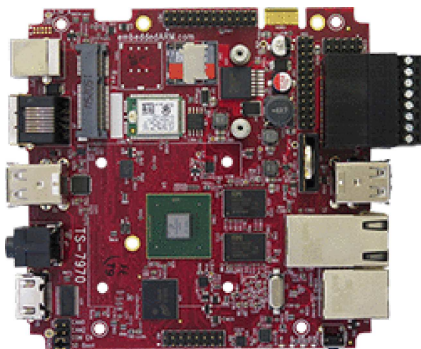


From Technologic Systems Manuals

TS-7970

TS-7970



Product Page (<http://www.embeddedarm.com/products/board-detail.php?product=TS-7970>)

Product Images (<https://www.embeddedarm.com/product-images/TS-7970>)

Specifications (<https://www.embeddedarm.com/products/TS-7970?tab=specs>)

Documentation

Schematic (<http://www.embeddedarm.com/documentation/ts-7970-schematic.pdf>)

Mechanical Drawing (<http://www.embeddedarm.com/documentation/ts-7970-mechanical.pdf>)

FTP Path (<ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7990-linux/>)

Processor

NXP i.MX6 Quad core, or Solo

i.MX6 Quad Product Page (<http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx-6quad-processors-high-performance-3d-graphics-hd-video-arm-cortex-a9-core:i.MX6Q>)

i.MX6 Solo Product Page (<http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors/i.mx-6-processors/i.mx-6solo-processors-single-core-multimedia-3d-graphics-arm-cortex-a9-core:i.MX6S>)

IMX6Q Reference Manual (http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6DQRM.pdf?

1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation)

IMX6S Reference Manual (http://cache.freescale.com/files/32bit/doc/ref_manual/IMX6SDLRM.pdf?

1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation)

Contents

- 1 Getting Started
 - 1.1 Getting Console and Powering up
 - 1.2 First Linux Boot
 - 1.3 Comparison of Distributions
- 2 U-Boot
 - 2.1 U-Boot Environment
 - 2.2 U-Boot Commands
 - 2.3 Modify Linux Kernel cmdline
 - 2.4 U-boot Recovery
 - 2.5 Linux NFS Boot
 - 2.6 Update U-Boot
 - 2.7 U-boot Development
 - 2.8 Access U-boot Environment from Linux
- 3 Debian
 - 3.1 Getting Started with Debian
 - 3.2 Debian Networking
 - 3.2.1 Debian WIFI Client
 - 3.2.2 Debian WIFI Access Point
 - 3.3 Debian Application Development
 - 3.3.1 Debian Jessie Cross Compiling
 - 3.4 Debian Installing New Software

- 3.5 Debian Setting up SSH
- 3.6 Debian Starting Automatically
- 4 Ubuntu
 - 4.1 Getting Started with Ubuntu
 - 4.2 Ubuntu Networking
 - 4.2.1 Ubuntu WIFI Client
 - 4.2.2 Ubuntu WIFI Access Point
 - 4.3 Ubuntu Installing New Software
 - 4.4 Ubuntu Setting up SSH
 - 4.5 Ubuntu Starting Automatically
- 5 Ubuntu Core
 - 5.1 Getting Started with Ubuntu Core
 - 5.2 Ubuntu Core Reference Links
- 6 Yocto
 - 6.1 Getting Started with Yocto
 - 6.2 Yocto Networking
 - 6.2.1 Yocto Wireless
 - 6.3 Yocto Application Development
 - 6.3.1 Configure Qt Creator IDE
 - 6.3.1.1 Qt Creator Hello World
 - 6.3.2 Yocto Hide Cursor
 - 6.4 Yocto Startup Scripts
 - 6.5 Custom Build Yocto
- 7 QNX
 - 7.1 QNX BSP
 - 7.2 QNX Booting
- 8 Android
 - 8.1 Getting Started with Android
 - 8.2 Android Networking
 - 8.3 Android Software Development
 - 8.4 Android Manually Install APK
- 9 Backup / Restore
 - 9.1 MicroSD Card
 - 9.2 eMMC
- 10 Compile the Kernel
 - 10.1 Change Kernel Splash Screen
- 11 Production Mechanism
- 12 Features
 - 12.1 ADC
 - 12.2 Bluetooth
 - 12.3 CAN
 - 12.4 COM Ports
 - 12.5 CPU
 - 12.6 eMMC
 - 12.7 Enclosures
 - 12.8 FPGA
 - 12.8.1 FPGA Crossbar
 - 12.9 GPIO
 - 12.9.1 FPGA GPIO
 - 12.10 Interrupts
 - 12.11 LEDs
 - 12.12 MicroSD Card Interface
 - 12.13 NVRAM
 - 12.14 Onboard SPI Flash
 - 12.15 RTC
 - 12.16 USB
 - 12.16.1 USB OTG
 - 12.16.2 USB Host
 - 12.17 SATA
 - 12.18 Silabs Microcontroller
 - 12.18.1 Silabs Sleep Mode
 - 12.19 SPI

- 12.20 TWI
- 12.21 Watchdog
- 12.22 WIFI
- 13 External Interfaces
 - 13.1 Audio
 - 13.2 COM2 Header
 - 13.3 Ethernet
 - 13.4 HDMI
 - 13.4.1 Rotate the video output
 - 13.5 HD1
 - 13.6 HD2
 - 13.7 HD3
 - 13.8 Mini Card Connector
 - 13.9 Push Button
 - 13.10 RJ45 2W-Modbus
 - 13.11 Terminal Blocks
 - 13.12 USB Device
 - 13.13 USB Hosts
- 14 Specifications
 - 14.1 Power Specifications
 - 14.2 Power Consumption
 - 14.3 Temperature Specifications
 - 14.4 IO Specifications
 - 14.5 Rail Specifications
- 15 Revisions and Changes
 - 15.1 TS-7970 PCB Revisions
 - 15.2 U-Boot Changelog
 - 15.3 FPGA Changelog
 - 15.4 Silabs Changelog
 - 15.5 Software Images
 - 15.5.1 Yocto Changelog
 - 15.5.2 Debian Changelog
 - 15.5.3 Arch Linux Changelog
 - 15.5.4 Ubuntu Linux Changelog
 - 15.5.5 Ubuntu Core Linux Changelog
 - 15.6 TS-7970 Errata
- 16 Product Notes
 - 16.1 FCC Advisory
 - 16.2 Limited Warranty

1 Getting Started

A Linux PC is recommended for development, and will be assumed for this documentation. For users in Windows or OSX we recommend virtualizing a Linux PC. Most of our platforms run Debian and if you have no other distribution preference this is what we recommend.

- Debian.org (<https://www.debian.org/>)

Virtualization

- Virtualbox (Windows or OSX hosts) (<https://www.virtualbox.org/wiki/Downloads>)
- VMware Player (<https://www.vmware.com/products/player>)
- Parallels (OSX) (<http://www.parallels.com/>)

Suggested Linux Distributions

- Debian (<https://www.debian.org/distrib/>)
- Ubuntu (<http://www.ubuntu.com/desktop>)

It may be possible to develop using a Windows or OSX system, but this is not supported. Development will include accessing drives formatted for Linux and often Linux based tools.

1.1 Getting Console and Powering up

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 device.

Get console input by setting the "CON EN" jumper (located near the HDMI connector) and plug a USB type B cable into P2. Connect the host side to a workstation for development. Console can be viewed before or after power is applied. Boot messages will only be printed once the device is powered on.

The cp210x (USB Serial) driver is included in most popular distributions. This will show up as /dev/ttyUSB0. For other operating systems:

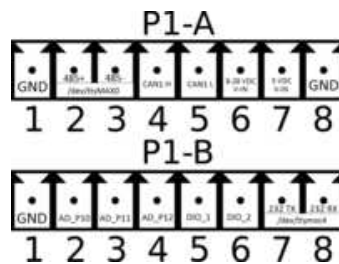
- Silabs USB-to-UART drivers (<http://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx>)

The serial console is provided through this port at 115200 baud, 8n1, with no flow control. Picocom is the recommended linux client to use which can be run with the following command:

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

This will output some serial setting information and then "Terminal ready". Any messages after this point will be from the device via the serial output. The terminal is now ready and power can be applied in order to boot up the device. Power is applied through the removable terminal block. This accepts 5 VDC, or 8-28 VDC input, only a single power input can be connected at any time.

A power supply should be prepared to provide 15 W for most uses. The device's power consumption will average around 3 W on an idle quad core. See the #Specifications section for further details on power requirements.



P1-A, the top row of headers, are used in the following pin designations. For 5 V in, connect pin 7 to a 5 VDC source, and pin 8 to ground. For 8-28 V in, connect pin 6 to the voltage source and pin 8 to ground. See the terminal blocks section for more information on this header.

Once power is applied to either the 5 VDC, or 8-28 VDC the device will output information via the console. The first output is from U-Boot:

```
U-Boot 2015.04-07892-g9a2f707 (Jan 11 2017 - 16:14:53)
CPU: Freescale i.MX6Q rev1.2 at 792 MHz
CPU: Temperature 52 C
Reset cause: WDOG
Board: TS-7970 REV D
I2C: ready
DRAM: 2 GiB
MMC: FSL_SDHC: 0, FSL_SDHC: 1
SF: Detected N25Q64 with page size 256 Bytes, erase size 4 KiB, total 8 MiB
In: serial
Out: serial
Err: serial
FPGA Rev: 7
SilabRev: 2
Net: using phy at 1
FEC [PRIME]
```

Boot will continue immediately unless SW1 is depressed before power is applied and is held down. This will stop boot in U-Boot allowing access to the U-Boot console. This will also cause the U-Boot to check for USB production.

Jumpers on the header near HDMI influence where the system boots. The "SD Boot" jumper, when set, will cause U-Boot to boot to SD, and when unset, U-Boot will boot to eMMC. If SW1 is not depressed, then U-Boot will boot to the selected media immediately.



Note: The "*** Warning - bad CRC, using default environment" can be safely ignored. This indicates that u-boot scripts are not being customized. Typing "env save" will hide these messages, but this is not needed.

1.2 First Linux Boot

U-Boot is always loaded from the onboard SPI flash. U-Boot has the ability to boot Linux, Android, QNX, or other operating systems on the SD or eMMC. The eMMC and SD cards shipped with the unit are pre-programmed with our Debian Jessie image. See other OS sections for information on the various OS options that we provide: Yocto, Ubuntu, Android, QNX.

```
[ OK ] Started Serial Getty on ttymxc0.
[ OK ] Reached target Login Prompts.
[ OK ] Started SLiM Simple Login Manager.
[ OK ] Created slice user-0.slice.
       Starting LSB: RPC portmapper replacement...
       Starting User Manager for UID 0...
[ OK ] Started User Manager for UID 0.
[ OK ] Started LSB: RPC portmapper replacement.
[ OK ] Reached target RPC Port Mapper.
       Starting Authenticate and Authorize Users to Run Privileged Tasks...

Debian GNU/Linux 8 ts-imx6 ttymxc0
ts-imx6 login:
```

By default, the startup output is verbose and includes kernel messages and systemd output. The display, if connected, will boot to a minimalistic XFCE desktop. This is provided as a demo and is not intended for use in development or a shipping application. See the Debian automatic startup section for information on booting to a single application.

Note: During development it is recommended to leave on verbose messages for debugging. The non-error output can be disabled by modifying the kernel cmdline.

Once booted up, the serial port will display a login prompt which asks for a username/password. Under Debian this is "root" with no password which will allow the initial login. From here see the Debian section to continue on with Debian application development.

1.3 Comparison of Distributions

We currently offer Debian, Ubuntu, Yocto, QNX, and Android OSs for the TS-7970. Each of these have advantages and disadvantages, the major points are outlined below. We recommend Debian if the user does not need GPU support. Yocto is recommended for QT Creator, Eclipse support, or significant distribution customization.

Distribution	Advantages	Disadvantages
Debian	<ul style="list-style-type: none"> ▪ Long Life cycles (https://wiki.debian.org/DebianReleases) ▪ Strong focus on reliability ▪ Huge online repository of prebuilt applications available (https://www.debian.org/distrib/packages) ▪ Lots of examples and documentation available 	<ul style="list-style-type: none"> ▪ Cross compilation requires running the same Debian release on a host ▪ Not patched for hardware support <ul style="list-style-type: none"> ▪ No OpenGL or 2D acceleration from GPU. 2D applications through the framebuffer are still supported.
Ubuntu	<ul style="list-style-type: none"> ▪ Long Life cycles (https://wiki.ubuntu.com/Releases) ▪ More focus on up to date packages ▪ Huge online repository of prebuilt applications available (http://packages.ubuntu.com/) ▪ Lots of examples and documentation available 	<ul style="list-style-type: none"> ▪ Cross compilation requires running the same Ubuntu release on a host ▪ Not patched for hardware support <ul style="list-style-type: none"> ▪ No OpenGL or 2D acceleration from GPU. 2D applications through the framebuffer are still supported.
Yocto	<ul style="list-style-type: none"> ▪ Large focus on up to date packages ▪ Allows rebuilding all packages with changes ▪ Supports portable toolchain packages that integrate with QT Creator and Eclipse ▪ Includes all patches needed for graphics support. ▪ Distribution can be rebuilt to include specific needs. 	<ul style="list-style-type: none"> ▪ Short life cycles (https://wiki.yoctoproject.org/wiki/Releases) ▪ Does not support any online repository of prebuilt applications. Adding packages requires rebuilding Yocto or building the required package. ▪ Less examples and documentation available online
Android	<ul style="list-style-type: none"> ▪ Simple well defined API using well documented tools ▪ Allows existing apps to be run without huge customization 	<ul style="list-style-type: none"> ▪ Under Android it is difficult to access hardware that is not found on an Android tablet/phone. This includes interfaces such as UARTs, GPIO, or ADC <ul style="list-style-type: none"> ▪ A common method is to write a C application which communicates over a localhost socket to an Android application in order to interface with hardware ▪ Slow boot time ▪ Poor documentation for OS customization
QNX Neutrino RTOS	<ul style="list-style-type: none"> ▪ Real time OS allowing deterministic response times in an application ▪ Commercial application support available through QNX ▪ Eclipse support 	<ul style="list-style-type: none"> ▪ License fee required through QNX ▪ Not as much driver support as Linux

2 U-Boot

The TS-7970 includes U-Boot as the bootloader to launch the full operating system. When the i.MX6 processor starts it loads U-Boot from the onboard 8MB SPI flash. This allows you to include your boot image on either the SD, eMMC, SATA, NFS, or USB. The U-Boot bootloader is capable of booting Linux, Android, or other operating systems.

On a normal boot you should see output resembling this:

```
U-Boot 2014.10-gee73348 (Oct 07 2015 - 11:12:20)
I2C: ready
DRAM: 1 GiB
MMC: FSL_SDHC: 0, FSL_SDHC: 1
SF: Detected N25Q64 with page size 256 Bytes, erase size 4 KiB, total 8 MiB
In: serial
Out: serial
Err: serial
Net: using phy at 7
FEC [PRIME]
```

By default the device will boot to SD or eMMC depending on the status of "SD Boot" on startup.

To break into the U-Boot console, press and hold the SW1 button while the unit is being powered up. This mode will also check for a USB mass storage device to use for production purposes.

2.1 U-Boot Environment

The eMMC flash contains both the U-Boot executable binary and U-Boot environment. Our default build has 2MiB of environment space which can be used for variables and boot scripts. The following commands are examples of how to manipulate the U-Boot environment:

```
# Print all environment variables
env print -a

# Sets the variable bootdelay to 5 seconds
env set bootdelay 5;

# Variables can also contain commands
env set hellocmd 'led red on; echo Hello world; led green on;'

# Execute commands saved in a variable
env run hellocmd;

# Commit env changes to the spi flash
# Otherwise changes are Lost
env save

# Restore env to default
env default -a

# Remove a variable
env delete emmcboot
```

2.2 U-Boot Commands

```
# The most important command is
help
# This can also be used to see more information on a specific command
help i2c

# This is a command added to u-boot by TS to read the baseboard id
bbdetect
echo ${baseboard} ${baseboardid}
# The echos willreturn:
# TS-8390 2

# Boots into the binary at $loadaddr. This file needs to have
# the uboot header from mkimage. A uImage already contains this.
bootm
# Boots into the binary at $loadaddr, skips the initrd, specifies
# the fdtaddr so Linux knows where to find the board support
bootm ${loadaddr} - ${fdtaddr}

# Get a DHCP address
dhcp
# This sets ${ipaddr}, ${dnsip}, ${gatewayip}, ${netmask}
# and ${ip_dyn} which can be used to check if the dhcp was successful

# These commands are used for scripting:
false # do nothing, unsuccessfully
true # do nothing, successfully

# This command Lets you set fuses in the processor
# Setting fuses can brick your board, will void your warranty,
# and should not be done in most cases
fuse

# GPIO can be manipulated from u-boot. Keep in mind that the IOMUX
# in u-boot is only setup enough to boot the board, so not all pins will
# be set to GPIO mode out of the box. Boot to the full operating system
# for more GPIO support.
# GPIO are specified in bank, IO in the imx6 manual. U-boot uses a flat numberspace,
# so for CN2_83/EIM_OE this is bank 2 dio 25, or (32*2)+25=89
# Set CN1_83 Low
gpio clear 83
# Set CN1_83 high
gpio set 83
# Read CN1_83
gpio input 83

# Control LEDs
led red on
led green on
led all off
led red toggle

# This command is used to copy a file from most devices
# Load kernel from SD
load mmc 0:1 ${loadaddr} /boot/uImage
# Load Kernel from eMMC
load mmc 1:1 ${loadaddr} /boot/uImage
# Load kernel from USB
```

```

usb start
load usb 0:1 ${loadaddr} /boot/uImage
# Load kernel from SATA
sata init
load sata 0:1 ${loadaddr} /boot/uImage

# You can view the fdt from u-boot with fdt
load mmc 0:1 ${fdtaddr} /boot/imx6q-ts4900.dtb
fdt addr ${fdtaddr}
fdt print

# You can blindly jump into any memory
# This is similar to bootm, but it does not use the
# u-boot header
load mmc 0:1 ${loadaddr} /boot/custombinary
go ${loadaddr}

# Browse fat,ext2,ext3,or ext4 filesystems:
ls mmc 0:1 /

# Access memory like devmem in Linux, you can read/write arbitrary memory
# using mw and md
# write
mw 0x10000000 0xc0ffee00 1
# read
md 0x10000000 1

# Test memory.
mtest

# Check for new SD card
mmc rescan
# Read SD card size
mmc dev 0
mmcinfo
# Read eMMC Size
mmc dev 1
mmcinfo

# The NFS command is Like 'Load', but used over the network
dhcp
env set serverip 192.168.0.11
nfs ${loadaddr} 192.168.0.11:/path/to/somefile

# Test ICMP
dhcp
ping 192.168.0.11

# Reboot
reset

# SPI access is through the SF command
# Be careful with sf commands since
# this is where u-boot and the FPGA bitstream exist
# Improper use can render the board unbootable
sf probe

# Delay in seconds
sleep 10

# You can Load HUSH scripts that have been created with mkimage
load mmc 0:1 ${loadaddr} /boot/ubootscript
source ${loadaddr}

# Most commands have return values that can be used to test
# success, and HUSH scripting supports comparisons like
# test in Bash, but much more minimal
if load mmc 1:1 ${fdtaddr} /boot/uImage;
then echo Loaded Kernel
else
echo Could not find kernel
fi

# Commands can be timed with "time"
time sf probe

# Print U-boot version/build information
version

```

2.3 Modify Linux Kernel cmdline

The Linux kernel cmdline can be customized by modifying the `cmdline_append` variable. If new arguments are added, the existing value should also be included with the new arguments.

```

env set cmdline_append console=ttymxc0,115200 init=/sbin/init quiet
env save

```

The kernel command line can also be modified from from the onboard Linux. From the linux shell prompt run the following commands to install the necessary tools and create the script:

```

apt-get update && apt-get install u-boot-tools -y
echo "env set cmdline_append console=ttymxc0,115200 init=/sbin/init quiet" > /boot/boot.scr
mkimage -A arm -T script -C none -n 'tsimx6 boot script' -d /boot/boot.scr /boot/boot.ub

```


The boot.scr includes the plain text commands to be run in U-Boot on startup. The mkimage tool adds a checksum and header to this file which can be loaded by U-Boot. The .ub file should not be edited directly.

2.4 U-boot Recovery

U-Boot itself handles CPU and RAM setup/configuration that needs to be run every boot. Therefore separate binaries are maintained for each CPU grade, RAM part, and RAM size; and the proper binary must be used for the device configuration. On a functional unit, run 'env print imx_type' in U-Boot and it will return the correct device variant.

On startup, the TS-7970 checks the SPI flash for a valid boot header in SPI flash. If it is unable to locate a valid boot header, the CPU falls back to the "serial downloader" which allows the CPU to execute code sent via USB. If the board has a valid boot header, but a damaged or invalid U-Boot binary located in SPI; an RMA return or a TS-9468 development board will be required in order to properly recover the unit. Please contact us (<https://www.embeddedarm.com/support/contact-us.php>) for assistance with this.

- 1) Download u-boot for the correct imx_type variant from the list here: <ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/>. See the U-Boot Changelog for information on the changes between released versions.
- 2) Download and build/install the "imx_usb" loader: <https://community.freescale.com/docs/DOC-94117>
- 3) Disconnect power from the device.
- 4) Remove the "CON EN" jumper.
- 5) Apply power to the device.
- 6) Plug a USB type B cable into the P2 connector on the device and connect it to a host PC.
- 7) Check 'dmesg' or 'lsusb' on the host PC for a new USB connection. This should show a HID device listing NXP or Freescale as the manufacturer. For example:

```
hid-generic 0003:15A2:0054.0006: hiddev0,hidraw3: USB HID v1.10 Device [Freescale Semiconductor Inc SE Blank ARIK] on usb-0000:00:14.0-6.4.2/input
```

If it does not show the above output, an RMA return or TS-9468 development board will be required in order to properly recover the unit. Please contact us (<https://www.embeddedarm.com/support/contact-us.php>) for assistance with this.

- 8) Hold down SW1.
- 9) Run 'imx_usb path/to/u-boot.imx' on the host PC while still holding down SW1. Continue holding SW1 for a few seconds after the command is run.
- 10) Disconnect the USB cable on P2.
- 11) Set the "CON EN" jumper.
- 12) Re-insert the USB cable into P2.

At this point, the USB serial device should show up on the host, opening it will reveal that the unit is stopped at the U-Boot prompt. Follow the steps in Update U-Boot to reinstall U-Boot on the SPI flash.

