/binaries/ts-images/4gbsd-471x-20141027.dd.bz2) ▪ 2gb-emmc-slc-471x-20141027.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gb-emmc-slc-471x-20141027.dd.bz2)	▪ Fixed booting to lun1 ▪ Updated touchscreen calibration for all TPCs ▪ Added SLC eMMC image for TS-4720	
▪ 4gbsd-471x-20141031.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/4gbsd-471x-20141031.dd.bz2) ▪ 2gb-emmc-slc-471x-20141031.dd.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4710-linux/binaries/ts-images/2gb-emmc-slc-471x-20141031.dd.bz2)	▪ Fixed tshwctl tagmem regression	

8.3.2 3.14 Debian Changelog

3.14 Based Image

Image File	Changelog	Known Issues

* 4gbsd-471x-3x-20140828.dd.bz2 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7250-v2-linux/binaries/ts-images/4gbsd-471x-3x-20140828.dd.bz2)	▪ This is a beta image
	▪ New 3.14.16 kernel
	▪ Fixed ts_lcd which would incorrectly attempt to load calibration from tagmem which is no longer used.
	▪ Fixed udev bug which broke keyboards & mice on x11
	  ▪ Fixed with this on older systems:
	  ▪ echo "SUBSYSTEM==\"input\", ENV{ID_INPUT}==\"\", IMPORT{builtin}=\"input_id\"" >> /etc/udev/rules.d/50-udev-default.rules
		▪ USB OTG host does not work. The 1 CPU USB host still works.
		▪ PCIe, and intel I210 (TS-4740) support not yet functional.
		▪ Muxed IRQs (PC104, CAN, optionally XUART) currently do not work.

▪ 4gbsd-471x-3x-20141013.dd.bz2 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7250-v2-linux/binaries/ts-images/4gbsd-471x-3x-20141013.dd.bz2)
▪ 2gb-emmc-slc-471x-3x-20141013.dd.bz2 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7250-v2-linux/binaries/ts-images/2gb-emmc-slc-471x-3x-20141013.dd.bz2)
	▪ Switched sdctl to using Unix Domain Sockets instead of localhost tcp.
	  ▪ This gets around a rare bug that would cause the kernel to drop packets on localhost under heavy congestion.
	  ▪ This changes only involves a new kernel/initramfs and does not impact the debian filesystem.
	▪ Fixed touchscreen regression
	  ▪ On older images the calibration was slightly off in the bottom right corner of the TS-TPC-8390
	  ▪ Fixed in ts_lcd.c in the kernel
	▪ Updated Debian for shellshock vulnerability.
	  ▪ Older images can just "apt-get update && apt-get dist-upgrade"
		▪ No Audio
		▪ LCD still requires timing updates since vsync is off
		▪ USB OTG is not currently working

▪ 4gbsd-471x-3x-20141031.dd.bz2 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7250-v2-linux/binaries/ts-images/4gbsd-471x-3x-20141031.dd.bz2) ▪ 2gb-emmc-slc-471x-3x-20141031.dd.bz2 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7250-v2-linux/binaries/ts-images/2gb-emmc-slc-471x-3x-20141031.dd.bz2)	▪ More common devices included in default kernel ▪ USB OTG Fixed ▪ tshwctl tagmem regression fixed	▪ LCD timing still off, not noticable on 1GHz PXA168 ▪ No Audio

9 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)
- Doublestore (http://www.embeddedarm.com/about/resource.php?item=628)
- 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)

10 Product Notes

10.1 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.

10.2 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.