2.5 Linux NFS Boot

U-Boot's NFS support can be used to load a kernel, device tree binary, and root filesystem over the network. The default scripts include an example NFS boot script.

```
# Set this to your NFS root path. The server root should be accessible at this path.
env set nfsroot 192.168.0.36:/mnt/storage/imx6/
env save
```

To boot to an NFS root:

```
# Boot to NFS once
run nfsboot;

# To make the NFS boot the persistent default
env set bootcmd run nfsboot;
env save
```

2.6 Update U-Boot

WARNING: Installing a custom U-Boot is not recommended and may cause the device to fail to boot.

U-Boot requires a different build for Quad/Dual and Solo/Duallite. When booted to the U-Boot shell, run 'env print imx_type' and it will return the correct U-Boot build that should be used. Copy the built u-boot.imx file or the pre-built binary from our FTP site (<ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/>) to the SD card as "/u-boot.imx", and run the following U-Boot commands:

```
mmc dev 0
load mmc 0:1 ${loadaddr} /u-boot.imx
sf probe
sf erase 0 0x80000
sf write ${loadaddr} 0x400 $filesize
```

2.7 U-boot Development

While we do provide our u-boot sources, we typically do not recommend rebuilding a custom uboot if it can be avoided. This CPU has a long lifetime which will outlast most RAM chips. If we have to update the RAM timing later in the boards life due to an EOL, die change, or any other change that may require new RAM configuration/timing changes, we will update this in our shipping u-boot. If this happens in the future our shipping u-boot will include these fixes and not require user intervention. If you are using a custom u-boot you may need to pull down our new changes and rebuild to get the updated configuration.

Our u-boot includes a variable "imx_type". With a custom u-boot, make sure to check the value of this before writing. If we are forced to update the RAM configuration we will change this variable. We will also send out a product change to anyone who is subscribed to our PCS system (<http://pcs.embeddedarm.com/>).

If you still want to proceed with building a custom u-boot, use the imx_v2015.04_3.14.52_1.1.0_ga branch from the github here: <https://github.com/embeddedarm/u-boot-imx>

Boot up a TS-7970 into u-boot and run "echo \${imx_type}". This will show you the u-boot config to use for the correct RAM timing. We use the same GCC 6.2 used from Yocto Morty to compile the u-boot binary.

```
export ARCH=arm
export CROSS_COMPILE=/opt/poky/2.2.1/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-
git clone https://github.com/embeddedarm/u-boot-imx.git -b imx_v2015.04_3.14.52_1.1.0_ga
cd u-boot-imx

# For example, one of the quad core variants. Replace this with your imx_type
make ts7970-s-1g-800mhz-i_defconfig
make -j4
```

This will output a u-boot.imx that can be written to the board using the steps in #Update u-boot.

2.8 Access U-boot Environment from Linux

U-Boot includes a utility fw_printenv which set/read environment variables from Linux. This must be built and provided with a config file before it will work.

On the board first boot to u-boot by holding SW1 pressed on startup. At the prompt run:

```
U-Boot > env print imx_type
imx_type=ts7970-s-1g-800mhz-i
```

Save this output then boot to Linux to build the fw_printenv tool.

```
cd /usr/src/
git clone --depth 1 https://github.com/embeddedarm/u-boot-imx.git -b imx_v2015.04_3.14.52_1.1.0_ga
cd u-boot-imx
```

Since u-boot gave us "imx_type=ts7970-s-1g-800mhz-i", the example defconfig is ts4900-s-1g-800mhz-i_defconfig. Your board's defconfig may be different and this build should match.

```
make ts7970-s-1g-800mhz-i_defconfig
make -j4 env
```

```
cp tools/env/fw_printenv /usr/bin/
# The same utility sets environment variables when
# called as fw_setenv
ln -s /usr/bin/fw_printenv /usr/bin/fw_setenv
```

The board will also need a config file to know where to load the environment. Create a file /etc/fw_env.config

```
# SPI flash on the TS-7970/TS-7990
# MTD device name      Device offset  Env. size      Flash sector size  Number of sectors
/dev/mtdblock0         0x100000      0x2000         0x1000             2
/dev/mtdblock0         0x180000      0x2000         0x1000             2
```

From here you can run "fw_printenv" and read the environment variables. If first line of output is:

```
Warning: Bad CRC, using default environment
```

Then the environment is blank, and u-boot is loading the environment compiled into the u-boot binary. This is normal and is how boards are shipped.

You can modify variables with this command as well:

```
# Set cmdline_append to include "quiet"
fw_setenv cmdline_append console=ttymxc0,115200 ro init=/sbin/init quiet
```

3 Debian

Debian is a community run Linux distribution. Debian provides tens of thousands of precompiled applications and services. This distribution is known for stability and large community providing support and documentation. This distribution does not include the same support as Yocto for the GPU. OpenGL ES, Gstreamer, or OpenCL are not supported in Debian. The framebuffer is supported so 2D applications will still run well. Debian is also very well suited for headless applications.

3.1 Getting Started with Debian

Once installed the default user is "root" with no password.

- debian-armhf-jessie-latest.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-latest.tar.bz2) (md5 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-latest.tar.bz2.md5))

Note: This is a shared image that supports the TS-4900, TS-7970, and TS-TPC-7990.

To prepare an SD card, use partitioning tools such as 'fdisk' 'cfdisk' or 'gparted' in linux to create a single linux partition on the SD card. See the guide here for more information. Once it is formatted, extract the above tarball with:

```
# Assuming your SD card is /dev/sdc with one partition
mkfs.ext3 /dev/sdc1
mkdir /mnt/sd/
sudo mount /dev/sdc1 /mnt/sd/
sudo tar --numeric-owner -xjf debian-armhf-jessie-latest.tar.bz2 -C /mnt/sd
sudo umount /mnt/sd
sync
```

Note: The ext4 filesystem can be used instead of ext3, but it may require additional options. U-Boot does not support the 64bit addressing added as the default behavior in recent revisions of mkfs.ext4. If using e2fsprogs 1.43 or newer, the options "-O ^64bit,^metadata_csum" must be used with ext4 for proper compatibility. Older versions of e2fsprogs do not need these options passed nor are they needed for ext3.

To rewrite the eMMC the unit must be booted to SD or any other media that is not eMMC. Once booted, run the following commands.:

```
mkfs.ext3 /dev/mmcblk2p1
mkdir /mnt/emmc
mount /dev/mmcblk2p1 /mnt/emmc
wget -qO- ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-latest.tar.bz2 | tar xj -C /mnt/emmc
umount /mnt/emmc
sync
```

The same commands can be used to write a SATA drive by substituting /dev/mmcblk2p1 with /dev/sda1.

3.2 Debian Networking

From almost any Linux system you can use 'ip' command or the 'ifconfig' and 'route' commands to initially set up the network.

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

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

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

# Set a specific subnet
ifconfig eth0 192.168.0.50 netmask 255.255.0.0

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

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

Most networks will offer a DHCP server, an IP address can be obtained from a server with a single command in linux:

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
```

Systemd provides a networking configuration option to allow for automatic configuration on startup. Systemd-networkd has a number of different configuration files, some of the default examples and setup steps are outlined below.

/etc/systemd/network/eth.network

```
[Match]
Name=eth*

[Network]
DHCP=yes
```

To use DHCP to configure DNS via systemd, start and enable the network name resolver service, systemd-resolved:

```
systemctl start systemd-resolved.service
systemctl enable systemd-resolved.service
ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

For a static config create a network configuration for that specific interface.

/etc/systemd/network/eth0.network

```
[Match]
Name=eth0

[Network]
Address=192.168.0.50/24
Gateway=192.168.0.1
DNS=192.168.0.1
```

For more information on networking, see Debian and systemd's documentation:

- Systemd Networking Documentation (<http://www.freedesktop.org/software/systemd/man/systemd.network.html>)
- Debian Documentation (<http://wiki.debian.org/Network>)

3.2.1 Debian WIFI Client

If connecting to a WPA/WPA2 network, a wpa_supplicant config file must first be created:

```
wpa_passphrase yournetwork yournetworkpassphrase > /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

Create the file /lib/systemd/system/wpa_supplicant@.service with these contents

```
[Unit]
Description=WPA supplicant daemon (interface-specific version)
Requires=sys-subsystem-net-devices-%i.device
After=sys-subsystem-net-devices-%i.device

[Service]
Type=simple
ExecStart=/sbin/wpa_supplicant -c/etc/wpa_supplicant/wpa_supplicant-%I.conf -i%i

[Install]
Alias=multi-user.target.wants/wpa_supplicant@%i.service
```

Create the file /etc/systemd/network/wlan0.network with:

```
[Match]
Name=wlan0

[Network]
DHCP=yes
```

See the systemctl-networkd example for setting a static IP for a network interface. The wlan0.network can be configured the same way as an eth.network.

To enable all of the changes that have been made, run the following commands:

```
systemctl enable wpa_supplicant@wlan0
systemctl start wpa_supplicant@wlan0
systemctl restart systemd-networkd
```

3.2.2 Debian WIFI Access Point

First, hostapd needs to be installed in order to manage the access point on the device:

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

Note: The install process will start an unconfigured hostapd process. This process must be killed and restarted before a new hostapd.conf will take effect.

Edit /etc/hostapd/hostapd.conf to include the following lines:

```
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 start up an access point that can be detected by WIFI clients. A DHCP server will likely be desired to assign IP addresses. Refer to Debian's documentation for more details on DHCP configuration (https://wiki.debian.org/DHCP_Server).

3.3 Debian Application Development

3.3.1 Debian Jessie Cross Compiling

Debian Jessie provides cross compilers from its distribution. An install on a workstation can build for the same release on other architectures. A PC, virtual machine, or chroot will need to be used for this. Install Debian Jessie for your workstation here (<https://www.debian.org/releases/jessie/>).

From a Debian workstation (not the target), run these commands to set up the cross compiler:

```
# Run "lsb_release -a" and verify Debian 8.X is returned. These instructions are not
# expected to work on any other version or distribution.
apt-get install curl build-essential
su root
echo "deb http://emdebian.org/tools/debian jessie main" > /etc/apt/sources.list.d/emdebian.list
curl http://emdebian.org/tools/debian/emdebian-toolchain-archive.key | apt-key add -
# Note that while Ubuntu uses apt as well, Ubuntu host setup is slightly different, instead of the above commands use the following:
# echo "deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports trusty main restricted universe multiverse" >> /etc/apt/sources.list
# echo "deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports trusty-updates main restricted universe multiverse" >> /etc/apt/sources.list
# echo "deb [arch=armhf] http://ports.ubuntu.com/ubuntu-ports trusty-security main restricted universe multiverse" >> /etc/apt/sources.list
dpkg --add-architecture armhf
apt-get update
apt-get install crossbuild-essential-armhf
```

This will install a toolchain that can be used with the prefix "arm-linux-gnueabihf-". The standard GCC tools will start with that name, eg "arm-linux-gnueabihf-gcc".

The toolchain can now compile a simple hello world application. Create hello-world.c on the Debian workstation:

```
#include <stdio.h>
int main(){
    printf("Hello World\n");
}
```

To compile this:

```
arm-linux-gnueabihf-gcc hello-world.c -o hello-world
file hello-world
```

This will return that the binary created is for ARM. Copy this to the target platform to run it there.

Debian Jessie supports multiarch which can install packages designed for other architectures. On workstations this is how 32-bit and 64-bit support is provided. This can also be used to install armhf packages on an x86 based workstation.

This cross compile environment can link to a shared library from the Debian root. The package would be installed in Debian on the workstation to provide headers and .so. This is included in most "-dev" packages. When run on the arm target it will also need a copy of the library installed, but it does not need the -dev package.

```
apt-get install libcurl4-openssl-dev:armhf
# Download the simple.c example from curl:
wget https://raw.githubusercontent.com/bagder/curl/master/docs/examples/simple.c
# After installing the supporting library, curl will link as compiling on the unit.
arm-linux-gnueabihf-gcc simple.c -o simple -lcurl
```

Copy the binary to the target platform and run on the target. This can be accomplished with network protocols like NFS, SCP, FTP, etc.

If any created binaries do not rely on hardware support like GPIO or CAN, they can be run using qemu.

```
# using the hello world example from before:
./hello-world
# Returns Exec format error
apt-get install qemu-user-static
./hello-world
```

3.4 Debian Installing New Software

Debian provides the apt-get system which allows management of pre-built applications. First apt will need a network connection to the internet. The update command will download a list of prebuilt packages and the current version.

Debian provides the apt-get system which lets you manage pre-built 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
```

A common example is installing Java runtime support for a system. Find the package name first with search, and then install it.

```
root@ts:~# apt-cache search openjdk
jvm-7-avian-jre - lightweight virtual machine using the OpenJDK class library
freemind - Java Program for creating and viewing Mindmaps
icedtea-7-plugin - web browser plugin based on OpenJDK and IcedTea to execute Java applets
default-jdk - Standard Java or Java compatible Development Kit
default-jdk-doc - Standard Java or Java compatible Development Kit (documentation)
default-jre - Standard Java or Java compatible Runtime
default-jre-headless - Standard Java or Java compatible Runtime (headless)
jtest - Regression Test Harness for the OpenJDK platform
libreoffice - office productivity suite (metapackage)
icedtea-7-jre-jamvm - Alternative JVM for OpenJDK, using JamVM
openjdk-7-dbg - Java runtime based on OpenJDK (debugging symbols)
openjdk-7-demo - Java runtime based on OpenJDK (demos and examples)
openjdk-7-doc - OpenJDK Development Kit (JDK) documentation
openjdk-7-jdk - OpenJDK Development Kit (JDK)
openjdk-7-jre - OpenJDK Java runtime, using Hotspot Zero
openjdk-7-jre-headless - OpenJDK Java runtime, using Hotspot Zero (headless)
openjdk-7-jre-lib - OpenJDK Java runtime (architecture independent libraries)
openjdk-7-source - OpenJDK Development Kit (JDK) source files
uwsgi-app-integration-plugins - plugins for integration of uWSGI and application
uwsgi-plugin-jvm-openjdk-7 - Java plugin for uWSGI (OpenJDK 7)
uwsgi-plugin-jwsgi-openjdk-7 - JWSGI plugin for uWSGI (OpenJDK 7)
```

In this case you will want the openjdk-7-jre package. Names of packages are on Debian's wiki (<http://wiki.debian.org/>) or the packages site (<https://packages.debian.org/jessie/>).

With the package name apt-get install can be used to install the prebuilt packages.

```
apt-get install openjdk-7-jre
# More than one package can be installed at a time.
apt-get install openjdk-7-jre nano vim mplayer
```

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

3.5 Debian Setting up SSH

To install ssh, install the package as normal with apt-get:

```
apt-get install openssh-server
```

Make sure the device is configured on the network and set a password for the remote user. SSH will not allow remote connections without a password or a valid SSH key pair.

```
passwd root
```

After this setup it is now possible to connect from a remote PC supporting SSH. On Linux/OS X this is the "ssh" command, or from Windows using a client such as putty (<http://www.chiark.greenend.org.uk/~sgtatham/putty/>).

Note: If a DNS server is not present on the target network, it is possible to save time at login by adding "UseDNS no" in `/etc/ssh/sshd_config`.

3.6 Debian Starting Automatically

A systemd service can be created to start up headless applications. Create a file in `/etc/systemd/system/yourapp.service`

```
[Unit]
Description=Run an application on startup

[Service]
Type=simple
ExecStart=/usr/local/bin/your_app_or_script
```

```
[Install]
WantedBy=multi-user.target
```

If networking is a dependency add "After=network.target" in the Unit section. Once you have this file in place add it to startup with:

```
# Start the app on startup, but will not start it now
systemctl enable yourapp.service

# Start the app now, but doesn't change auto startup
systemctl start yourapp.service
```

Note: See the systemd documentation (<http://www.freedesktop.org/software/systemd/man/systemd.service.html>) for in depth documentation on services.

To start an application on bootup with X11 instead change the x-session-manager. By default the system starts xfce:

```
root@ts:~# ls -lah /usr/bin/x-session-manager
lrwxrwxrwx 1 root root 35 May 26 2015 /usr/bin/x-session-manager -> /etc/alternatives/x-session-manager
root@ts:~# ls -lah /etc/alternatives/x-session-manager
lrwxrwxrwx 1 root root 19 May 26 2015 /etc/alternatives/x-session-manager -> /usr/bin/startxfce4
```

The x-session can be modified to only start specified processes. Create the file /usr/bin/mini-x-session with these contents:

```
#!/bin/bash
matchbox-window-manager -use_titlebar no &

exec xfce4-terminal
```

You may need to "apt-get install matchbox-window-manager." first. This is a tiny window manager which also has a few flags that simplify embedded use. Now enable this session manager and restart slim to restart x11 and show it now.

```
chmod a+x /usr/bin/mini-x-session
rm /etc/alternatives/x-session-manager
ln -s /usr/bin/mini-x-session /etc/alternatives/x-session-manager
service slim restart
```

If the x-session-manager process ever closes x11 will restart. The exec command allows a new process to take over the existing PID. In the above example xfce4-terminal takes over the PID of x-session-manager. If the terminal is closed with commands like exit the slim/x11 processes will restart.

4 Ubuntu

Ubuntu is a distribution provided by Canonical which is based on Debian. Ubuntu often has more recent packages but follows a shorter release cycle. The image we provide is based on Ubuntu. We use the root filesystem, but the kernel is not provided by Ubuntu or in any way associated with Canonical. Our Kernel is based on the imx_4.1.15_1.0.0_ga release from git.freescale.com. This is patched to provide support for our boards and peripherals.

- Ubuntu Releases (<https://wiki.ubuntu.com/Releases>)
- Freescale git (<http://git.freescale.com/git/cgi.cgi/imx/linux-2.6-imx.git/>)
- Technologic 4.1 imx git (https://github.com/embeddedarm/linux-3.10.17-imx6/tree/imx_4.1.15_1.0.0_ga)

This image includes support for the TS-4900, TS-7970, and TS-TPC-7990.

4.1 Getting Started with Ubuntu

The latest release is available here:

- ubuntu-armhf-16.04-latest.tar.bz2 (<ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu/ubuntu-armhf-16.04-latest.tar.bz2>)

The login is either "root" with no password, or username "ubuntu" with the password "ubuntu". The ubuntu user is allowed to run sudo.

To write this to an SD card, first partition the SD card to have one large ext3, or ext4 partition. See the guide here for more information. Once it is formatted, extract this tar with:


```
# Assuming your SD card is /dev/sdc with one partition
mkfs.ext3 /dev/sdc1
mkdir /mnt/sd/
sudo mount /dev/sdc1 /mnt/sd/
sudo tar --numeric-owner -xjf ubuntu-armhf-16.04-latest.tar.bz2 -C /mnt/sd
sudo umount /mnt/sd
sync
```

To rewrite the eMMC, boot to the SD card. You cannot rewrite the emmc while it is mounted elsewhere, or used to currently boot the system. Once booted to the SD, run:

```
mkfs.ext3 /dev/mmcblk2p1
mkdir /mnt/emmc
mount /dev/mmcblk2p1 /mnt/emmc
wget -q0- ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu/ubuntu-armhf-16.04-latest.tar.bz2 | tar --numeric-owner -xjf - -C /mnt/emmc
umount /mnt/emmc
sync
```

The ext4 filesystem can be used instead of ext3, but it may require additional options. U-Boot does not support the 64bit addressing added as the default behavior in recent revisions of mkfs.ext4. If using e2fsprogs 1.43 or newer, the options "-O ^64bit,^metadata_csum" must be used with ext4 for proper compatibility. Older versions of e2fsprogs do not need these options passed nor are they needed for ext3.

Note:

4.2 Ubuntu Networking

From almost any Linux system you can use "ip" or the ifconfig/route commands to set up the network.

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

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

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

# Set a specific subnet
ifconfig eth0 192.168.0.50 netmask 255.255.0.0

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

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

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

```
# 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
```

To make DHCP run on startup systemd's networking will need to be configured.

In /etc/systemd/network/eth.network

```
[Match]
Name=eth*

[Network]
DHCP=yes
```

Then, if you intend to use DHCP to configure your DNS, start and enable the network name resolver service:

```
systemctl start systemd-resolved.service
systemctl enable systemd-resolved.service
ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

For a static configuration create a config file for that specific interface. /etc/systemd/network/eth0.network

```
[Match]
Name=eth0

[Network]
Address=192.168.0.50/24
Gateway=192.168.0.1
DNS=192.168.0.1
```

For more information on networking, see Ubuntu and systemd's documentation:

- Systemd Networking Documentation (<http://www.freedesktop.org/software/systemd/man/systemd.network.html>)
- Ubuntu Documentation (<https://help.ubuntu.com/lts/serverguide/network-configuration.html>)

4.2.1 Ubuntu WIFI Client

If connecting to a WPA/WPA2 network, a wpa_supplicant config file must first be created:

```
wpa_passphrase yournetwork yournetworkpassphrase > /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

Create the file `/lib/systemd/system/wpa_supplicant@.service` with these contents

```
[Unit]
Description=WPA supplicant daemon (interface-specific version)
Requires=sys-subsystem-net-devices-%i.device
After=sys-subsystem-net-devices-%i.device

[Service]
Type=simple
ExecStart=/sbin/wpa_supplicant -c/etc/wpa_supplicant/wpa_supplicant-%I.conf -i$I

[Install]
Alias=multi-user.target.wants/wpa_supplicant@%i.service
```

Next, enable the service to start up on boot:

```
systemctl enable wpa_supplicant@wlan0
```

Create the file `/etc/systemd/network/wlan0.network` with:

```
[Match]
Name=wlan0

[Network]
DHCP=yes
```

Enable networkd to run dhcp on startup:

```
systemctl enable systemd-networkd
```

See the `systemctl-networkd` example for setting a static IP for a network interface. The `wlan0.network` can be configured the same way as an `eth.network`. To enable all of the changes that have been made, run the following commands:

```
systemctl enable wpa_supplicant@wlan0
systemctl start wpa_supplicant@wlan0
systemctl restart systemd-networkd
```

4.2.2 Ubuntu WIFI Access Point

First, `hostapd` needs to be installed in order to manage the access point on the device:

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

Note: The install process will start an unconfigured `hostapd` process. This process must be killed and restarted before a new `hostapd.conf` will take effect.

Edit /etc/hostapd/hostapd.conf to include the following lines:

```
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 start up an access point that can be detected by WIFI clients. A DHCP server will likely be desired to assign IP addresses. Refer to Debian's documentation for more details on DHCP configuration (https://wiki.debian.org/DHCP_Server).

4.3 Ubuntu Installing New Software

Ubuntu provides the apt-get system which lets you manage pre-built applications. Before you do this you need to update Ubuntu'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.

```
root@ts-imx6:~# apt-cache search openjdk
jvm-7-avian-jre - lightweight virtual machine using the OpenJDK class library
freemind - Java Program for creating and viewing Mindmaps
icedtea-7-plugin - web browser plugin based on OpenJDK and IcedTea to execute Java applets
default-jdk - Standard Java or Java compatible Development Kit
default-jdk-doc - Standard Java or Java compatible Development Kit (documentation)
default-jre - Standard Java or Java compatible Runtime
default-jre-headless - Standard Java or Java compatible Runtime (headless)
jtest - Regression Test Harness for the OpenJDK platform
libreoffice - office productivity suite (metapackage)
icedtea-7-jre-jamvm - Alternative JVM for OpenJDK, using JamVM
openjdk-7-dbg - Java runtime based on OpenJDK (debugging symbols)
openjdk-7-demo - Java runtime based on OpenJDK (demos and examples)
openjdk-7-doc - OpenJDK Development Kit (JDK) documentation
openjdk-7-jdk - OpenJDK Development Kit (JDK)
openjdk-7-jre - OpenJDK Java runtime, using Hotspot Zero
openjdk-7-jre-headless - OpenJDK Java runtime, using Hotspot Zero (headless)
openjdk-7-jre-lib - OpenJDK Java runtime (architecture independent libraries)
openjdk-7-source - OpenJDK Development Kit (JDK) source files
uwsgi-app-integration-plugins - plugins for integration of uWSGI and application
uwsgi-plugin-jvm-openjdk-7 - Java plugin for uWSGI (OpenJDK 7)
uwsgi-plugin-jwsgi-openjdk-7 - JWSGI plugin for uWSGI (OpenJDK 7)
```

In this case you will likely want openjdk-7-jre to provide a runtime environment, and possibly openjdk-7-jdk to provide a development environment.

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-7-jre
# You can also chain packages to be installed
apt-get install openjdk-7-jre nano vim mplayer
```

For more information on using apt-get refer to Ubuntu's documentation here (<https://help.ubuntu.com/community/AptGet/Howto>).

4.4 Ubuntu Setting up SSH

To install ssh, install the package as normal with apt-get:

```
apt-get install 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/>) .

4.5 Ubuntu Starting Automatically

A systemd service can be created to start up headless applications. Create a file in /etc/systemd/system/yourapp.service

```
[Unit]
Description=Run an application on startup

[Service]
Type=simple
ExecStart=/usr/local/bin/your_app_or_script

[Install]
WantedBy=multi-user.target
```

If networking is a dependency add "After=network.target" in the Unit section. Once you have this file in place add it to startup with:

```
# Start the app on startup, but will not start it now
systemctl enable yourapp.service

# Start the app now, but doesn't change auto startup
systemctl start yourapp.service
```

Note: See the systemd documentation (<http://www.freedesktop.org/software/systemd/man/systemd.service.html>) for in depth documentation on services.

To start an application on bootup with X11 instead change the x-session-manager. By default the system starts xfce:

```
root@ts:~# ls -lah /usr/bin/x-session-manager
lrwxrwxrwx 1 root root 35 May 26 2015 /usr/bin/x-session-manager -> /etc/alternatives/x-session-manager
root@ts:~# ls -lah /etc/alternatives/x-session-manager
lrwxrwxrwx 1 root root 19 May 26 2015 /etc/alternatives/x-session-manager -> /usr/bin/startxfce4
```

The x-session can be modified to only start specified processes. Create the file /usr/bin/mini-x-session with these contents:

```
#!/bin/bash
matchbox-window-manager -use_titlebar no &

exec xfce4-terminal
```

You may need to "apt-get install matchbox-window-manager." first. This is a tiny window manager which also has a few flags that simplify embedded use. Now enable this session manager and restart slim to restart x11 and show it now.

```
chmod a+x /usr/bin/mini-x-session
rm /etc/alternatives/x-session-manager
ln -s /usr/bin/mini-x-session /etc/alternatives/x-session-manager
service slim restart
```

If the x-session-manager process ever closes x11 will restart. The exec command allows a new process to take over the existing PID. In the above example xfce4-terminal takes over the PID of x-session-manager. If the terminal is closed with commands like exit the slim/x11 processes will restart.

5 Ubuntu Core

Ubuntu Core is a new distribution provided by Canonical targetted towards embedded/IoT projects. This requires users to generate "snap" packages for their application, but provides a mechanism for save remote updates to the OS and packages. Our kernel is based on Ubuntu 16's 4.4 based kernel to provide the best compatibility and support. Bug fixes to units using our kernel snap are provided through the ubuntu core app store.

Read more about Ubuntu Core here (<https://www.ubuntu.com/core>) .

5.1 Getting Started with Ubuntu Core

Download our latest image here (<ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu-core/ubuntu-core-16-latest.img.bz2>) .

Write this to an SD card with:

```
wget ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions\
/ubuntu-core/ubuntu-core-16-latest.img.bz2
bzip2 -d ubuntu-core-16-latest.img.bz2
# Assuming /dev/sdd is your SD card. Check dmesg after inserting for your device.
# Make sure this is the block device (/dev/sdd) and not a partition (/dev/sdd1).
dd if=/path/to/ubuntu-core-16-latest.img bs=4M of=/dev/sdd conv=fsync && sync
```

This can be written to emmc using the USB production mechanism.

Next make an Ubuntu SSO account (<https://login.ubuntu.com/>) . Generate SSH keys (<https://help.ubuntu.com/community/SSH/OpenSSH/Keys>) and upload your SSH keys (<https://login.ubuntu.com/ssh-keys>) to your account.

Once written to either boot media start up the board. After boot has completed it will leave you at a screen:

```
Press Enter to Configure
```

Ubuntu core has no default username/password and must be configured with their single sign on service to fetch your SSH keys.

Press enter and it will have you confirm DHCP, or use a static network configuration. Once configured it will ask for your Ubuntu SSO username. This will create an account on the Ubuntu Core image and allow access only with your SSH keys present on the store. After it has fetched your keys it will print out the ssh commands to connect to your unit which will now allow access just with your SSH keys.

Connect to the board with:

```
ssh <ubuntu SSO username>@<IP of board>
```

Note: This must be run on a system that has your SSH keys installed.

Once connected you have access to a shell prompt and can install any needed snaps. See <http://snapcraft.io/> for more information on developing your own snaps for your application which can be uploaded to the store.

5.2 Ubuntu Core Reference Links

- <https://myapps.developer.ubuntu.com/dev/click-apps/>
- <https://github.com/embeddedarm/ubuntu-core>
- <https://github.com/embeddedarm/ubuntu-kernel>
- <http://snapcraft.io/>
- <http://snapcraft.io/docs/>
- <https://www.ubuntu.com/core>

6 Yocto

Yocto is our recommended distribution for graphics packages as the software includes patches to support the GPU. X11 in Yocto includes drivers for providing 2D support as well. Support is also provided for OpenGL ES 1&2, as well as GStreamer acceleration, included standalone or with Qt. Yocto also provides cross toolchains that include the rootfs. This toolchain allows integration with the Qt Creator IDE and Eclipse.

Yocto does not provide binary security updates. This distribution also does not have any remote repository of pre-built applications. For either of these we features we recommend using Debian.

Our current Yocto support is based off of Yocto 2.2 "Morty".

6.1 Getting Started with Yocto

Yocto itself is a set of scripts and tools used to build a custom Distribution. In our default images we try to include all the common utilities requested by users. Rebuilding Yocto should not be necessary for many users, but is possible if needed. Once installed the default user is "root" with no password.

Our Yocto rootfs is available here:

Yocto Download Links

Yocto Image	Download Link
ts-x11-image	Download (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/ts-x11-image-tsimx6-latest.rootfs.tar.bz2)

To write this to an SD card, first partition the SD card to have one large ext3 partition. See the guide here for more information. Once it is formatted, extract this tar with:

```
# Assuming your SD card is /dev/sdc with one partition
mkfs.ext3 /dev/sdc1
mkdir /mnt/sd/
sudo mount /dev/sdc1 /mnt/sd/
sudo tar --numeric-owner -jxf ts-x11-image-tsimx6-latest.rootfs.tar.bz2 -C /mnt/sd
sudo umount /mnt/sd
sync
```

Note:

The ext4 filesystem can be used instead of ext3, but it may require additional options. U-Boot does not support the 64bit addressing added as the default behavior in recent revisions of mkfs.ext4. If using e2fsprogs 1.43 or newer, the options "-O ^64bit,^metadata_csum" must be used with ext4 for proper compatibility. Older versions of e2fsprogs do not need these options passed nor are they needed for ext3.

To rewrite the eMMC, boot to the SD card. You cannot rewrite the eMMC while it is mounted elsewhere, or used to currently boot the system. Once booted to the SD, run:

```
mkfs.ext3 /dev/mmcblk2p1
mkdir /mnt/emmc
mount /dev/mmcblk2p1 /mnt/emmc
wget -qO- ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/\
ts-4900-linux/distributions/yocto/morty/ts-x11-image-tsimx6-\
latest.rootfs.tar.bz2 | tar --numeric-owner xj -C /mnt/emmc/
umount /mnt/emmc
sync
```

The same commands can be used to write SATA by substituting /dev/mmcblk2p1 with /dev/sda1.

6.2 Yocto Networking

Our Yocto image uses systemd which stores its network files in "/etc/systemd/network/". The simplest network config for a DHCP configuration would look like this:

In /etc/systemd/network/eth.network

```
[Match]
Name=eth*

[Network]
DHCP=yes
```

For a static configuration create a config file for that specific interface. /etc/systemd/network/eth0.network

```
[Match]
Name=eth0

[Network]
Address=192.168.0.50/24
Gateway=192.168.0.1
DNS=192.168.0.1
```

DNS will be loaded from /etc/resolv.conf. To make this use a static DNS:

```
rm /etc/resolv.conf
echo "nameserver 8.8.8.8" > /etc/resolv.conf
echo "nameserver 8.8.4.4" >> /etc/resolv.conf
```

To use the DNS assigned by DHCP, run:

```
ln -s /run/systemd/resolve/resolv.conf /etc/resolv.conf
```

For more information on what options are available to configure the network, see the systemd network documentation (<http://www.freedesktop.org/software/systemd/man/systemd.network.html>) .

6.2.1 Yocto Wireless

Yocto uses systemd to start wpa_supplicant, and systemd-networkd to set an IP address via a static setting or DHCP.

Scan for a network

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

An example of connecting to an open network with an SSID of "default":

```
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 manually for just this boot:

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

Use the iwconfig command to determine authentication to an access point. Before connecting it will show something like 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 connecting using WEP, also specify a network key:

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

If connecting to a WPA network use wpa_passphrase and wpa_supplicant:

```
mkdir /etc/wpa_supplicant/
wpa_passphrase "ssid name" "full passphrase" >> /etc/wpa_supplicant/wpa_supplicant-wlan0.conf
```

After generating the configuration file the wpa_supplicant daemon can be started.

```
wpa_supplicant -iwlan0 -c/etc/wpa_supplicant/wpa_supplicant-wlan0.conf -B
```

This will come back with:

```
root@ts-imx6-q:~# wpa_supplicant -iwlan0 -c/etc/wpa_supplicant.conf -B
Successfully initialized wpa_supplicant
root@ts-imx6-q:~# [ 306.924691] wlan0: authenticate with 28:cf:da:b0:f5:bb
[ 306.959415] wlan0: send auth to 28:cf:da:b0:f5:bb (try 1/3)
[ 306.968137] wlan0: authenticated
[ 306.978477] wlan0: associate with 28:cf:da:b0:f5:bb (try 1/3)
[ 306.988577] wlan0: RX AssocResp from 28:cf:da:b0:f5:bb (capab=0x1431 status=0 aid=9)
[ 307.009751] wlan0: associated
[ 307.012768] IPv6: ADDRCONF(NETDEV_CHANGE): wlan0: link becomes ready
[ 307.047989] wlcore: Association completed.
```

Use "iwconfig wlan0" to verify an "Access Point" is specified to verify a connection. This will also report the link quality to the AP.

Wireless may be associated, but this does not get an IP on the network. To connect to the internet or talk to the internal network first configure the interface. See configuring the network, but on many networks only a DHCP client is needed:

```
udhcpc -i wlan0
```

Systemd can also be configured to start wpa_supplicant on boot up.

```
# Assuming the same path for the wpa conf file as shown above
systemctl enable wpa_supplicant@wlan0
systemctl start wpa_supplicant@wlan0
```

Once this service is started it will bring up the wlan0 link and associate it to the SSID that is noted in the wpa_supplicant.conf file. Configure the IP settings the same way as a wired network.

In /etc/systemd/network/wlan0.network

```
[Match]
Name=wlan0

[Network]
DHCP=yes
```

For a static configuration create a config file for that specific interface. /etc/systemd/network/wlan0.network

```
[Match]
Name=wlan0

[Network]
Address=192.168.0.50/24
Gateway=192.168.0.1
DNS=192.168.0.1
```

For more information on what options are available to configure the network, see the systemd network documentation (<http://www.freedesktop.org/software/systemd/man/systemd.network.html>) .

6.3 Yocto Application Development

Yocto provides cross toolchains including the native tools and required arm files. First get the toolchain by right clicking and "Save as":

- x86_64 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/toolchain/poky-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-neon-toolchain-2.2.2.sh)
- i686 (<ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/toolchain/poky-glibc-i686-meta-toolchain-qt5-cortexa9hf-neon-toolchain-2.2.2.sh>)

In the case of either toolchain you would run these commands to install them:

```
chmod a+x poky-*.sh
sudo ./poky-*.sh
```

To build an application first source the environment for the toolchain:

```
source /opt/poky/2.2.2/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
# Assuming you have a hello.c:
$CC hello.c -o hello
#If you cat the environment file you can see all the paths this sets up.
$ echo $CC
arm-poky-linux-gnueabi-gcc -march=armv7-a -marm -mthumb-interwork -mfloat-abi=hard -mfpu=neon -mtune=cortex-a9 --sysroot=/opt/poky/2.2.2/sysroots/
```

It is also possible to develop over the serial console or ssh on the board itself. Yocto includes development tools such as vim, gcc, g++, gdb, make, autoconf, binutils, and more. See the next sections for using the cross toolchain with IDEs.

6.3.1 Configure Qt Creator IDE

Note: This guide is intended for our stock Yocto image using systemd. On custom images the directions should apply if a toolchain is compiled. "bitbake meta-toolchain-qt5", and update the paths if you are using a different distribution.

Install the qtcreator tool on a host Linux PC. Any recent version from a modern Linux distribution should be sufficient and work without issue. On a Debian/Ubuntu desktop, run:

```
sudo apt-get update && sudo apt-get install qtcreator -y
```

You will also need to download the SDK which includes the Qt support (Right click and save as):

- x86_64 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/toolchain/poky-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-neon-toolchain-2.2.2.sh)
- i686 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/toolchain/poky-glibc-i686-meta-toolchain-qt5-cortexa9hf-neon-toolchain-2.2.2.sh)

You can install these with:

```
sudo bash ./poky-*.sh
```

These instructions assume the path will be default at "/opt/poky/2.2.2/".

Note: An environment script has to be sourced before *every* execution of qtcreator. Without this, builds will fail.

```
source /opt/poky/2.2.2/environment-setup-cortexa9hf-neon-poky-linux-gnueabi
qtcreator
```

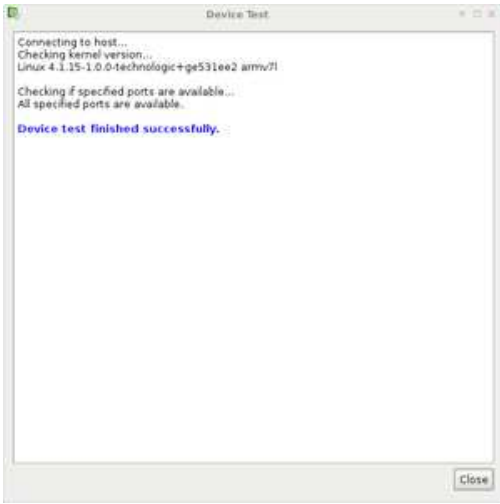
Qt Creator needs to be configured to build using this toolchain. Once Qt Creator is launched, select Tools->Options->Devices. Click "Add," select "Generic Linux Device," and then click "Start Wizard".



On the next page specify the IP address or hostname of the device running Yocto. In this example, the unit has an IP address of 192.168.2.45 obtained by DHCP. The default Yocto image will use "root" with no password to connect. Set the name to TSIMX6.



It will then verify connectivity. Click close and continue.



If this returns an error: "SSH connection failure: SSH Protocol error: Server and client capabilities don't match. Client list was: aes128-cbc,3des-cbc.

Note: Server list was chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com.". If this happens connect to the board's console and edit /etc/ssh/sshd_config and append the line "Ciphers +aes128-cbc". Reset sshd, or reboot the board and try again. Upgrading Qt Creator may also resolve this issue.

Note: The paths given in the images may not match the latest toolchain, but are meant to show where these values would go. Follow the text appropriate to the architecture of your host PC for the correct values

In the left column of the Options menu, select "Build & Run". On the "Qt Versions" tab, click "Add" in the upper right to configure the TS Kit. Qt Creator may see the "qmake" binary added to your path from the sourced environment script. If this is detected add in the string "TSIMX6" to the title. If not, add the full path and ensure the version name is set to "TSIMX6 QT 5.7.1". This will allow it to be recognized when setting the right binary for the kit.

i686 /opt/poky/2.2.2/sysroots/x86-pokysdk-linux/usr/bin/qt5/qmake

x86_64 /opt/poky/2.2.2/sysroots/x86_64-pokysdk-linux/usr/bin/qt5/qmake



On the "Compilers" tab click "Add", and select "GCC". Set the Name to "TSIMX6 GCC". For the "Compiler Path" use one of the following:

i686 /opt/poky/2.2.2/sysroots/i686-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++

x86_64 /opt/poky/2.2.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-g++



On the "Debuggers" tab click "Add". For name, specify "TSIMX6 GDB". For the paths, specify the location of gdb with one of the following:

i686 /opt/poky/2.2.2/sysroots/i686-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb
 x86_64 /opt/poky/2.2.2/sysroots/x86_64-pokysdk-linux/usr/bin/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb



On the "Kits" tab click "Add". For "Name", enter "TSIMX6". Set device type to "Generic Linux Device". Set the device to "TSIMX6 (default for Generic Linux)". Set Qt mkspec to:

/opt/poky/2.2.2/sysroots/cortexa9hf-neon-poky-linux-gnueabi/usr/lib/qt5/mkspecs/linux-oe-g++

Make sure there is no space at the end.

Set "Compiler" to "TSIMX6 GCC". Set "Debugger" to "TSIMX6 GDB". Set the "Qt version" to "TSIMX6 QT 5.7.1". Finally, click Apply.

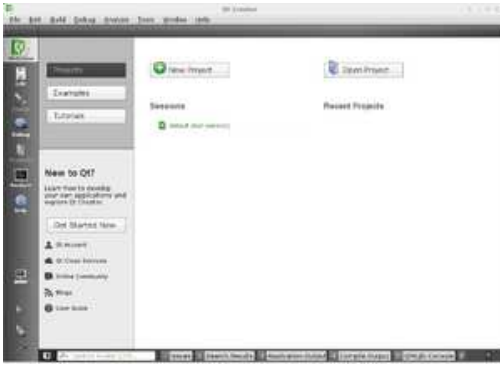


Note: If there is a red exclamation point over the kits icon, it indicates that the compiler ABI does not match. In this case, you will need to revisit the "Compiler", "Debugger", and "Qt Versions" tabs, and browse the host PC for these files manually rather than copy/pasting the paths from these instructions. This is a bug in Ubuntu 16.04's Qt Creator, and may be in later versions as well.

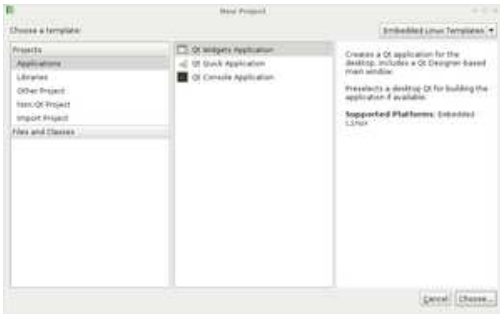
At this point Qt Creator is set up to begin a hello world project.

6.3.1.1 Qt Creator Hello World

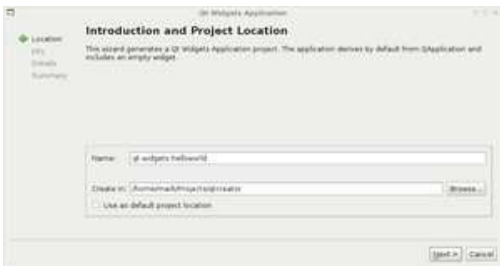
Open the Qt Creator IDE and click "New Project".



Qt provides multiple templates for application development. For this example select the default "Qt Widgets Application".



Specify the location for your project. Keep in mind that the compile process will create more build paths in the "Create In:" path.



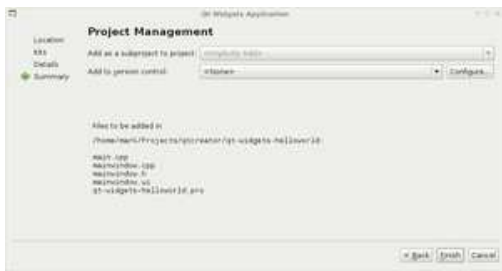
Next, select the kit. The TSIMX6 is the kit we set up in the last section, but you may have other kits preinstalled on your system. These can be used for testing graphical development on your PC. Keep in mind distribution versions may contain different functionality.



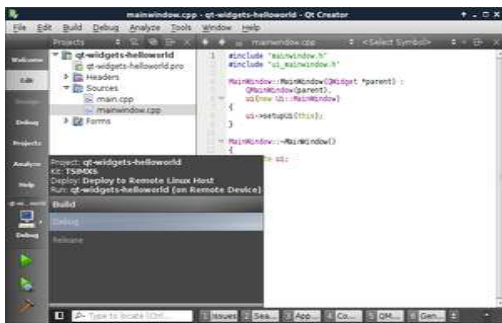
Next select the class and filename information. This example will use the defaults.



Select any version control for the project. The example will use none and finish the wizard. This will generate the new project.



Click the button under "Help" on the left column, and select "TSIMX6" debug. If you only have one kit selected this will be default.

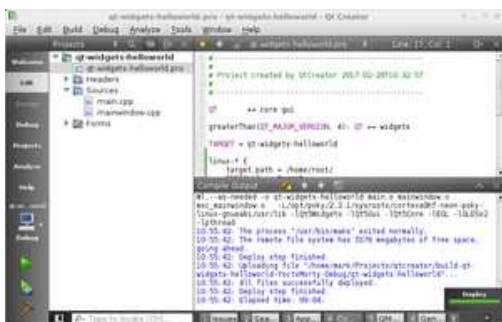


Now return to edit, and open the Qt project file (qt5-helloworld.pro). Add in these lines anywhere after the target is specified:

```
linux-* {
    target.path = /home/root/
    INSTALLS += target
}
```



At this point click the green allow in the bottom left to run the application. This can also be launched from the menu at Build->Run.



From here, you can begin customizing your application. Refer to the official Qt documentation for more information

- Qt 5 Documentation (<http://qt-project.org/doc/qt-5/index.html>)

6.3.2 Yocto Hide Cursor

The default image includes the xcursor-transparent icon theme. This can hide the mouse pointer. To enable this, run these commands:

```
mkdir -p ~/.icons/default/
echo "[Icon Theme]" > ~/.icons/default/index.theme
echo "Inherits=xcursor-transparent" >> ~/.icons/default/index.theme
# Now reset x, or reset the unit and the cursor will be invisible.
```

6.4 Yocto Startup Scripts

To have a custom headless application start up at boot a systemd service needs to be created. Create the file `/etc/systemd/system/yourapp.service` with contents similar to below:

```
[Unit]
Description=Run an application on the i.MX6

[Service]
Type=simple
ExecStart=/usr/local/bin/your_app_or_script

[Install]
WantedBy=multi-user.target
```

If you depend on networking you should add `"After=network.target"` in the Unit section. Once this file is in place, it can be added to automatic startup with the following:

```
# Start your app on bootup, but will not start it now
systemctl enable yourapp.service

# Start your app now, but doesn't change auto startup
systemctl start yourapp.service
```

Note: See the systemd documentation (<http://www.freedesktop.org/software/systemd/man/systemd.service.html>) for in depth documentation on services.

To set up a graphical application startup, change the file: `/usr/bin/mini-x-session`

At the end of the script replace `"matchbox-terminal"` with your application (absolute path may need to be specified):

```
matchbox-terminal&
exec matchbox-window-manager
```

The `exec` statement must be last in the script in order to take over this script's PID for correct operation.

6.5 Custom Build Yocto

If our stock Yocto distribution does not meet all of your needs, it is possible to re-build it with a custom set of features. Including less options for a smaller footprint, or more packages to add more features.

While we may provide guidance, our free support does not include every situation that can cause a build failure in generating custom images.

- Build Yocto Distribution

7 QNX

QNX is an RTOS that supports the i.MX6 CPU. We provide a BSP for the TS-4900 and TS-7970 quad core or solo based on QNX Neutrino (<http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html>) 6.6.0. The supporting files are available here:

- Disk Image: `ts7970-qnx-6.6.0-latest.dd.bz2` (<ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-qnx/images/ts7970-qnx-6.6.0-latest.dd.bz2>) (md5) (<ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-qnx/images/ts7970-qnx-6.6.0-latest.dd.bz2.md5>)
- Quad core source `BSP_freescale-imx6q-ts7970-latest.tar.gz` (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-qnx/source/BSP_freescale-imx6q-ts7970-latest.tar.gz) (md5) (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-qnx/source/BSP_freescale-imx6q-ts7970-latest.tar.gz.md5)
- Solo core source `BSP_freescale-imx6dl-ts7970-latest.tar.gz` (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-qnx/source/BSP_freescale-imx6dl-ts7970-latest.tar.gz) (md5) (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-qnx/source/BSP_freescale-imx6dl-ts7970-latest.tar.gz.md5)

We provide support for booting QNX on our platforms, but further support is provided by QNX

- QNX Support (<http://www.qnx.com/support/>)

Known Working:

- UARTs 1-5
- Ethernet
- I2C 1, I2C 2
- SD (/dev/hd0)
- eMMC (/dev/emmc0)
- USB Host
- SPI NOR (/dev/fs0)
- HDMI (TS-7970 only)
- LCD Interface (TS-TPC-8390 with TS-4900 only)
- RS485

Known not working:

- WIFI
- FPGA based UARTs

Not yet tested:

- I210 (Second gig eth)

7.1 QNX BSP

Before compiling QNX be sure to edit the file: `src/hardware/startup/boards/imx6x/ts7970/board.h` Set either `BOARD_TS7970` or `BOARD_TS4900` depending on the target board.

We have also included a port of `tshwctl` which is used to access the FPGA. This allows you to read/write FPGA registers and to change the crossbar. For example, to set up auto TXEN on the TS-7970 RS-485 port (/dev/ser4):

```
export MB_TXD=TTYSER4_TXD
export TTYMAX1_RXD=GPIO
export TTYSER4_RXD=MB_RXD_485
export TXD_232_COM=GPIO
export MB_TX_EN_485=TTYSER4_TXEN
tshwctl -b 0x7970 -s
tshwctl -b 0x7970 -c
```

This will print out the modified state of the crossbar. The relevant pins are now:

```
TTYSER4_RXD ( in) ( 0) MB_RXD_485
MB_TXD ( in) ( 0) TTYSER4_TXD
MB_TX_EN_485 ( in) ( 0) TTYSER4_TXEN
TTYMAX1_RXD ( in) ( 0) GPIO
TXD_232_COM ( in) ( 0) GPIO
```

Use `tshwctl` to specify the baud rate and mode of the uart so the TX enable pin will be automatically toggled.

```
tshwctl -b 0x7970 -a 4 -x 115200 -i 8n1
```

/dev/ser4 is now configured for RS485 traffic.

7.2 QNX Booting

Write the example image to a disk.

```
gzip -d ts7970-qnx-6.6.0-20150707.dd.bz2
#Replace sdx with your device. Try lsblk to find your SD card.
sudo dd if=ts7970-qnx-6.6.0-20150707.dd bs=4M of=/dev/sdx
sync
```

Reinsert or partprobe the disk, and there will be a single partition present. The partition includes the QNX IFS, and a u-boot script. On startup the imx6 is configured to launch the hush script. If present, at /boot/boot.ub on either the SD or eMMC depending on if the SD boot jumper is present. The script loads the FPGA, then copies the QNX ifs into memory and jumps into it to begin execution.

8 Android

This Android distribution is based off of Freescale's port of AOSP to the i.MX6 platform. This allows users to run existing APKs to use this platform with no modifications, or develop new projects using Android Studio.

8.1 Getting Started with Android

Android must be run from the eMMC. This can be written with the USB production tool, or from the SD card. To use the USB drive, follow the instructions here, and download the image and copy it to the USB drive as emmcimage.dd.bz2.

Download the Android image here:

- android-7.1.1-tsimx6-tiwifi-latest.dd.bz2 (<ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7990-linux/distributions/android/>) (md5 (<ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7990-linux/distributions/android/android-7.1.1-tsimx6-tiwifi-latest.dd.bz2.md5>))

To load from the SD card, boot up to any Linux distribution from the SD card such as the default Yocto. Once booted here, run:

```
wget -qO- ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7990-linux\
/distributions/android/android-7.1.1-tsimx6-tiwifi-\
latest.dd.bz2 | bzipcat | dd bs=4M of=/dev/mmcblk2 conv=fsync
```

This will download it, decompress it, and write it to the eMMC drive. Reboot and boot into Android.

8.2 Android Networking

On startup android will automatically start dhcpcd on eth0, or WIFI can be configured via the Settings->Wi-Fi menu.

8.3 Android Software Development

AOSP development works exactly the same as on an Android phone, except the Google APIs associated with the store are not available. See The android documentation for getting started on development: <http://developer.android.com/training/basics/firstapp/index.html>

8.4 Android Manually Install APK

APKs can be installed just like on any other Android device. On the device go to settings->About Tablet and press the "build number" until the text states "You are now a developer". Go back to Settings and there is now a "Developer Options" menu. Under Debugging enable USB Debugging. You should now be able to run adb commands to install apk files.

```
adb install </path/to/app.apk>
```

9 Backup / Restore

9.1 MicroSD Card

These instructions assume you have an SD card with one partition. Most SD cards ship this way by default. If the card has had its partition table modified this can be corrected with a tool like 'gparted' or 'fdisk'.

Plug the SD card into a USB reader and connect it to a linux workstation PC. Newer distributions include a utility called 'lsblk' which lists all block devices like a USB SD card reader:

```
lsblk
```

```
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sdY 8:0 0 400G 0 disk
├─sdY1 8:1 0 398G 0 part /
├─sdY2 8:2 0 1K 0 part
└─sdY5 8:5 0 2G 0 part [SWAP]
sr0 11:0 1 1024M 0 rom
sdX 8:32 1 3.9G 0 disk
├─sdX1 8:33 1 7.9M 0 part
└─sdX2 8:34 1 2M 0 part
```



```
└─sdX3  8:35  1    2M  0 part
└─sdX4  8:36  1   3.8G 0 part
```

In this case the SD card is 4GB, so sdX is the target device. Note that on your system, sdX will not be a real device, it could be sda, sdb, mmcblk0, etc. Technologic Systems is not responsible for any damages cause by using the improper device node for imaging an SD card.

After plugging in the device after Linux has booted you can use dmesg to print out the kernel log. When the USB drive is added it will append to the end of that file. Try running:

```
dmesg | tail -n 100
```

```
scsi 54:0:0:0: Direct-Access   Generic Storage Device   0.00 PQ: 0 ANSI: 2
sd 54:0:0:0: Attached scsi generic sg2 type 0
sd 54:0:0:0: [sdX] 3862528 512-byte logical blocks: (3.97 GB/3.84 GiB)
```

In this case, sdXc is shown as a 3.97GB card. Note that on your system, sdX will not be a real device, it could be sda, sdb, mmcblk0, etc. Technologic Systems is not responsible for any damages cause by using the improper device node for imaging an SD card.

The following commands will reformat the first partition of the SD card, and unpack the latest filesystem on there:

```
# Verify nothing else has this mounted
sudo umount /dev/sdX1

sudo mkfs.ext3 /dev/sdX1
sudo mkdir /mnt/sd
sudo mount /dev/sdX1 /mnt/sd/
wget ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7990-linux/distributions/debian/debian-armhf-jessie-latest.tar.bz2

sudo tar --numeric-owner -xf debian-armhf-jessie-latest.tar.bz2 -C /mnt/sd
sudo umount /mnt/sd
sync
```

The ext4 filesystem can be used instead of ext3, but it may require additional options. U-Boot does not support the 64bit addressing added as the default behavior in recent revisions of mkfs.ext4. If using e2fsprogs 1.43 or newer, the options "-O ^64bit,^metadata_csum" must be used with ext4 for proper compatibility. Older versions of e2fsprogs do not need these options passed nor are they needed for ext3.

Once written, the files on disk can be verified to ensure they are the same as the source files in the archive. Reinsert the disk to flush the block cache completely, then run the following commands:

```
mount /dev/sdX1 /mnt/sd
cd /mnt/sd/
sudo md5sum --quiet -c md5sums.txt
cd -
umount /mnt/sd
sync
```

The md5sum command will report what differences there are, if any, and return if it passed or failed.

9.2 eMMC

Write the image:

These commands assume you are booted to the SD card, and

```
# Verify nothing else has this mounted
umount /dev/mmcblk2p1

mkfs.ext3 /dev/mmcblk2p1
mkdir /mnt/emmc
mount /dev/mmcblk2p1 /mnt/emmc
wget ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/ts-x11-image-tsimx6-latest.rootfs.tar.bz2
tar --numeric-owner -xf ts-x11-image-tsimx6-latest.rootfs.tar.bz2 -C /mnt/emmc
umount /mnt/emmc
sync
```

The ext4 filesystem can be used instead of ext3, but it may require additional options. U-Boot does not support the 64bit addressing added as the default behavior in recent revisions of mkfs.ext4. If using e2fsprogs 1.43 or newer, the options "-O ^64bit,^metadata_csum" must be used with ext4 for proper compatibility. Older versions of e2fsprogs do not need these options passed nor are they needed for ext3.

After it is written you can verify the data was written correctly.

```
# Drop any bLock cache
echo 3 > /proc/sys/vm/drop_caches
mount /dev/mmcblk2p1 /mnt/emmc
cd /mnt/emmc/
sudo md5sum -c md5sums.txt
umount /mnt/emmc
sync
```

The md5sum command will report what differences there are, if any, and return if it passed or failed.

Backup the image:

First boot the board to any SD image. The SD should have enough space for the compressed image of your eMMC. From our default image this is ~500MB. To create an image to your SD card:

```
mkdir /mnt/sd/
mount /dev/mmcblk1p1 /mnt/sd/
cd /mnt/sd/
tar --numeric-owner -cjf /root/backup.tar.bz2 *
cd -
umount /mnt/sd/
```

10 Compile the Kernel

This board uses a 4.1.15 kernel on most all images. Our kernels are based on NXP's which are changed from upstream for their board support. We change them to support the various hardware we use with this processor.

- [embeddedarm/linux-3.10.17-imx6](https://github.com/embeddedarm/linux-3.10.17-imx6) (<https://github.com/embeddedarm/linux-3.10.17-imx6>)
- The "imx_4.1.15_1.0.0_ga" branch includes support for our TS-4900, TS-7970, TS-7990, and TS-4100.
 - This kernel works with all our Debian releases, Ubuntu, Yocto Jethro, and Yocto Krogtho.

You can pick the branch below. The kernel can be rebuilt by cross compiling from an X86/X86_64 Linux. Our default kernels are rebuilt using the toolchains built by Yocto. You can download the appropriate cross toolchain for your Linux system here:

- X86_64 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/jethro/toolchains/poky-glibc-x86_64-meta-toolchain-qt5-cortexa9hf-vfp-neon-toolchain-2.0.2.sh)
- i686 (<ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/jethro/toolchains/poky-glibc-i686-meta-toolchain-qt5-cortexa9hf-vfp-neon-toolchain-2.0.2.sh>)

```
chmod a+x poky-*toolchain-*.sh
sudo ./poky-*toolchain-*.sh
```

This will ask for the install directory for the toolchain. Our instructions will assume the default path is used.

This also requires several tools from your distribution. For Ubuntu/Debian:

```
sudo apt-get install git build-essential lzop u-boot-tools libncursesw5-dev
```

Once those are installed:

```
# This will pull our shared kernel, using the 4.1.15 branch, use a folder linux-tsimx, and only
# download the latest changes.
git clone https://github.com/embeddedarm/linux-3.10.17-imx6.git -b imx_4.1.15_1.0.0_ga linux-tsimx --depth 1
cd linux-tsimx

# These two must be run each time you open a terminal to build the kernel.
source /opt/poky/2.0.2/environment-setup-cortexa9hf-vfp-neon-poky-linux-gnueabi
export LOADADDR=0x10008000

make ts4900_defconfig

## Make any changes in "make menuconfig" or driver modifications, then compile
make && make uImage
```

To install this to a board you would use a USB SD reader and plug in the card. Assuming your Linux rootfs is all on "sdcl":

```
export DEV=/dev/sdc1
sudo mount "$DEV" /mnt/sd
sudo rm /mnt/sd/boot/uImage
sudo cp arch/arm/boot/uImage /mnt/sd/boot/uImage
```

```

sudo cp arch/arm/boot/dts/imx6*ts*.dtb /mnt/sd/boot/
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

```

10.1 Change Kernel Splash Screen

The kernel splashscreen allows 224 colors. It also allows up to the full screen resolution, but for fastest boot speed it should be kept as small as possible. The image will be centered around a black background.

To convert your image, for example, "mylogo.png":

```

convert mylogo.png mylogo.ppm
ppmquant 224 mylogo.ppm > mylogo-224.ppm
pnmnoraw mylogo-224.ppm > logo_user_clut224.ppm
cp logo_user_clut224.ppm <kernel build sources>/drivers/video/logo/

```

Now recompile the kernel following the guide in the previous section.

Add the kernel cmdline "logo.nologo" in u-boot to completely disable the splash screen.

11 Production Mechanism

On startup if SW1 is pressed when power is applied then TS-7970's U-boot will look in a USB drive for a file called /tsinit.ub. If found it will copy this to \${loadaddr} and "source \${loadaddr}" to run this u-boot script. This is intended for the initial production of boards and allows mass programming boards with a USB thumbdrive.

The blast image and scripts require a minimum of 50 MB; this plus any disk images or tarballs used dictate the minimum disk size required. The USB drive must have at least 1 partition, with the first partition being formatted ext2/3 or fat32/vfat.

Note: The ext4 filesystem can be used instead of ext3, but it may require additional options. U-Boot does not support the 64bit addressing added as the default behavior in recent revisions of mkfs.ext4. If using e2fsprogs 1.43 or newer, the options "-O ^64bit,^metadata_csum" must be used with ext4 for proper compatibility. Older versions of e2fsprogs do not need these options passed nor are they needed for ext3.

```

# This assumes your thumbdrive is /dev/sdc:
sudo mkfs.ext3 /dev/sdc1
sudo mkdir /mnt/sd/
sudo mount /dev/sdc1 /mnt/sd/
sudo tar --numeric-owner -xf /path/to/tsimx6_usb_blaster-latest.tar.bz2 -C /mnt/sd/

# Now you would create your images on the /mnt/sd/, but for
# an example these steps would write our latest debian image:
sudo wget -O /mnt/sd/emmcimage.tar.bz2 http://ftp.embeddedarm.com/ftp/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-j
# You can use a symlink to write the same image to sd
sudo ln -s /mnt/sd/emmcimage.tar.bz2 /mnt/sd/sdimage.tar.bz2
sudo umount /mnt/sd
sync

```

The USB drive boots into a small buildroot initramfs environment with filesystem and partitioning tools. This can be used to format SD, eMMC, SATA, or even rewrite u-boot and its environment. The buildroot starts up and calls /blast.sh on the USB device. By default this script is set up to look for a number of of specific files on the USB disk and write to media on the host device. Upon completion of the script, the green or red LEDs will blink to visually indicate a pass or fail of the script. This script can be used without modification to write images from USB with these filenames:

SD Card	sdimage.tar.bz2	Tar of the filesystem. This will repartition the SD card to 1 ext4 partition and extract this tar to the filesystem. If present, a /md5sums.txt will be checked and every file can be verified on the filesystem. This md5sums file is optional and can be omitted, but it must not be blank if present.
	sdimage.dd.bz2	Disk image of the card. This will be written to mmcblk0 directly. If present a sdimage.dd.md5 will cause the written data on the SD card to be read back and verified against this checksum.
eMMC	emmcimage.tar.bz2	Tar of the filesystem. This will repartition the eMMC to 1 ext4 partition and extract this tar to the filesystem. If present, a /md5sums.txt will be checked and every file can be verified on the filesystem. This md5sums file is optional and can be omitted, but it must not be blank if present.
	emmcimage.dd.bz2	Disk image of the card. This will be written to mmcblk1 directly. If present a emmcimage.dd.md5 will cause the written data on the eMMC to be read back and verified against this checksum.
SATA [1]	sataimage.tar.bz2	Tar of the filesystem. This will repartition the SATA drive to 1 ext4 partition and extract this tar to the filesystem. If present, a /md5sums.txt will be checked and every file can be verified on the filesystem. This md5sums file is optional and can be omitted, but it must not be blank if present.
	sataimage.dd.bz2	Disk image of the card. This will be written to sda directly. If present a sataimage.dd.md5 will cause the written data on the SATA to be read back and verified against this checksum.
SPI	u-boot.imx	This will rewrite u-boot on the SPI flash. This will also verify the imx_type variables match before allowing it to be rewritten to make sure the RAM configs match. If u-boot.imx.md5 is present the SPI flash will be read back and verified.

- ↑ SATA is only present on the Dual/Quad CPUs

Most users should be able to use the above script without modification, but our buildroot sources are available from our github repo (<https://github.com/embeddedarm/buildroot-2017.05>). To build the whole setup and create a USB drive, the following commands can be used. This will wipe any data on the specified partition and replace it with an ext2 formatted filesystem. This filesystem will have all of the necessary files written to it to create a bootable USB drive. Note that this must be the first partition of the disk.

```
# Assuming /dev/sdc1 is your usb drive's first partition
make ts4900_defconfig && make && sudo make_usb_prog.sh /dev/sdc1
```

12 Features

12.1 ADC

The TS-7970 includes 3 channels of current-loop sensing ADC that can sample a 4-20mA current loop at about 2/3 samples per second. Note, these ADC cannot be used for voltage sensing. These are accessed using the tsmicroctl utility:

```
root@ts-imx6:# tsmicroctl --info
VDD_ARM_CAP=1216
VDD_HIGH_CAP=2618
VDD_SOC_CAP=1246
VDD_ARM=1456
SILAB_P10=0x39B
SILAB_P11=0x0
SILAB_P12=0x0
VIN=12241
V5_A=5207
V3P1=3276
DDR1_1P5V=1571
V1P8=1894
V1P2=1262
RAM_VREF=783
V3P3=3543
SILABREV=1
SILAB_P10_UA=21472
SILAB_P11_UA=0
SILAB_P12_UA=0
```

The other samples are the various voltages on the board. The terminal block ADC values are returned with SILAB_Pnn and SILAB_Pnn_UA. These include both raw and microamp values.

12.2 Bluetooth

The WIFI option on the board also includes a bluetooth 4.0 LE module. To connect this to bluez first pulse the BT_EN pin, and then call hciattach:

```
# Install bluez if it is not already present
apt-get update
apt-get install bluez bluez-tools
```

```
# Loads firmware for the wifi+BT module
ifconfig wlan0 up

echo 237 > /sys/class/gpio/export
echo low > /sys/class/gpio/gpio237/direction
echo high > /sys/class/gpio/gpio237/direction
sleep .1
hciattach /dev/ttymx1 texas 3000000
hciconfig hci0 up
```

Once this is loaded you can scan for devices with:

```
hcitool scan
```

This will return a list of devices such as:

```
14:74:11:A1:1E:C9      BlackBerry 8530
```

Bluez has support for many different profiles for HID, A2DP, and many more. Refer to the Bluez documentation for more information.

12.3 CAN

The TS-7970 CAN ports are located on the #COM2 Header and the #Terminal_Blocks.

The i.MX6 includes 2 CAN controllers which support the SocketCAN interface. Before proceeding with the examples, see the Kernel's CAN documentation here (<https://www.kernel.org/doc/Documentation/networking/can.txt>).

This board comes preinstalled with can-utils. These can be used to communicate over a CAN network without writing any code. The candump utility can be used to dump all data on the network

```
## First, set the baud rate and bring up the device:
ip link set can0 type can bitrate 250000
ip link set can0 up

## Dump data & errors:
candump can0 &

## Send the packet with:
#can_id = 0x7df
#data 0 = 0x3
#data 1 = 0x1
#data 2 = 0x0c
cansend can0 -i 0x7Df 0x3 0x1 0x0c
## Some versions of cansend use a different syntax. If the above
## commands gives an error, try this instead:
#cansend can0 7DF#03010C
```

The above example packet is designed to work with the Ozen Elektronik myOByDic 1610 ECU simulator to read the RPM speed. In this case, the ECU simulator would return data from candump with:

```
<0x7e8> [8] 04 41 0c 60 40 00 00 00
<0x7e9> [8] 04 41 0c 60 40 00 00 00
```

In the output above, columns 6 and 7 are the current RPM value. This shows a simple way to prove out the communication before moving to another language.

The following example sends the same packet and parses the same response in C:

```
#include <stdio.h>
#include <pthread.h>
#include <net/if.h>
#include <string.h>
#include <unistd.h>
#include <net/if.h>
#include <sys/ioctl.h>
#include <assert.h>
#include <linux/can.h>
#include <linux/can/raw.h>

int main(void)
{
    int s;
    int nbytes;
    struct sockaddr_can addr;
    struct can_frame frame;
    struct ifreq ifr;
    struct iovec iov;
```

```

struct msghdr msg;
char ctrlmsg[MSG_SPACE(sizeof(struct timeval)) + MSG_SPACE(sizeof(__u32))];
char *ifname = "can0";

if((s = socket(PF_CAN, SOCK_RAW, CAN_RAW)) < 0) {
    perror("Error while opening socket");
    return -1;
}

strcpy(ifr.ifr_name, ifname);
ioctl(s, SIOCGIFINDEX, &ifr);
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;

if(bind(s, (struct sockaddr *)&addr, sizeof(addr)) < 0) {
    perror("socket");
    return -2;
}

/* For the ozen myOByDic 1610 this requests the RPM gauge */
frame.can_id = 0x7df;
frame.can_dlc = 3;
frame.data[0] = 3;
frame.data[1] = 1;
frame.data[2] = 0x0c;

nbytes = write(s, &frame, sizeof(struct can_frame));
if(nbytes < 0) {
    perror("write");
    return -3;
}

iov.iov_base = &frame;
msg.msg_name = &addr;
msg.msg_iov = &iov;
msg.msg_iovlen = 1;
msg.msg_control = &ctrlmsg;
iov.iov_len = sizeof(frame);
msg.msg_namelen = sizeof(struct sockaddr_can);
msg.msg_controllen = sizeof(ctrlmsg);
msg.msg_flags = 0;

do {
    nbytes = recvmsg(s, &msg, 0);
    if (nbytes < 0) {
        perror("read");
        return -4;
    }

    if (nbytes < (int)sizeof(struct can_frame)) {
        fprintf(stderr, "read: incomplete CAN frame\n");
    }
} while(nbytes == 0);

if(frame.data[0] == 0x4)
    printf("RPM at %d of 255\n", frame.data[3]);

return 0;
}

```

See the Kernel's CAN documentation here (<https://www.kernel.org/doc/Documentation/networking/can.txt>) . Other languages have bindings to access CAN such as Python using C-types (<https://bitbucket.org/hardbyte/python-can>) , Java using JNI (<https://github.com/entropia/libsocket-can-java>) .

12.4 COM Ports

This board uses UARTs from both the CPU and the FPGA. The CPU uart 0 (/dev/ttymx0) is dedicated to console for Linux and U-boot and not suggested to be reused. The other CPU UARTs for ttymx1-4 are usable for end applications. These support up to 5Mb/s UART data with DMA.

The FPGA also emulates a MAX3100 UART interface accessible at /dev/ttyMAX0-2. These UARTs support a total throughput of about 115200^[1]. These UARTs include hardware that makes implementing RS-485 half duplex software very simple. If higher throughput is needed, the FPGA crossbar can be adjusted to use a CPU UART with TXEN support instead.

Note: Our SPI interface matches the max3100 almost entirely, except optionally a single 8-bit transaction can be sent to act as a chip select between the three uarts supported on our interface. The default FPGA supports 3 UARTs on this interface. This is handled automatically by our driver (max3100-ts).

RS-485 half duplex's direction control is built into the ttyMAX UARTs. By default they are connected to the RS-485 ports and no code is required for the transmit enable to toggle. The CPU UARTs however do not have transmit enable built in. The FPGA however includes a core that will toggle transmit enable for ttymx1/ttymx3, but it needs to know the baud rate, and symbol size (data bits, parity, stop bits).

For example:

```
# Configure mxc1 and mxc3 as 115200, 8n1
stty -F /dev/ttymx1 115200 cs8 -cstopb
tshwctl --autotxen 1

stty -F /dev/ttymx3 115200 cs8 -cstopb
tshwctl --autotxen 3
```

The tshwctl tool will read the UART settings from the moment when it is run and it sets up the FPGA's timing for TXEN. Your baud rate and mode settings should be set before running this.

Using the FPGA for either the ttyMAX uarts, or the CPU uarts, the TXEN timing will happen well under a single bit time ^[2] of any baud rate possible by the hardware.

All of these UARTs are accessed using the standard /dev/ interfaces. See these resources for information on programming with UARTs 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>)

1. ↑ Idle periods do not count towards the total throughput limitation.
2. ↑ This is a requirement for half duplex MODBUS

The #FPGA includes a crossbar to select where UARTs are routed so these can be changed, but these are the default mappings:

UART	Type	TX (or +)	RX (or -)
ttymxc0	USB	USB Device	USB Device
ttymxc1	1.8V TTL (onboard only)	Onboard Bluetooth RX	Onboard Bluetooth TX
ttymxc2	TTL (5V Tolerant)	HD1 Header pin 12	HD1 Header pin 10
ttymxc3	RS232	COM2 Header pin 3	COM2 Header pin 2
ttymxc4	RS232	P1-B Terminal Block pin 7	P1-B Terminal Block pin 8
ttyMAX0	RS485	P1-A Terminal Block pin 2, COM2 Header pin 1	P1-A Terminal Block pin 3, COM2 Header pin 6
ttyMAX1	RS485	RJ45 2W-Modbus pin 4	RJ45 2W-Modbus pin 5
ttyMAX2	RS232	COM2 Header pin 7	COM2 Header pin 8

12.5 CPU

The i.MX6 is an armv7a Cortex-A9 by NXP. The CPU itself is available in 792MHz, 996MHz, and 1.2GHz with a solo, dual, or quad core processor.

Refer to NXP's documentation for in depth documentation on these CPU cores:

- i.MX6S (<http://www.nxp.com/products/automotive-products/microcontrollers-and-processors/arm-mcus-and-mpus/i.mx-application-processors/i.mx-6-processors/i.mx-6solo-processors-single-core-multimedia-3d-graphics-arm-cortex-a9-core:i.MX6S>)
- i.MX6Q (<http://www.nxp.com/products/automotive-products/microcontrollers-and-processors/arm-mcus-and-mpus/i.mx-application-processors/i.mx-6-processors/i.mx-6quad-processors-high-performance-3d-graphics-hd-video-arm-cortex-a9-core:i.MX6Q>)

12.6 eMMC

This board includes a Micron eMMC module with builds that have "4096F" in the part number. Our off the shelf builds are 4GiB, but up to 64GiB are available for larger builds. The eMMC flash appears to Linux as an SD card at /dev/mmcblk2. Our default programming will include one partition programmed with our Yocto image.

The eMMC are like SD cards in that they should not be powered down during a write/erase cycle. This eMMC module includes support for setting a fuse for a "Write Reliability" mode, and a "psuedo SLC" mode. With both of these enabled then any writes will be atomic to 512B. If a sector is being written during a power loss, a block is guaranteed to have either the old or new data. This scheme is far more resilient to power loss than more traditional flash media. In cases of old 512B data fsck will still be able to recover a mountable filesystem. In cases where the corrupted file is a database it can still need a mechanism for recovery.

When this pSLC mode is turned on it will reduce the available space to under half, and reduce the write speed.

See our post on preventing filesystem corruption (<https://www.embeddedarm.com/blog/preventing-filesystem-corruption-in-embedded-linux/>).

The `mmc-utils` package is used to enable these modes. First determine the exact size of the flash you're using:

```
mmc extcsd read /dev/mmcblk2 | grep MAX_ENH_SIZE_MULT -A 1
```

```
Max Enhanced Area Size [MAX_ENH_SIZE_MULT]: 0x0001cd  
i.e. 1888256 KiB
```

So in this case, 1888256 KiB is the max size of the enhanced partition. This number should be used for the `enh_area` command:

```
mmc write_reliability set -n 0 /dev/mmcblk2  
mmc enh_area set -y 0 1888256 /dev/mmcblk2
```

WARNING: Setting either of those modes is permanent. Using the wrong value it is possible to brick eMMC which will not be covered by the warranty. Evaluation units with fuses set will not be accepted through returns.

After this is run, reboot the board. On all future boots the eMMC will be detected at the smaller size. Changing the enhanced area will erase the drive. After these `mmc` commands the disk will need to be rewritten.




12.7 Enclosures

Every enclosure includes a front label which exposes 1 button, and 4 status LEDs.



TS-ENC797

The TS-7970 is available with 3 enclosures matching the various build options:

Enclosure Model	Images	Supported Boards
<p>TS-ENC797</p>		<ul style="list-style-type: none"> ■ TS-7970-1G-4GF-S8S-RTC-I ■ TS-7970-2G-4GF-Q10S-RTC-E
<p>TS-ENC797-CP</p>		<ul style="list-style-type: none"> ■ TS-7970-1G-4GF-S8S-RTC-CP-WIFI-I ■ TS-7970-2G-4GF-Q10S-RTC-CP-WIFI-E
<p>TS-ENC797-CP-WIFI</p> <p>This enclosure includes WIFI Antenna and UFL to SMA cable.</p>		<ul style="list-style-type: none"> ■ TS-7970-1G-4GF-S8S-RTC-CP-WIFI-I with WIFI antenna ■ TS-7970-2G-4GF-Q10S-RTC-CP-WIFI-E with WIFI antenna

These 3 enclosures can also be ordered with a DIN clip as TS-ENC797-DIN, TS-ENC797-CP-DIN, and TS-ENC797-CP-WIFI-DIN.



12.8 FPGA

The Lattice MachXO2 FPGA provides several features used by default on the TS-7970:

- auto TX enable for RS-485 half duplex
- DIO expander
- UART/DIO crossbar
- Clock generator

It is also software reloadable and can be customized for specific purposes. The registers are accessed over I2C using the "tshwctl" utility in the ts4900-utils (<https://github.com/embeddedarm/ts4900-utils>) repository. The DIO can be accessed using the sysfs GPIOs 224 to 288 using the "tsgpio" driver. See the #GPIO section for more information on the recommended GPIO access.

```
Usage: tshwctl [OPTIONS] ...
Technologic Systems i.mx6 FPGA Utility
-m, --addr <address>  Sets up the address for a peek/poke
-v, --poke <value>    Writes the value to the specified address
-t, --peek            Reads from the specified address
-i, --mode <8n1>     Used with -a, sets mode like '8n1', '7e2', etc
-x, --baud <speed>   Used with -a, sets baud rate for auto485
-a, --autotxen <uart> Enables autotxen for supported CPU UARTs
                     Uses baud/mode if set or reads the current
                     configuration of that uart
-c, --dump            Prints out the crossbar configuration
-g, --get             Print crossbar for use in eval
-s, --set            Read environment for crossbar changes
-q, --showall        Print all possible FPGA inputs and outputs.
-h, --help           This message
```

Addr	Bits	Function
00	7:2	TTYMXC2_RXD Crossbar
	1	Reserved
	0	TTYMXC2_RXD Output Enable
01	7:2	TTYMXC4_RXD Crossbar
	1	Reserved
	0	TTYMXC4_RXD Output Enable
02	7:2	TTYMXC2_RTS Crossbar
	1	TTYMXC2_RTS Data
	0	TTYMXC2_RTS Output Enable
03	7:2	TTYMXC3_RXD Crossbar
	1	Reserved
	0	TTYMXC3_RXD Output Enable
04	7:2	TTYMXC1_CTS Crossbar
	1	Reserved
	0	TTYMXC1_CTS Output Enable
05	7:2	TTYMXC2_CTS Crossbar
	1	TTYMXC2_CTS Output Data
	0	TTYMXC2_CTS Output Enable
06	7:2	MB_TXD Crossbar
	1	Reserved
	0	MB_TXD Output Enable
07	7:2	MB_TX_EN_485 Crossbar
	1	Reserved
	0	MB_TX_EN_485 Output Enable
08	7:2	STC_TXD_485 Crossbar
	1	Reserved
	0	STC_TXD_485 Output Enable
09	7:2	STC_TX_EN_485 Crossbar
	1	Reserved
	0	STC_TX_EN_485 Output Enable
10	7:2	TXD_232_COM Crossbar
	1	Reserved
	0	TXD_232_COM Output Enable
11	7:2	RTS_232_COM Crossbar
	1	Reserved
	0	RTS_232_COM Output Enable
12	7:2	HD1_TXD Crossbar
	1	HD1_TXD Data
	0	HD1_TXD Output Enable
13	7:2	Reserved
	1	BT_EN Data
	0	BT_EN Output Enable
14	7:2	Reserved
	1	WL_EN Data
	0	WL_EN Output Enable
15	7:3	Reserved
	2	BT_RTS Input Data
	1:0	Reserved
16	7:2	BT_CTS Crossbar

	1	BT_CTS Data
	0	BT_CTS Output Enable
17	7:2	BT_RXD Crossbar
	1:0	Reserved
18	7:2	TTYMXC1_RXD Crossbar
	1:0	Reserved
19	7:2	HD1_DIO_1 Crossbar
	1	HD1_DIO_1 Data
	0	HD1_DIO_1 Output Enable
20	7:2	HD1_DIO_2 Crossbar
	1	HD1_DIO_2 Data
	0	HD1_DIO_2 Output Enable
21	7:2	HD1_DIO_3 Crossbar
	1	HD1_DIO_3 Data
	0	HD1_DIO_3 Output Enable
22	7:2	HD1_DIO_4 Crossbar
	1	HD1_DIO_4 Data
	0	HD1_DIO_4 Output Enable
23	7:2	HD1_DIO_5 Crossbar
	1	HD1_DIO_5 Data
	0	HD1_DIO_5 Output Enable
24	7:2	HD1_DIO_6 Crossbar
	1	HD1_DIO_6 Data
	0	HD1_DIO_6 Output Enable
25	7:2	EN_OUT_1 Crossbar
	1	EN_OUT_1 Data
	0	EN_OUT_1 Output Enable
26	7:2	EN_OUT_2 Crossbar
	1	EN_OUT_2 Data
	0	EN_OUT_2 Output Enable
27	7:2	FPGA_IRQ_1 Crossbar
	1	Input Data
	0	Reserved
28	7:2	STC_TXD_232 Crossbar
	1:0	Reserved
29	7:2	Reserved
	1	push_sw reset ^[1]
	0	Reserved
30	7:2	Reserved
	1	Reboot (on 1) ^[2]
	0	Reserved
31	7:2	Reserved
	1	Push SW Input Data
	0	Reserved
32	7:0	RS485_CNT0 [23:16]
33	7:0	RS485_CNT0 [15:8]
34	7:0	RS485_CNT0 [7:0]
35	7:0	RS485_CNT1 [23:16]
36	7:0	RS485_CNT1 [15:8]
37	7:0	RS485_CNT1 [7:0]

38	7:0	RS485_CNT2 [23:16]
39	7:0	RS485_CNT2 [15:8]
40	7:0	RS485_CNT2 [7:0]
41	7:0	RS485_CNT3 [23:16]
42	7:0	RS485_CNT3 [15:8]
43	7:0	RS485_CNT3 [7:0]
44	7:2	TTYMAX0_RXD Crossbar
	1	Reserved
	0	TTYMAX0_RXD Output Enable
45	7:2	TTYMAX1_RXD Crossbar
	1	Reserved
	0	TTYMAX1_RXD Output Enable
46	7:2	TTYMAX2_RXD Crossbar
	1	Reserved
	0	TTYMAX2_RXD Output Enable
51	7:4	FPGA Revision
	3	R39 Option Resistor (1 = not present)
	2	R34 Option Resistor (1 = not present)
	1	R36 Option Resistor (1 = not present)
	0	R37 Option Resistor (1 = not present)
53	7:2	TTYMAX0_CTS Crossbar
	1	Reserved
	0	TTYMAX0_CTS Output Enable
54	7:2	TTYMAX1_CTS Crossbar
	1	Reserved
	0	TTYMAX1_CTS Output Enable
55	7:2	TTYMAX2_CTS Crossbar
	1	Reserved
	0	TTYMAX2_CTS Output Enable
56	7:6	DIO1 and DIO2 input data.
	5:0	HD1_DIO input data
57	7:2	Reserved
	1	LCD_D10
	0	CN_99_BOOT_SEL Input Data
58	7:2	HD1_SPI_CLK Crossbar
	1	HD1_SPI_CLK Data
	0	HD1_SPI_CLK Output Enable
59	7:2	HD1_SPI_MOSI Crossbar
	1	HD1_SPI_MOSI Data
	0	HD1_SPI_MOSI Output Enable
60	7:2	HD1_SPI_MISO Crossbar
	1	HD1_SPI_MISO Data
	0	HD1_SPI_MISO Output Enable
61	7:2	Reserved
	1	1 = Always pass through SPI rather than on OFF_BD_CS# assert only
	0	Reserved

1. ↑ If this is set to 1, then when SW1 is pressed a hardware reboot will happen
2. ↑ This power cycles all rails rather than a software reboot

12.8.1 FPGA Crossbar

The FPGA crossbar allows almost any of the FPGA pins to be rerouted. All the FPGA addresses that have a crossbar mux register can be written with these output values.

Crossbar Value	Selected Function
0	Do not change
1	BT_RTS
2	BT_TXD
3	TTYMXC4_TXD
4	TTYMXC2_TXD
5	TTYMXC2_RTS
6	TTYMXC1_RTS
7	TTYMXC2_CTS
8	MB_RXD_485
9	STC_RXD_485_3V
10	RXD_232_COM
11	CTS_232_COM
12	STC_RXD
13	HD1_RXD
14	TTYMXC3_TXD
15	TTYMXC1_TXD
16	TTYMAX0_TXD
17	TTYMAX0_TXEN
18	TTYMAX0_RTS
19	TTYMAX1_TXD
20	TTYMAX1_TXEN
21	TTYMAX1_RTS
22	TTYMAX2_TXD
23	TTYMAX2_TXEN
24	TTYMAX2_RTS
25	TTYMXC1_TXEN
26	TTYMXC3_TXEN
27	CLK_12MHZ
28	CLK_14MHZ
29	FPGA_24MHZ_CLK
30	CLK_28MHZ
31	GPIO
32	HD1_DIO_1
33	HD1_DIO_2
34	HD1_DIO_3
35	HD1_DIO_4
36	HD1_DIO_5
37	HD1_DIO_6
38	DIO_1_IN
39	DIO_2_IN
40	LCD_D10
41	PUSH_SW_CPU
42	HD1_SPI_CLK
43	HD1_SPI_MOSI
44	HD1_SPI_MISO

For example, we can remap three ttyMAX ports to the HD1 GPIO.

Pin	Function
HD1_DIO_1	ttyMAX0 txd
HD1_DIO_2	ttyMAX0 rxd
HD1_DIO_3	ttyMAX1 txd
HD1_DIO_4	ttyMAX1 rxd
HD1_DIO_5	ttyMAX2 txd
HD1_DIO_6	ttyMAX2 rxd

```
tshwctl --dump
```

This will return the mapping of all of the pins as they are currently set. These are the relevant pins:

```
FPGA Pad (DIR) (VAL) FPGA Output
MB_TXD ( in) ( 0) TTYMAX1_TXD
STC_TXD_485 ( in) ( 0) TTYMAX0_TXD
RTS_232_COM ( in) ( 0) TTYMAX2_TXD
HD1_DIO_1 ( in) ( 0) GPIO
HD1_DIO_2 ( in) ( 0) GPIO
HD1_DIO_3 ( in) ( 0) GPIO
HD1_DIO_4 ( in) ( 0) GPIO
HD1_DIO_5 ( in) ( 0) GPIO
HD1_DIO_6 ( in) ( 0) GPIO
TTYMAX0_RXD ( in) ( 0) STC_RXD_485_3V
TTYMAX1_RXD ( in) ( 0) MB_RXD_485
TTYMAX2_RXD ( in) ( 0) CTS_232_COM
...
```

The tshwctl tool uses the bash environment to set/get pin status. To remap these pins:

```
eval $(tshwctl --get)
export HD1_DIO_1=TTYMAX0_TXD
export HD1_DIO_3=TTYMAX1_TXD
export HD1_DIO_5=TTYMAX2_TXD
export TTYMAX0_RXD=HD1_DIO_2
export TTYMAX1_RXD=HD1_DIO_4
export TTYMAX2_RXD=HD1_DIO_6

# These last 3 aren't required, but this will disable ttyMAX pins on
# their default locations. Without this, writes to /dev/ttyMAX0
# would go to both STC_TXD_485 and to HD1_DIO_1.
export MB_TXD=GPIO
export STC_TXD_485=GPIO
export RTS_232_COM=GPIO

# This will read the environment and look for the PAD names
# for any changes and apply them.
tshwctl --set
```

12.9 GPIO

The i.MX6 GPIO are available using the kernel's sysfs. See the kernel's documentation here (<https://www.kernel.org/doc/Documentation/gpio/sysfs.txt>) for more detail. This interface provides a set of files and directories for interacting with GPIO. This allows GPIO to be accessed from any language that can write files. For example to toggle CN1_89/EIM_A22 the kernel maps this to GPIO 48. See the table below for the full IO listing.

Note: It is possible to use GPIO registers as documented in the CPU reference manual to control GPIO. If this is needed keep in mind the kernel may attempt to access the same GPIO banks for various drivers. Be aware of the other IO in the same bank or use a read/modify/write.

To interact with this pin, first export it to userspace:

```
echo "48" > /sys/class/gpio/export
```

If you receive a permission denied on a pin that means it is claimed by another kernel driver. If it succeeds you will have a `/sys/class/gpio/gpio48/` directory. The relevant files in this directory are:

```
direction - "out" or "in"
value - write "1" or "0", or read "1" or "0" if direction is in
edge - write with "rising", "falling", or "none"
```

```
# Set GPIO 48 high
echo "out" > /sys/class/gpio/gpio48/direction
echo "1" > /sys/class/gpio/gpio48/value
# Set GPIO 48 Low
echo "0" > /sys/class/gpio/gpio48/value
# Read the value of GPIO 48
echo "in" > /sys/class/gpio/gpio48/direction
cat /sys/class/gpio/gpio48/value
```

As an output, the in can be written to 0 for low (GND), or 1 for high (3.3V). As an input the GPIO will have a 100k pullup. The GPIO pins off of the i.MX6 processor support an absolute maximum of -0.5 to 3.6V. It possible to use any processor GPIO as an interrupt. This is done by writing the edge value, and using select() or poll() on the value file for changes. See the #Interrupts section for more details.

The GPIO numbers in the table below are relevant to how the Linux references these numbers. The CPU documentation refers to bank and IO while Linux flattens this out to one number space.

Pins #224 and above are from the perspective of the FPGA rather than the CPU. For example, toggling the IO #224 is ttymxc2_rxd which does not toggle the cpu's uart pin, but the FPGA IO directed at that pin in the CPU. Many of these UART pins are not set as GPIO by default, and the FPGA includes its own crossbar. The UART pins will be mapped to cpu or fpga uarts.

Pad Name ^[1]	GPIO Number	Common Functions ^[2]	Location
SD4_DAT3	43	USB HUB Reset#	Onboard
DISP0_DAT23	145	SEL_DC_USB#	Onboard
EIM_A16	54	EN_USB_5V	Onboard
EIM_D27	91	Green LED	Onboard
GPIO_2	2	Red LED	Onboard
GPIO_9	9	Yellow LED	Onboard
DISP0_DAT4	121	Blue LED	Onboard
CSI0_DATA_EN	148	FPGA_IRQ_0 (FPGA UART irq)	Onboard
GPIO_4	4	FPGA_IRQ_1 (unused)	Onboard
DISP0_DAT14	136	JTAG_FPGA_TMS	Onboard
DISP0_DAT17	139	JTAG_FPGA_TCK	Onboard
DISP0_DAT18	140	JTAG_FPGA_TDO	Onboard
DISP0_DAT22	144	JTAG_FPGA_TDI	Onboard
DISP0_DAT20	142	Gyro IRQ	Onboard
EIM_LBA	59	GPIO	HD2 pin 3
EIM_OE	57	Modbus fault	Onboard
EIM_RW	58	SD Boot Jumper	Onboard
EIM_A19	51	EN_MODBUS_24V#	Onboard
DISP0_DAT5	122	EN_MODBUS_3V#	Onboard
EIM_D23	87	EN_RTC_PWR	Onboard
DISP0_DAT0	117	EN_CAN_1#	Onboard
EIM_BCLK	191	EN_CAN_2#	Onboard
DISP0_DAT7	124	GPIO	HD1 pin 7
DISP0_DAT9	126	GPIO	HD1 pin 21
DISP0_DAT10	127	GPIO	HD1 pin 9
DISP0_DAT11	133	GPIO	HD1 pin 14
EIM_CS0	55	GPIO	HD2 pin 5
EIM_A24	132	GPIO	HD2 pin 12
EIM_WAIT	128	GPIO	HD2 pin 11
EIM_EB1	61	GPIO	HD2 pin 10
EIM_DA0	64	GPIO	HD2 pin 2
EIM_DA1	65	GPIO	HD2 pin 4
EIM_DA2	66	GPIO	HD2 pin 6
EIM_DA3	67	GPIO	HD2 pin 8
EIM_DA4	68	GPIO	HD2 pin 7
EIM_DA5	69	GPIO	HD2 pin 9
EIM_DA6	70	GPIO	HD2 pin 13
EIM_DA7	71	GPIO	HD2 pin 15
EIM_DA8	72	GPIO	HD2 pin 16
EIM_DA9	73	GPIO	HD2 pin 17
EIM_DA10	74	GPIO	HD2 pin 14
EIM_DA11	75	GPIO	HD2 pin 24
EIM_DA12	76	GPIO	HD2 pin 21
EIM_DA13	77	GPIO	HD2 pin 19
EIM_DA14	78	GPIO	HD2 pin 20
EIM_DA15	79	GPIO	HD2 pin 18
SD4_DAT5	45	TTYMXC1_RTS	FPGA Crossbar
SD4_DAT6	46	TTYMXC1_CTS	FPGA Crossbar

1. ↑ The pad name does not often correspond with the functionality of the IO we use, but can be used to reference the pad in the CPU manual.
2. ↑ This does not contain all of the functions possible for a pin, but the common functions as they are used on our off the shelf baseboards. Consult the i.MX6 CPU Reference manual, and FPGA crossbar section for a complete list.

12.9.1 FPGA GPIO

The FPGA is used as a GPIO expander, and a crossbar. In most cases the FPGA IO are usable for low speed IO. The crossbar allows passing through some spare CPU GPIO which are interruptible.

Note: If any IO are used from the kernel, keep in mind these IO cannot be called from an interrupt context. These IO "can sleep". Instead of `gpio_set_value` use `gpio_set_value_cansleep()`.

Some pins will need to be changed into GPIO before they can be used. For example, to toggle HD1 pin 12 (HD1_TXD):

```
tshwctl --dump
```

```
root@ts-imx6:~# tshwctl --dump
FPGA Pad (DIR) (VAL) FPGA Output
TTYMXC2_RXD ( in) ( 0) HD1_RXD
TTYMXC4_RXD ( in) ( 0) STC_RXD
TTYMXC2_CTS ( in) ( 0) GPIO
TTYMXC3_RXD ( in) ( 0) RXD_232_COM
TTYMXC1_CTS ( in) ( 0) BT_RTS
TTYMXC2_RTS ( in) ( 1) GPIO
MB_TXD ( in) ( 0) TTYMAX1_TXD
MB_TX_EN_485 ( in) ( 0) TTYMAX1_TXEN
STC_TXD_485 ( in) ( 0) TTYMAX0_TXD
STC_TX_EN_485 ( in) ( 0) TTYMAX0_TXEN
TXD_232_COM ( in) ( 0) TTYMXC3_TXD
RTS_232_COM ( in) ( 0) TTYMAX2_TXD
HD1_TXD ( in) ( 0) TTYMXC2_TXD
BT_CTS ( in) ( 1) TTYMXC1_RTS
...
```

In this case HD1_TXD is the signal we want to toggle. The HD1_TXD signal is passing through TTYMXC2_TXD. It is possible to toggle `ttymxc2_tx` from the CPU as a GPIO, but the CPU IOMUXC would first need to be configured. On the CPU IOMUXC this is a UART, not a GPIO by default. On the FPGA as well this is configured to pass through the CPU pin, but it can be configured to be a GPIO:

```
export HD1_TXD=GPIO
tshwctl --set
```

Now running "tshwctl --dump" will show this HD1_TXD signal is now a GPIO. Refer to the below table to see the FPGA pin to toggle. In this case, 236.

```
echo 236 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio236/direction
echo low > /sys/class/gpio/gpio236/direction
```

Pad Name ^[1]	GPIO Number	Default Function	Location
TTYMXC2_RXD	224	ttymxc2 rxd	HD1 pin 12
TTYMXC4_RXD	225	ttymxc4 rxd	P1-B pin 16
TTYMXC2_RTS	226	NC	NC
TTYMXC3_RXD	227	ttymxc3 rxd	COM Header pin 2
TTYMXC1_CTS	228	ttymxc1 cts	Onboard (Bluetooth RTS)
TTYMXC2_CTS	229	NC	NC
MB_TXD	230	ttyMAX1 txd	Modbus RJ45 Data pins 4/5 +/-
MB_TX_EN_485	231	ttyMAX1 txen	Onboard
STC_TXD_485	232	ttyMAX0 TXD	P1-A Terminal Block pin 2, COM2 Header pin 1
STC_TX_EN_485	233	ttyMAX0 TXEN	Onboard ^[2]
TXD_232_COM	234	ttymxc3 TXD	COM2 Header pin 3
RTS_232_COM	235	ttyMAX2 TXD	COM2 Header pin 7
HD1_TXD	236	ttymxc2 RXD	HD1 pin 10
BT_EN	237	GPIO	Onboard
WL_EN	238	GPIO	Onboard
BT_RTS	239	ttymxc1 CTS	Onboard
BT_CTS	240	ttymxc1 RTS	Onboard
BT_RXD	241	ttymxc1 TXD	Onboard
TTYMXC1_RXD	242	ttymxc1 TXD	Onboard
HD1_DIO_1	243	GPIO	HD1 pin 8
HD1_DIO_2	244	GPIO	HD1 pin 6
HD1_DIO_3	245	GPIO	HD1 pin 4
HD1_DIO_4	246	GPIO	HD1 pin 2
HD1_DIO_5	247	GPIO	HD1 pin 24
HD1_DIO_6	248	GPIO	HD1 pin 22
EN_OUT_1	249	GPIO	Onboard/Terminal Block P1-B pin 5
EN_OUT_2	250	GPIO	Onboard/Terminal Block P1-B pin 6
STC_TXD_232	252	ttymxc4 TXD	P1-B Terminal Block pin 7
FPGA Register	253	1 = Reboot on push_sw	Register
FPGA Register	254	1 = Reboot	Register
TTYMAX0_RXD	268	ttyMAX0 RXD	P1-A Terminal Block pin 3, COM2 Header pin 6
TTYMAX1_RXD	269	ttyMAX1 RXD	RJ45 2W-Modbus pin 5
TTYMAX2_RXD	270	ttyMAX2 RXD	COM2 Header pin 8
HD1_SPI_CLK	282	#SPI, GPIO	HD1 pin 17
HD1_SPI_MOSI	283	#SPI, GPIO	HD1 pin 20
HD1_SPI_MISO	284	#SPI, GPIO	HD1 pin 18

- ↑ The pad name rarely corresponds with the functionality of the IO we use. This name can be used to reference the pad in the CPU manual.
- ↑ This pin is set up to automatically toggle with TX data in the FPGA. You do not need to manually toggle this to transmit/recieve.

12.10 Interrupts

The i.MX6 CPU GPIO are also able to function as interrupts on rising and falling edges. This is accessible from the kernel as well as userspace. Userspace IRQs are exposed through the sysfs gpio mechanism. This example will trigger on a falling edge for GPIO 48:

```
echo "48" > /sys/class/gpio/export
echo "in" > /sys/class/gpio/gpio48/direction
echo "falling" > /sys/class/gpio/gpio48/edge
```

From here, you would need to use a lower level language to poll() or select() on /sys/class/gpio/gpio48/value.

```

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

int main(int argc, char **argv)
{
    char gpio_irq[64];
    int ret, irqfd = 0, i = 0;
    fd_set fds;
    FD_ZERO(&fds);
    int buf;

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

    snprintf(gpio_irq, sizeof(gpio_irq), "/sys/class/gpio/gpio%d/value", atoi(argv[1]));
    irqfd = open(gpio_irq, O_RDONLY, S_IREAD);

    if(irqfd == -1) {
        printf("Could not open IRQ %s\n", argv[1]);
        printf("Make sure the GPIO is already exported", argv[1]);
        return 1;
    }

    // Read first since there is always an initial status
    ret = read(irqfd, &buf, sizeof(buf));

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

        if(FD_ISSET(irqfd, &fds))
        {
            FD_CLR(irqfd, &fds); //Remove the filedes from set
            printf("IRQ detected %d\n", i);
            fflush(stdout);
            i++;

            /* The return value includes the actual GPIO register value */
            read(irqfd, &buf, sizeof(buf));
            lseek(irqfd, 0, SEEK_SET);
        }

        //Sleep, or do any other processing here
        usleep(100000);
    }

    return 0;
}

```

This example can be run as `./irqtest 48` which will echo every time the pin changes, but will otherwise take no cpu time.

12.11 LEDs

The kernel provides access to control the LEDs using the sysfs:

```

# Set Red Led on
echo 1 > /sys/class/leds/red-led/brightness
# Set Red Led off
echo 0 > /sys/class/leds/red-led/brightness

# Set Green Led on
echo 1 > /sys/class/leds/green-led/brightness
# Set Green Led off
echo 0 > /sys/class/leds/green-led/brightness

```

The kernel provides various triggers that can be useful for debugging purposes. The trigger for a given LED is in its directory:

```
echo "heartbeat" > /sys/class/leds/red-led/trigger
```

Trigger value	LED toggles on
none	Default, no action
mmc0	MicroSD card activity
mmc1	eMMC activity
mmc2	WIFI SDIO activity
timer	2hz blink
oneshot	Blinks after delay. ^[1]
heartbeat	Similar to timer, but varies the period based on system load
backlight	Toggles on FB_BLANK
gpio	Toggle based on a specified gpio. ^[2]
cpu0	Blink on CPU core 0 activity
cpu1	Blink on CPU core 1 activity
cpu2	Blink on CPU core 2 activity
cpu3	Blink on CPU core 3 activity
default-on	Only turns on by default. Only useful for device tree.
transient	Specify on/off with time to turn off. ^[3]
flash/torch	Toggle on Camera activation. Not currently used.

- ↑ See the Kernel documentation (<https://github.com/embeddedarm/linux-3.10.17-imx6/blob/master/Documentation/leds/ledtrig-oneshot.txt>) for more details
- ↑ When this trigger is set, a "gpio" file appears in the same directory which can be used to specify what GPIO to follow when it blinks
- ↑ See the Kernel documentation (<https://github.com/embeddedarm/linux-3.10.17-imx6/blob/master/Documentation/leds/ledtrig-transient.txt>) for more details

12.12 MicroSD Card Interface

The i.MX6 SDHCI driver supports MicroSD (0-2GB), MicroSDHC (4-32GB), and MicroSDXC(64GB-2TB). The cards available on our website on average support up to 16MB/s read, and 22MB/s write using this interface. The linux driver provides access to this socket at /dev/mmcblk1 as a standard Linux block device.

See chapter 67 of the IMX6 reference manual for more information on this mmc controller.

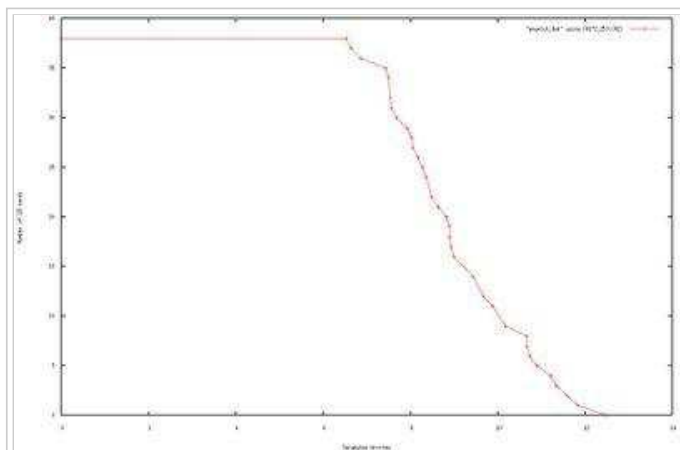
We have performed compatibility testing on the Sandisk MicroSD cards we provide. We do not suggest switching brands/models without your own qualification testing. While SD cards specifications are standardized, in practice cards behave very differently. We do not recommend ATP or Transcend MicroSD cards due to known compatibility issues.

Our testing has shown that on average microSD cards will last between 6-12TB. After this cards can begin to experience corruption, or stop being recognized by the host pc. This may be enough storage for many applications to write for years without problems. For more reliable storage consider using the eMMC. Our endurance testing showed a write lifetime on average of about 123TiB.

MicroSD cards should not have power removed during a write or they will have disk corruption. Keep the filesystem mounted read only if this is a possibility. It is not always possible for fsck to recover from the types of failures that will be seen with SD power loss. Consider using the eMMC for storage instead which is far more resilient to power loss.

12.13 NVRAM

The RTC includes 128 bytes of NVRAM which can be used for custom applications. There is a utility, nvramctl which can be used to read/write the NVRAM.



This graph shows our SD write endurance test for 40x TS-7553 boards. These boards are running a doublestore stress test on 4GB Sandisk MicroSD cards. A failure is marked on the graph for a card once a single bit of corruption is found.

ts4900-utils github (<https://github.com/embeddedarm/ts4900-utils/>) .

The utility reads/writes a byte at a time, and returns the value in hex.

```
nvractl --addr 10 --set 0x40
nvractl --addr 10 --get
# Returns "nvractl=0x40".
# This can also be used with eval
eval $(nvractl --addr 10 --get)
echo $nvractl
# Returns "0x40"
```

The NVRAM code can be included in your application by using these two files:

- nvractl.c (<https://github.com/embeddedarm/ts4900-utils/blob/master/src/nvractl.c>)
- nvractl.h (<https://github.com/embeddedarm/ts4900-utils/blob/master/src/nvractl.h>)

12.14 Onboard SPI Flash

This board includes 8MiB of SPI flash using a Micron N25Q064A13ESE40F. The CPU uses this for the initial boot to load u-boot, as well as the u-boot environment. In Linux this is accessed with the /dev/mtdblock devices.

Bytes	Size	Description
0-0x3ff	1KB	Unused
0x400-0xffff	0.999MiB	U-boot
0x100000-0x101fff	8KiB	U-boot environment #1
0x102000-0x17ffff	504KiB	Unused
0x180000-0x181fff	8KiB	U-boot environment #2
0x182000-0x1ffff	504KiB	Unused
0x200000-0x700000	5MiB	Unused

12.15 RTC

We include the Intersil ISL12020 RTC onboard. This provides a long RTC battery life, as well as a built in temperature sensor to provide +/- 5ppm across -40 to 85C. This is /dev/rtc0 in our images, and is accessed using the standard hwclock (<http://linux.die.net/man/8/hwclock>) command.

12.16 USB

12.16.1 USB OTG

This SBC includes support to act as a USB peripheral to another system. Remove the "CON EN" jumper to disable the onboard usb serial, and connect the P1 header to the CPU's OTG port. This port is strapped to only act as a USB device. Several devices are compiled into the default kernel. Other devices can be compiled into the kernel by following the section here.

USB Serial

```
modprobe g_serial use_acm=1
```

This will create a /dev/ttyGS0. See the kernel documentation for more information:

- USB Gadget Serial documentation (https://www.kernel.org/doc/Documentation/usb/gadget_serial.txt)
- Windows CDC-ACM INF file (<https://github.com/embeddedarm/linux-3.10.17-imx6/blob/master/Documentation/usb/linux-cdc-acm.inf>)

USB Ethernet

```
modprobe g_ether
```

This provides a usb0 network interface. This driver simulates an Ethernet network connection between the host pc and the i.MX6.

- Windows driver inf (<https://github.com/embeddedarm/linux-3.10.17-imx6/blob/master/Documentation/usb/linux.inf>)

12.16.2 USB Host

The i.MX6 provides 1 USB Host with supporting USB 2.0 (480Mbit/s). The TS-7970 includes a USB Hub expanding this to 4 USB host ports.

Typically USB is interfaced with by using standard Linux drivers, but low level USB communication is possible using libusb (<http://www.libusb.org/>).

The TS-7970 USB 5V rail can be toggled on/off through a GPIO. This can be used to save power, or to reset USB devices that get stuck in a bad state.

```
# Power disabled
echo 1 > /sys/class/leds/en-usb-5v/brightness
sleep 2 # Let any devices reset
# Enable power
echo 0 > /sys/class/leds/en-usb-5v/brightness
```

Note: The USB OTG which can act as a host does not always use the same controllable 5V supply. Refer to the schematic's EN_USB_5V/USB_5V for more information on this control.

The TS-7970 also has a USB port on the HD1 header (11 data-, 13 data+). This is used for some off the shelf daughter cards like the DC767-MT which supports Multitech cellular modems. To use this USB port you must toggle a GPIO which switches a USB mux onboard. This switches one of the USB hosts from the top port next to the Ethernet connector to HD1.

```
# Set USB4 to HD1
echo 1 > /sys/class/leds/sel_dc_usb/brightness
# Set USB4 to J1 (default)
echo 0 > /sys/class/leds/sel_dc_usb/brightness
```

12.17 SATA

The i.MX6 Quad and Dual include integrated SATA II support. This interface has been tested to provide 72MiB/s write, and 75MiB/s read through block access. In linux this is accessed through the /dev/sda device:

```
[ 1.768036] ata1: SATA link up 3.0 Gbps (SStatus 123 SControl 300)
[ 1.785377] ata1.00: ATA-8: MKNSSDAT30GB-DX, 507ABBF0, max UDMA/133
[ 1.791716] ata1.00: 58626288 sectors, multi 16: LBA48 NCQ (depth 31/32)
[ 1.805380] ata1.00: configured for UDMA/133
[ 1.810320] scsi 0:0:0:0: Direct-Access ATA MKNSSDAT30GB-DX 507A PQ: 0 ANSI: 5
[ 1.819459] sd 0:0:0:0: [sda] 58626288 512-byte logical blocks: (30.0 GB/27.9 GiB)
[ 1.827427] sd 0:0:0:0: [sda] Write Protect is off
[ 1.832812] sd 0:0:0:0: [sda] Write cache: enabled, read cache: enabled, doesn't support DPO or FUA
[ 1.843621] sda: sda1
[ 1.847381] sd 0:0:0:0: [sda] Attached SCSI dis
```

To use the SATA for booting, press the SW1 button on startup and enter the u-boot command prompt. Change the default bootcmd to instead load from sata by running:

```
env set bootcmd 'run sataboot';
env save;
```

On startup now the SD boot jumper will be ignored, and the board will boot straight to sata.

12.18 Silabs Microcontroller

The onboard silabs microcontroller includes 3 primary functions:

- ADC channels
- Sleep Mode
- USB Console

The USB console passes through the CPU's ttyMXC0 port using a CP201x driver. This is present on most Linux distributions. On Windows this is available with a WHQL signed driver from Silabs.

The Silabs exists at 0x10 on the i2c bus 0 using 8-bit address and data. It can be read for up to 32 bytes to get the ADC values, and Silabs revision. Our example code "tsmicroctl -i" includes reading all the ADCs in millivolts. This also includes a variable for the revision. For example:

```
# tsmicroctl -i
VDD_ARM_CAP=1026
VDD_HIGH_CAP=2603
VDD_SOC_CAP=1239
VDD_ARM=1451
SILAB_P10=0x0
SILAB_P11=0x0
SILAB_P12=0x0
VIN=4779
V5_A=5189
V3P1=3230
DDR_1P5V=1559
V1P8=1904
V1P2=1259
RAM_VREF=778
V3P3=3522
SILABREV=1
```

In u-boot you can use the "tsmicroctl" command with no arguments to read the same values.

The sleep mode is accessible in u-boot with "tsmicroctl <seconds>", and in Linux with "tsmicroctl -s <seconds>". This will power off everything on the board except the silabs microcontroller. The blue LED will blink while it is in this mode.

The silabs sample all ADC channels in a scale of 0-2.5V. The schematic shows the voltage dividers to bring the higher voltages it samples into this range.

Silabs Read Registers

Register	Description
0	VDD_ARM_CAP MSB
1	VDD_ARM_CAP LSB
2	VDD_HIGH_CAP MSB
3	VDD_HIGH_CAP LSB
4	VDD_SOC_CAP MSB
5	VDD_SOC_CAP LSB
6	VDD_ARM MSB
7	VDD_ARM LSB
8	SILAB_P10 MSB
9	SILAB_P10 LSB
10	SILAB_P11 MSB
11	SILAB_P11 LSB
12	SILAB_P12 MSB
13	SILAB_P12 LSB
14	VIN MSB
15	VIN LSB
16	V5_A MSB
17	V5_A LSB
18	V3P1 MSB
19	V3P1 LSB
20	DDR_1P5V MSB
21	DDR_1P5V LSB
22	V1P8 MSB
23	V1P8 LSB
24	V1P2 MSB
25	V1P2 LSB
26	RAM_VREF MSB
27	RAM_VREF LSB
28	V3P3 MSB
29	V3P3 LSB
30	Silabs Revision

12.18.1 Silabs Sleep Mode

The TS-7970 implements a very low power sleep mode using the onboard supervisory microcontroller. This allows powering off the i.MX6 CPU entirely. While in this mode the entire board will use about 26mW.

The board can be woken 3 ways:

- Timer - sleep mode requires specifying an amount of seconds to sleep (up to 16777215).
- SW1 - Pressing the button on the side of the board.
- PUSH_SW# goes low on HD1. The SW1 signal is brought to the header so connected cards can wake the TS-7970.

The sleep mode can be entered at a low level calling "tshwctl --sleep 60" to sleep for 60 seconds, but this typically should not be called directly. This would be equivalent to disconnecting power while booted which can cause data loss.

The Yocto, Debian, or Ubuntu distributions use systemd to manage shutdown. When systemd shuts down it will call all executables in /lib/systemd/system-shutdown/. Create a script silabs-sleep in this directory with these contents:

```
#!/bin/bash
tsmicroctl --sleep 60
```

And make it executable:

```
chmod a+x /lib/systemd/system-shutdown/silabs-sleep
```

Now the board will sleep immediately following a shutdown. It is safe during the sleep mode to disconnect power.

12.19 SPI

The CPU has 1 SPI controller which is accessible offboard through either specific kernel drivers, or userspace using the /dev/spidev interface. On the TS-7970 these are exposed as /dev/spidev1.1 (FPGA) and /dev/spidev1.2 (HD1) in userspace.

- Linux kernel spidev documentation (<https://github.com/embeddedarm/linux-3.10.17-imx6/blob/master/Documentation/spi/spidev>)
- spidev example code (https://github.com/embeddedarm/linux-3.10.17-imx6/blob/master/Documentation/spi/spidev_test.c)

The /dev/spidevX.Y are created where X is the controller and Y is the chip select used. See the compiling the kernel section to get a build environment up. Any GPIO can be used as another SPI chip select by modifying the device tree. For example arch/arm/boot/dts/imx6qdl-ts7970.dtsi:

```
&ecspi2 {
    fsl,spi-num-chipselects = <3>;
    cs-gpios = <&gpio5 31 0>, <&gpio7 12 0>, <&gpio5 18 0>;
    pinctrl-names = "default";
    pinctrl-0 = <&pinctrl_ecspi2>;
    status = "okay";

    serial1: max3100-1@0 {
        compatible = "max3100-ts";
        reg = <0>;
        interrupt-parent = <&gpio1>;
        interrupts = <4 2>;
        spi-max-frequency = <1000000>;
        loopback = <0>;
        crystal = <1>;
        poll-time = <100>;
        fifo-size = <16>;
    };

    spidevfpga: spi@1 {
        compatible = "spidev";
        reg = <1>;
        spi-max-frequency = <1000000>;
    };

    spidevhd1: spi@2 {
        compatible = "spidev";
        reg = <2>;
        spi-max-frequency = <1000000>;
    };
};
```

This bus is shared with the onboard fpga uarts (/dev/ttyMAX*). The spidevfpga node is intended for customized FPGA communication. The HD1 node is for general use.

12.20 TWI

The i.MX6 supports standard I2C at 100khz, or using fast mode for 400khz operation. The CPU has 2 I2C buses used on the TS-7970 /dev/i2c-0 is internal to the board and connects to the RTC and FPGA.

Address	Device
0x10	#Silabs Microcontroller
0x28-0x2f	#FPGA
0x57	#NVRAM
0x6b	Onboard PCIe Clock Generator
0x6f	#RTC

The second I2C bus (/dev/i2c-1) is brought out on HD3 pin 15 (SCL) and HD3 pin 16 (SDA). This bus has no onboard devices.

Note: It is also possible to request the kernel to bitbang additional I2C buses as needed. See an example here ([https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/619e6bf97479243e9d7b7f6b34ce0ae8558ff1fd/arch/arm/boot/dts/imx6qdl-
ts4900-2.dtsi#L196](https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/619e6bf97479243e9d7b7f6b34ce0ae8558ff1fd/arch/arm/boot/dts/imx6qdl-
ts4900-2.dtsi#L196)).

The kernel makes the I2C available at /dev/i2c-#. You can use the i2c-tools (i2cdetect, i2cgetm, i2cset), or you can write your own client ([https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/619e6bf97479243e9d7b7f6b34ce0ae8558ff1fd/Documentation/i2c/dev-interface](https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/619e6bf97479243e9d7b7f6b34ce0ae8558ff1fd/Documentation/i2c/dev-interface)).

12.21 Watchdog

The kernel provides an interface to the watchdog driver at /dev/watchdog. Refer to the kernel documentation for more information:

- watchdog-api.txt ([https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/master/Documentation/watchdog/watchdog-
api.txt](https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/master/Documentation/watchdog/watchdog-
api.txt))
- watchdog-simple.c ([https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/master/Documentation/watchdog/src/watchdog-
simple.c](https://github.com/embeddedarm/linux-3.10.17-
imx6/blob/master/Documentation/watchdog/src/watchdog-
simple.c))

12.22 WIFI

This board includes a TiWi-BLE SDIO module that uses the Texas Instruments WL1271L Transceiver. Linux provides support for this using the wl12xx driver. See the LSR site (<http://www.lsr.com/embedded-wireless-modules/wifi-plus-bluetooth-module/tiwi-ble>) for detailed product information.

Summary Features:

- IEEE 802.11 b/g/n
- 2.4GHz
- Linux drivers include support for client and AP mode
- Industrial temp, -40 to 85C
- Certifications
 - FCC Bluetooth® Grant
 - FCC WLAN Grant
 - IC
 - CE
 - SAR Testing
 - SAR Testing EU

Linux uses the "wireless-tools", "wpa-suplicant", and "hostapd" packages to support most of the functionality in this module. Refer to the distribution support for #Yocto, #Debian, or #Android for more information.

13 External Interfaces

13.1 Audio

The TS-7970 includes two 3.5mm jacks. The top port is a mono microphone input, and the bottom port is a headphone port with left and right channels. Use "alsamixer" or "amixer" to adjust the volume.

In Linux select between the HDMI and 3.5mm ports with an alsa variable. By default all audio comes out of the sgtl5000 for the 3.5mm ports.

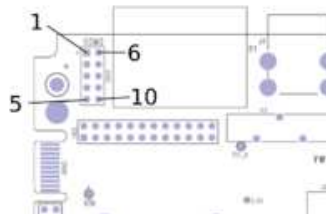
```
# List the sound cards
aplay -l

export ALSA_CARD=imxhdmisoc
espeak "this is playing from the HDMI monitor"

export ALSA_CARD=imx6qts7970sgtl
espeak "this is playing from the onboard sgtl5000"
```

13.2 COM2 Header

The COM2 header is a 2x5 0.1" pitch header with RS485, CAN, and RS232.



Pin #	Description
1	STC_485+ (/dev/ttyMAX0)
2	RS232 RXD (/dev/ttymx3)
3	RS232 TXD (/dev/ttymx3)
4	CAN_1_H (can0 interface)
5	GND
6	STC_485- (/dev/ttyMAX0)
7	RS232 TXD (/dev/ttyMAX2)
8	RS232 RXD (/dev/ttyMAX2)
9	CAN_1_L (can0 interface)
10	NC

13.3 Ethernet

The TS-7970 includes two 10/100/1000 Ethernet ports. The port with the larger connector uses the FEC MAC from the CPU with a Marvell PHY. The other smaller optional port is a PCIe Intel I210 chipset.

Under Linux the CPU ethernet is typically eth0, and depending on the distribution the second ethernet is either eth1 or enp1s0.

The CPU and I210 both receive unique sequential mac addresses. These are pulled from the Technologic Systems OUI "00:D0:69". Both chipsets support downshifting if some of the twisted pairs are missing connection.

On both Ethernet ports the right LED indicates speed. It is on for gigabit, and off for 10/100. The left LED will blink on activity.

See the #Debian_Networking for configuring the network on the default distribution.

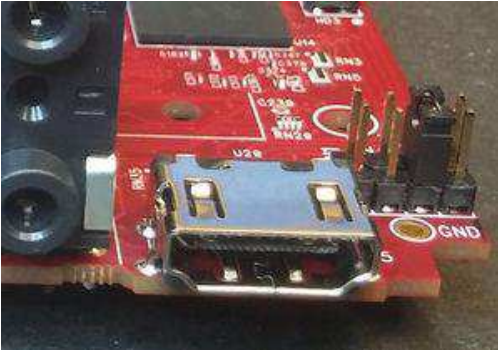
13.4 HDMI

The TS-7970 includes an HDMI 1.4 port which supports EDID for automatically configuring the video modes on your monitor, and HDMI audio. In Linux this will be /dev/fb0. The HDMI is capable of outputting up to 1080p60.

Under either distribution the mode will default to the largest and highest refresh rate compatible with both the monitor and the i.MX6. You can override this in yocto with xrandr, or under debian/ubuntu/yocto with a kernel cmdline change. Hold SW1 when power is applied and the board will stop at u-boot. Run these commands to override the EDID settings and use 1024x768M@60.

```
env set cmdline_append "console=ttymx0,115200 rootwait ro init=/sbin/init video=mxcfb0:dev=hdmi,1024x768M@60,if=RGB24"
env save
```

HDMI to VGA/DVI-A adapters are possible to use with the TS-7970, but the port protection chip strictly follows the HDMI standard. Some adapters are known to violate the standard and try to pull too much power off of the HDMI port. For an adapter to work it should accept a separate power input such as a microUSB port. The TS-7970 can still power the adapter through USB, but the HDMI header should not be used for sourcing current.



13.4.1 Rotate the video output

Under Yocto you can use xrandr to rotate the screen at any time:

```
export DISPLAY=:0
xrandr --rotate left
xrandr --rotate right
xrandr --rotate normal
xrandr --rotate inverted
```

Under Debian or Ubuntu you can rotate the screen in the Xorg.conf. Edit the file /etc/X11/xorg.conf and append this to the end:

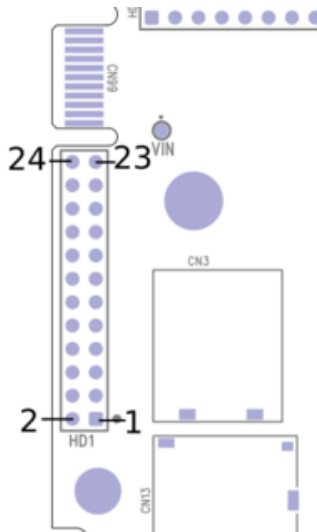
```
Section "Device"
    Identifier      "fbdev display"
    Driver          "fbdev"
    Option "Rotate" "CCW"
EndSection
```

After the display is rotated you will also need to rotate a touchscreen if this is being used. This example matches the CCW rotation, but swapaxes or the invertx/y options will need to be adjusted for other rotations.

```
Section "InputClass"
    Identifier "axis inversion"
    MatchIsTouchscreen "true"
    # swap x/y axes on the device. i.e. rotate by 90 degrees
    Option "SwapAxes" "on"
    # Invert the respective axis.
    Option "InvertX" "on"
    Option "InvertY" "off"
EndSection
```

13.5 HD1

HD1 is a 2x12 0.10" pitch header including DIO, USB, SPI, an IRQ, 3.3V, and 5V. All GPIO are 3.3V tolerant unless otherwise specified.

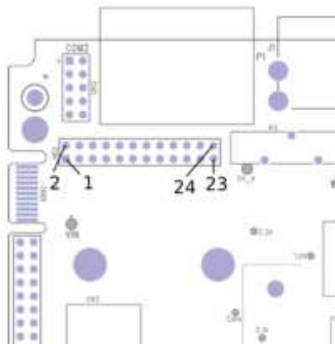


Pin #	GPIO Number	Description
1	N/A	NC
2	246	HD1_DIO_4
3	N/A	GND
4	245	HD1_DIO_3
5	N/A	GND
6	244	HD1_DIO_2
7	124	DISP0_DAT7
8	243	HD1_DIO_1
9	127	DISP0_DAT10
10	236	ttymxc2 RXD
11	N/A	USB host data-
12	224	ttymxc2 TXD
13	N/A	USB host data+
14	133	DISP0_DAT11

15	N/A	5V
16	N/A	5V
17	282	SPI_2_CLK
18	284	SPI_2_MISO
19	146	HD1_SPI_CS#
20	283	SPI_2_MOSI
21	126	HD1_IRQ
22	248	HD1_DIO_6
23	N/A	3.3V
24	247	HD1_DIO_5

13.6 HD2

HD2 is a 0.10" pitch header.

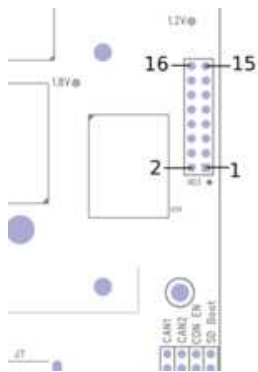


Pin #	Description
1	GND
2	EIM_DA0 (GPIO #64)
3	EIM_LBA (GPIO #59)
4	EIM_DA1 (GPIO #65)
5	EIM_CS0 (GPIO #55)
6	EIM_DA2 (GPIO #66)
7	EIM_DA4 (GPIO #68)
8	EIM_DA3 (GPIO #67)
9	EIM_DA5 (GPIO #69)
10	EIM_EB1 (GPIO #61)
11	EIM_WAIT (GPIO #128)
12	EIM_A24 (GPIO #132)
13	EIM_DA6 (GPIO #70)
14	EIM_DA10 (GPIO #74)
15	EIM_DA7 (GPIO #71)
16	EIM_DA8 (GPIO #72)
17	EIM_DA9 (GPIO #73)
18	EIM_DA15 (GPIO #79)
19	EIM_DA13 (GPIO #77)
20	EIM_DA14 (GPIO #78)
21	EIM_DA12 (GPIO #76)
22	EIM_RW (GPIO #58)
23	3.3V supply
24	EIM_DA11 (GPIO #75)

13.7 HD3

HD3 is a 2x8 0.10" pitch header including LVDS, I2C, 3.3V and 5V which can be used to connect up a third party display.

Pin #	Description
-------	-------------



1	LVDS0_TX1_N
2	LVDS0_TX1_P
3	3.3V supply
4	LVDS0_TX0_N
5	LVDS0_TX0_P
6	5V Supply
7	LVDS0_CLK_N
8	LVDS0_CLK_P
9	GND
10	LVDS0_TX2_N
11	LVDS0_TX2_P
12	GND
13	LVDS0_TX3_N
14	LVDS0_TX3_P
15	I2C 1 CLK
16	I2C 1 DAT

13.8 Mini Card Connector

The TS-7970 includes a mini card header which includes USB and power like a mini pcie header. It does not include a PCIe bus. Many peripherals do not actually need the PCIe bus and instead use the USB host which is present on this header.

This port also supports mSATA which can be used for higher capacity drives.

Note: SATA is not available on the i.MX6 solo or duallite processors.

13.9 Push Button

The push switch is accessed by reading FPGA registers:

```
tshwctl --addr 31 --peek
```

With no press bit 2 will be set so it will return "addr31=0x4". If there is a press it will be cleared, so "addr31=0x0".

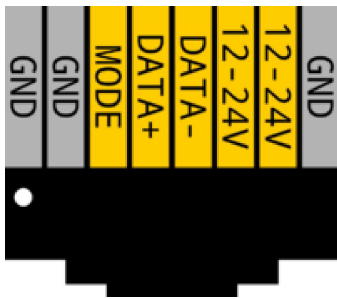
This pin is sampled in u-boot to detect if it should stop in u-boot and look for usb updates. If this interferes with your intended usage you can boot to u-boot and disable this by running:

```
env delete preboot;
env set bootdelay 1;
env save;
```

After this change u-boot will wait 1 second on every boot for the user to press ctrl+c to break into u-boot on startup.

13.10 RJ45 2W-Modbus

The 2W MODBUS ports both follow a standard pinout:



The RS485 port on pins 4 and 5 is accessed with /dev/ttyMAX1.

The MODBUS_FAULT signal (gpio 57) is used to determine if there is a dead-short on the MODBUS_POWER pins. This enables the developer to detect a line problem before turning potentially damaging power onto the line. The method of checking this is to set en_modbus_3v3, then read modbus_fault. If modbus_fault is high, then there is a problem with the cabling and en_modbus_24v should not be asserted.

```
# en_24v 51
# en_3v 122
# mb_fault 57
echo 51 > /sys/class/gpio/export
echo 122 > /sys/class/gpio/export
echo 57 > /sys/class/gpio/export

# test line with 3.3v
echo "out" > /sys/class/gpio/gpio51/direction
echo 0 > /sys/class/gpio/gpio51/value # en_mb_3v3 is active Low.

# Read mb_fault
echo "in" > /sys/class/gpio/gpio57/direction
cat /sys/class/gpio/gpio57/value
# If returns 1, do not continue.

# Switch 3V off pins 6 and 7:
echo 1 > /sys/class/gpio/gpio122/value

#Switch VIN to pins 6 and 7:
echo high > /sys/class/gpio/gpio51/direction
```

13.11 Terminal Blocks

The TS-7970 includes two removable terminal blocks (OSTTJ0811030) for power, UARTs, CAN, and other general purpose IO.



P1-A	
Pin #	Description
1	Ground
2	STC_485+ (/dev/ttyMAX0)
3	STC_485- (/dev/ttyMAX0)

P1-B	
Pin #	Description
1	Ground
2	AD_P10 (4-20mA analog input)
3	AD_P11 (4-20mA analog input)

4	STC_CAN_2_H (can1 interface)	4	AD_P12 (4-20mA analog input)
5	STC_CAN_2_L (can1 interface)	5	DIO_1 (30VDC IO)
6	Power Input (8-28VDC)	6	DIO_2 (30VDC IO)
7	Power Input (5VDC)	7	RS-232_STC_TXD (/dev/ttymx4)
8	Ground ^[1]	8	RS-232_STC_RXD (/dev/ttymx4)

- ↑ This ground should be used for the power supply since it includes a ferrite bead to help suppress noise.

The DIO_1 and DIO_2 IO can be outputs, or inputs. As inputs the digital threshold is 1.2V. To guarantee low it must be < 0.5V, or for high > 2.0V. When the IO is low the external device needs to sink up to 3.5mA. When the IO is high the external device needs to source 10 uA max. There is an internal 1.5k pullup to 5V that will bias the input high. As outputs these IO can sink up to 500mA.

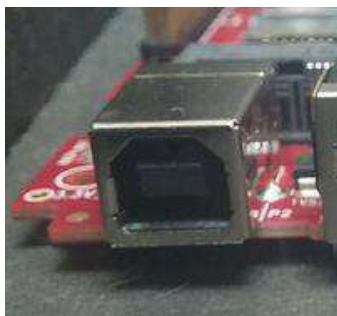
EN_DIO_1 and EN_DIO_2 outputs are controlled through FPGA DIO 249 and 250 respectively. See the #GPIO section for more information. If these pins are specified as low or in, then they are readable on FPGA reg 56 bits 7:6.

DIO_1 and DIO_2 are accessed through FPGA registers.

```
tshwctl --addr 56 --peek
# read "addr56" into bash variable
eval $(tshwctl --addr 56 --peek)
# Read bit 7 for DIO1
echo $((($addr56 >> 7))
# Read bit 6 for DIO2
echo $((($addr56 >> 6) & 0x1))
```

13.12 USB Device

The USB type B device port is connected to the onboard Silabs for USB to serial console, or to the CPU's #USB OTG. The USB functionality is picked from the "CON EN" jumper.



Note: Previous to REV C TS-7970, the USB device port can prevent the CPU from booting up if "CON_EN" is removed, and USB is plugged in before CPU power is connected. As a workaround on earlier revs you can cut the red wire from your USB cable.

13.13 USB Hosts

The TS-7970 includes 4x USB 2.0 ports. The top port on the USB Type A connector near the Ethernet connectors is brought through a MUX chip. This USB port can be toggled to switch to the Mini-PCIE connector for use with a cell modem or other peripheral.

14 Specifications

14.1 Power Specifications

The TS-7970 includes 2 methods for powering the board. There is a 5V input, and a 8-28V input. Only one of these should be provided to the board.

Input	Min voltage	Max voltage
5V input	4.75	5.25
8-28V Input	8.00	28.00

14.2 Power Consumption

The i.MX6 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 and GPU are idle. It is also possible to disable the Ethernet PHY for extra savings.

```
# Put ETH PHY in reset
echo 116 > /sys/class/gpio/export
echo high > /sys/class/gpio/gpio116/direction

# Put USB HUB in reset
echo 43 > /sys/class/gpio/export
echo low > /sys/class/gpio/gpio43/direction

# Lower backlight to 50%
echo 4 > /sys/class/backlight/backlight_local_lcd/brightness

# Disable backlight
echo 0 > /sys/class/backlight/backlight_local_lcd/brightness
```

Ethernet is not connected unless otherwise specified. Serial is disconnected during the measurement. The CPU test is 5x processes of "opnsnl speed". The GPU test is Qt5CinematicExperience in the Yocto image.

These tests are performed powering the board through 5V.

TS-7970 solo without WIFI or I210

Test	Max Watts	Average Watts
CPU 100% + GPU loaded (LCD 100%) + IO + Ethernet + HDMI	4.50 (0.90 A)	3.40 (0.68 A)
CPU 100%	2.80 (0.56 A)	2.35 (0.47 A)
CPU Idle + HDMI	2.75 (0.55 A)	2.05 (0.41 A)
CPU Idle + CPU Ethernet	2.75 (0.55 A)	2.20 (0.44 A)
CPU Idle	2.50 (0.50 A)	1.95 (0.39 A)
CPU Idle USB HUB off	2.75 (0.55 A)	1.95 (0.39 A)
CPU Idle USB HUB off, Ethernet PHY in reset	2.15 (0.43 A)	1.60 (0.32 A)
Using onboard uC to sleep CPU	0.025 (125 mA)	0.015 (3 mA)

TS-7970 quad core with WIFI and I210

Test	Max Watts	Average Watts
CPU 100% + GPU loaded (LCD 100%) + IO + Ethernet + HDMI	10.80 (2.16 A)	7.75 (1.55 A)
CPU 100%	6.15 (1.23 A)	5.40 (1.08 A)
CPU Idle + HDMI	4.55 (0.91 A)	2.90 (0.58 A)
CPU Idle + WIFI on wpa2 running iperf	6.85 (1.37 A)	3.95 (0.79 a)
CPU Idle + CPU Ethernet	5.00 (1.00 A)	3.10 (0.62 A)
CPU Idle + PCIe Ethernet	3.60 (0.72 A)	2.85 (0.57 A)
CPU Idle	4.85 (0.97 A)	2.80 (0.56 A)
CPU Idle USB HUB off	3.50 (0.70 A)	2.75 (0.55 A)
CPU Idle USB HUB off, Ethernet PHY in reset	3.30 (0.66 A)	2.40 (0.48 A)
Using onboard uC to sleep CPU	0.025 (125 mA)	0.015 (3 mA)

14.3 Temperature Specifications

The i.MX6 CPUs we provide off the shelf are either a solo industrial, or quad core extended temperature. The TS-7970 is designed using industrial components that will support -40C to 85C operation, but the CPU is rated to a max junction temperature rather than an ambient temperature. We expect the solo to work to 80C ambient while idle with a heatsink and open air circulation. To reach higher temperatures with this or other variants of this CPU some custom passive or active cooling may be required.

Model Number	Operating Min	Cooling Temp ^[1]	Passive Temp ^[2]	Critical/Max Junction Temp ^[3]
TS-7970-*S8S*	-40C	75C	85C	105C
TS-7970-*Q10S*	-20C	75C	85C	100C

1. ↑ CPU stops all throttling below this temperature
2. ↑ CPU begins throttling until the cooling temperature
3. ↑ CPU Max temperature. Linux will shut down to cool in u-boot at this temperature.

Our test data can be used to estimate the temperature rise of the CPU over the ambient temperature. These are tested without an enclosure in open air. The temp ranges show the CPU at idle at the low end, to a very high system load at the high end.

Configuration	Temp rise over ambient
Solo No Heatsink	21-27C
Solo with HS-50x53x13	18-20C
Quad No Heatsink	16-50C
Quad with HS-50x53x13	10-23C

For custom builds these are also exposed in /sys/:

```
# Passive
cat /sys/devices/virtual/thermal/thermal_zone0/trip_point_0_temp
# Critical
cat /sys/devices/virtual/thermal/thermal_zone0/trip_point_1_temp
```

The current CPU die temp can be read with:

```
cat /sys/devices/virtual/thermal/thermal_zone0/temp
```

When the CPU heats up past the cooling temp on a first boot, it will take no action. Heating up past the passive temperature the kernel will cool down the CPU by reducing clocks. This will show a kernel message:

```
[ 158.454693] System is too hot. GPU3D will work at 1/64 clock.
```

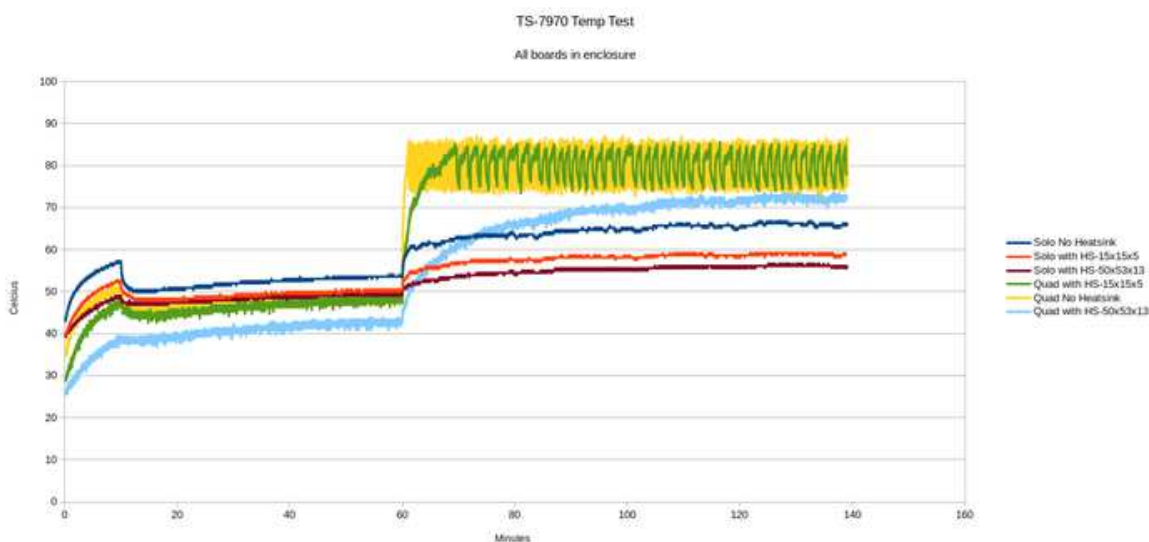
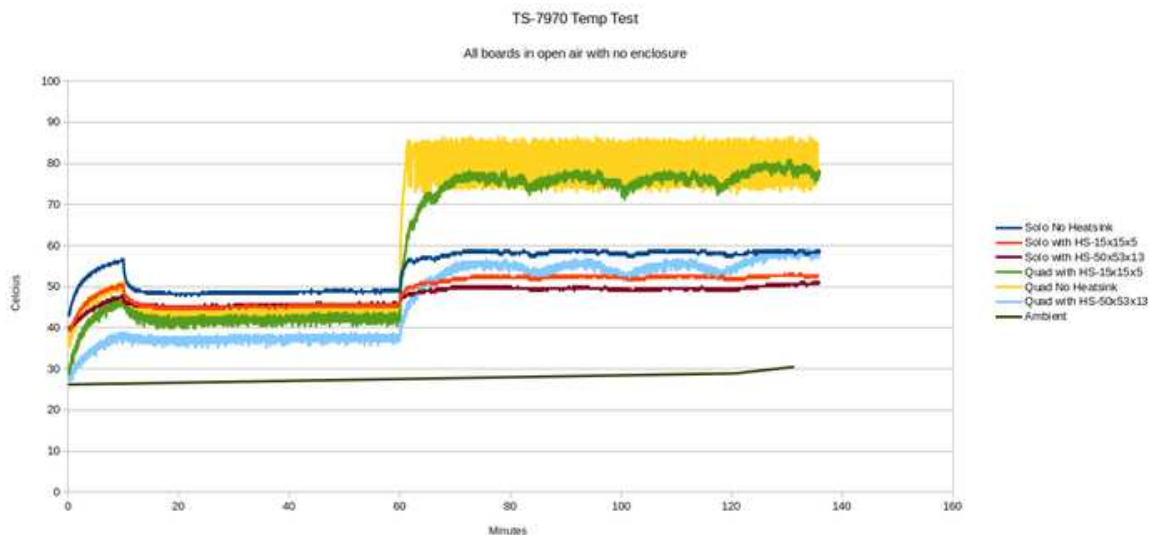
If it cools back down below the cooling temperature it will spin back up the clocks.

```
[ 394.082161] Hot alarm is canceled. GPU3D clock will return to 64/64
```

If it continues heating to the critical temperature it will overheat and reboot. Booting back up u-boot will block the boot until the temperature has been reduced to the Cooling Temp+5C. This will be shown on boot with:

```
U-Boot 2015.04-07857-g486fa69 (Jun 03 2016 - 12:04:30)
CPU: Freescale i.MX6S0LO rev1.1 at 792 MHz
CPU Temperature is 105 C, too hot to boot, waiting...
CPU Temperature is 102 C, too hot to boot, waiting...
CPU Temperature is 99 C, too hot to boot, waiting...
CPU Temperature is 90 C, too hot to boot, waiting...
CPU Temperature is 86 C, too hot to boot, waiting...
CPU Temperature is 84 C, too hot to boot, waiting...
CPU Temperature is 80 C, too hot to boot, waiting...
CPU Temperature is 80 C, too hot to boot, waiting...
CPU Temperature is 80 C, too hot to boot, waiting...
CPU: Temperature 78 C
Reset cause: WDOG
Board: TS-7970
```

These temperature tests show the TS-7970 with/without both the heatsink and enclosure. The HS-15x15x5 test data is provided as an example of a smaller heatsink, but this heatsink is not recommended for the TS-7970.



14.4 IO Specifications

The GPIO external to the board are all nominally 3.3V, but will vary depending on if they are CPU/FPGA pins.

The CPU pins can be adjusted in software and will have initial values in the device tree. This lets you adjust the drive strength, and pull strength of the IO. See the device tree for your kernel for further details on a specific IO.

The FPGA IO cannot be adjusted further in software.

IO	Typical Range	Absolute Range	Logic Low	Logic high	Drive strength
External CPU GPIO	0-3.3V	-0.5V to 3.3V Rail + 0.3V	0.3 * 3.3V Rail	0.7 * 3.3V Rail	27.5mA
External FPGA GPIO	0.3.3V	-0.5-3.75V	0.8	2.0	12mA

Refer to the MachXO Family Datasheet for more detail on the FPGA IO. Refer to the CPU quad or solo datasheet for further details on the CPU IO.

WARNING: Do not drive any IO from an external supply until 3.3V is up on the board. Doing so can violate the power sequencing of the board causing failures.

14.5 Rail Specifications

The TS-7970 generates all rails from either the 8-28VDC input, or the 5V input. This table does not document every rail. This will only cover those that can provide power to an external header for use in an application.

Direct 5V input will bypass our regulator, but the absolute max a supply can provide 5A to the board.

Rail	Current Available	Location
3.3V	200mA ^[1]	HD1 pin 23, HD2 pin 23, mPCIe, HD3 pin 3
5V	Quad core 2A, Solo 3A ^[2]	HD1 pins 15/16, HD3 pin 6, USB, mPCIe

1. ↑ Contact us if you need more on this rail
2. ↑ These limitations are only relevant if 8-28V is supplied into the board.

15 Revisions and Changes

15.1 TS-7970 PCB Revisions

Revision	Changes
A	<ul style="list-style-type: none"> ▪ Initial Release
B	<ul style="list-style-type: none"> ▪ Changed U17 to support the MPU-9250 9 axis accelerometer ▪ Changes TVS1 to use 4.7V_A ▪ Fix I210 ethernet LED polarity ▪ Use 153-ball eMMC to support bigger eMMCs on custom builds ▪ Added PU from 5V_A to SW_5V. This prevents a negative leakage on the the SW_5V rail. If the rail has a negative voltage it will otherwise not switch on. ▪ CPU pin U4 tied to ground. Used to detect REV B boards.
C	<ul style="list-style-type: none"> ▪ Not released
D	<ul style="list-style-type: none"> ▪ Changed PHY to Marvell 88E1512 due to published Microchip errata #9-10 which affects link reliability with some link partners. <div style="border: 1px solid black; padding: 5px; margin: 10px 0;"> <p>Note: This change should be transparent in Linux from older kernels, but if link problems are seen from older images make sure the Marvell PHY driver is actually disabled as it does not configure this PHY correctly. The genphy driver will communicate correctly with this PHY. Earlier shipping images had this enabled, but current images keep this driver disabled</p> </div> <ul style="list-style-type: none"> ▪ Changed to a larger Ethernet Magjack with separate centertaps as required by the PHY manufacturer. ▪ CPU pin C13 tied to ground. Used to detect REV D boards. ▪ Added two lane MIPI connector (CN1) ▪ Changed to smaller battery holder ▪ Pull USB_5V_DETECT on silabs low when CON_EN jumper is off. This fixes the bug on previous revisions which would cause the board to fail to boot when CON_EN is off and USB is connected on the P2 port before the TS-7970 is powered on. ▪ Added PUSH_SW# to HD1 which allows a pin to wake the board out of sleep. ▪ Changed pin 28 to Enable 2.5V REF to reduce power in silabs sleep mode ▪ Improvements for our internal production
E	<ul style="list-style-type: none"> ▪ Minor changes for internal production. ▪ H6 biased low to detect new rev

15.2 U-Boot Changelog

Jun-17-2015 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Added TS-7970 support
Jul-27-2015 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Added fix for PCIe hang in Linux. Some of the GPR1 regs were not being reset after a reboot. U-boot will now reset these before going into Linux. This hang was not present on all CPUs, usually solo, and only if PCIe is enabled in the kernel.
Oct-07-2015 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> TS-7970 now has a POR on every reboot TS-7970 can reload the FPGA from a /boot/ts7970.vme file PUSH_SW is now read through i2c to free up FPGA_IRQ_1
May-09-2016 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Updated to imx_v2015.04_3.14.52_1.1.0_ga branch Updated DDR config to latest NXP recommendations Includes new thermal driver. If the CPU has overheated and rebooted it will wait in u-boot until the system cools down to the temperature specified in the thermal fuses. These are adjustable one time in software. Disabled NFS umountall
May-27-2016 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Added tsmicroctl command to read adc values, or start the sleep mode for the board. Requires a silabs from May 27th 2016 or later to include the sleep mode. Added FPGA_RESET# through a signal rather than a FPGA POR. Requires FPGA REV 5 for FPGA reset to work correctly. <div style="border: 2px solid red; padding: 5px; margin-top: 10px;"> <p>WARNING: Do not update past this u-boot without having REV 5 of the FPGA or the FPGA will not be reset on startup.</p> </div>
Jun-03-2016 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Added suggested fixes for Micrel PHY errata. Added FPGA and Silabs revision to startup output.
Jan-11-2017 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Added REV D support Added Marvell PHY support Allow solo to boot at 85C instead of 80C, quad is still 80C.
Feb-17-2017 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Added check for 64bit ext4 filesystem.
Mar-27-2017 (ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7970-linux/u-boot/)	<ul style="list-style-type: none"> Changed string to indicate REV D/E for new boards. Added I2C recovery improvements for fixing stuck bus

15.3 FPGA Changelog

Check the FPGA rev with:

```
echo $($ (tshwctl --peek --addr 51)>>4)
```

Rev	Changes
0	<ul style="list-style-type: none"> Initial Release
1	<ul style="list-style-type: none"> Switched max3100 to use FPGA_IRQ_1 to leave FPGA_IRQ_0 to the silabs.
2	<ul style="list-style-type: none"> Corrected CTS/RTS polarity on MAX3100 Corrected flipped CPU UART CTS/RTS for bluetooth Corrected HD1 SPI bus
3	<ul style="list-style-type: none"> Disabled pulldown on HD1 SPI CS.
4	<ul style="list-style-type: none"> Fixed the FPGA ttyMAX* uarts
5	<ul style="list-style-type: none"> The signal FPGA_IRQ_0 is now FPGA_RESET which needs to be pulsed on reset by u-boot. This is implemented in the May-27-2016 release. Register 61, bit 1 is now used to force SPI at all times on the HD1 SPI pins rather than just on chip select assert. This should allow any GPIO to be used as chip selects.
6	<ul style="list-style-type: none"> Disables USB HUB 24mhz while in reset
7	<ul style="list-style-type: none"> Includes support for REV D pin changes.

Using the u-boot from Oct-07-2015 or later you can reload the FPGA during startup for custom FPGAs. During startup you will see u-boot reload this file:

```

Bytes transferred = 56341 (dc15 hex)
VME file checked: starting downloading to FPGA
Diamond Deployment Tool 3.5
CREATION DATE: Wed Oct 07 11:38:24 2015

Downloading FPGA 53248/56341 completed
FPGA downloaded successfully

```

15.4 Silabs Changelog

Revision	Changes
0	<ul style="list-style-type: none"> Initial Release
1	<ul style="list-style-type: none"> Added Sleep mode Blinks blue LED in low power modes. Sleep mode does this, as well as USB device connected with no power on the main VIN.
2	<ul style="list-style-type: none"> Added support for REV D boards. Earlier boards will continue to use REV 1 only.
3	<ul style="list-style-type: none"> Fixed Silabs not responding on I2C after sleep mode is used.

15.5 Software Images

15.5.1 Yocto Changelog

Quad/Dual Image	Solo/Duallite Image	Changes
ts-x11-image-ts4900-quad-20140905235640.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dora/ts-x11-image-ts4900-quad-20140905235640.rootfs.tar.bz2)	ts-x11-image-ts4900-solo-20140908160116.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dora/ts-x11-image-ts4900-solo-20140908160116.rootfs.tar.bz2)	<ul style="list-style-type: none"> ■ Initial Release
ts-x11-image-ts4900-quad-20141119190447.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dora/ts-x11-image-ts4900-quad-20141119190447.rootfs.tar.bz2)	ts-x11-image-ts4900-solo-20141119204157.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dora/ts-x11-image-ts4900-solo-20141119204157.rootfs.tar.bz2)	<ul style="list-style-type: none"> ■ Systemd default ■ Added /usr/lib/openssh/sftp-server (Fixes QtCreator/Eclipse deploy) ■ Added QtQuick ■ Added Sqlite to QT ■ Added early TS-7970 support. ■ Updated kernel with significant fixes, see github (https://github.com/embeddedarm/linux-3.10.17-imx6/commits/master) for more information.
ts-x11-image-ts4900-quad-20141224171440.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dora/ts-x11-image-ts4900-quad-20141224171440.rootfs.tar.bz2)	ts-x11-image-ts4900-solo-20141224175107.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dora/ts-x11-image-ts4900-solo-20141224175107.rootfs.tar.bz2)	<ul style="list-style-type: none"> ■ Updated Kernel <ul style="list-style-type: none"> ■ Fixed ISL RTC errors hardware builds that omit the RTC ■ Fixed I2C bus for 8390 ADC ■ Added small pop fix for sgtl5000 on the 8390 ■ Updated ts4900-utils <ul style="list-style-type: none"> ■ New util 8390adc for reading the low speed MCP ADC ■ Fixed tshwctl to support auto TX-EN RS485 on ttymxcl
ts-x11-image-ts4900-quad-20150331224909.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dizzy/ts-x11-image-ts4900-quad-20150331224909.rootfs.tar.bz2)	ts-x11-image-ts4900-solo-20150401003538.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dizzy/ts-x11-image-ts4900-solo-20150401003538.rootfs.tar.bz2)	<ul style="list-style-type: none"> ■ Updated to 3.10.53 kernel <ul style="list-style-type: none"> ■ Significant fixes to GPU, UARTs, CAN and more. ■ Added TS-TPC-8950 support ■ Fixed 7" twinkling pixels on TS-8390 w/solo ■ Included splash screen ■ Updated to Yocto Dizzy for new freescale GPU support ■ Added Chromium to default image (google-chrome) ■ Updated toolchain to match dizzy image ■ Included gstreamer in the image ■ Updated FPGA with crossbar, max3100 based spi uart, bluetooth fixes (REV C only)
ts-x11-image-ts4900-quad-20150527173205.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dizzy/ts-x11-image-ts4900-quad-20150527173205.rootfs.tar.bz2)	ts-x11-image-ts4900-solo-20150528210615.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dizzy/ts-x11-image-ts4900-solo-20150528210615.rootfs.tar.bz2)	<ul style="list-style-type: none"> ■ Fixed networkd ■ Enabled PCIe in default kernel <ul style="list-style-type: none"> ■ Added I210 support for TS-7970
ts-x11-image-ts4900-quad-20150620060219.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dizzy/ts-x11-image-ts4900-quad-20150620060219.rootfs.tar.bz2)	ts-x11-image-ts4900-solo-20150622150127.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/dizzy/ts-x11-image-ts4900-solo-20150622150127.rootfs.tar.bz2)	<ul style="list-style-type: none"> ■ Added TS-7970 support
ts-x11-image-tsimx6-20150821190815.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/fido/ts-x11-image-tsimx6-20150821190815.rootfs.tar.bz2)		<ul style="list-style-type: none"> ■ Updated to Yocto Fido <ul style="list-style-type: none"> ■ Removed GTK3 packages to reduce image size (GTK2 still available) ■ Removed distcc from default environment ■ Includes QT 5.4.3 ■ Included qtmultimedia, xcursor-transparent theme ■ Updated Kernel <ul style="list-style-type: none"> ■ Includes fix for rare screen flip issues

<p>ts-x11-image-tsimx6-20150821190815.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/fido/ts-x11-image-tsimx6-20150821190815.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Included significantly fixed support for the TS-7970 <ul style="list-style-type: none"> ■ I210 support is fixed, but some prototype boards will need to be RMA'd (https://www.embeddedarm.com/support/rma.php) to get MACs assigned. ■ All UARTs are now working ■ Included tsmicroctl for reading the silabs ADC (p10-12 4-20mA included) ■ Included load_fpga for software reloading fpgas later after boot ■ Updated TS-4900 FPGA to have CTS/RTS fixed for bluetooth, and corrected CTS/RTS polarity on the max3100s
<p>ts-x11-image-tsimx6-20151014183028.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/fido/ts-x11-image-tsimx6-20151014183028.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Corrected defconfig used in kernel <ul style="list-style-type: none"> ■ Fixed WIFI and other modules ■ If used with the u-boot release from 10-14-2015 this fixes the mac address for the smsc95xx
<p>ts-x11-image-tsimx6-20151221232637.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/fido/ts-x11-image-tsimx6-20151221232637.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Fixed MAC address to use device tree as well as parameter for the latest u-boot support. ■ Fixed tsgpio driver which was causing some incorrect DIO sets. <ul style="list-style-type: none"> ■ The WIFI driver uses tsgpio for toggling the enable which also corrects the behavior of ifdown/ifup wlan0. ■ Added rsync and lighttpd-cgi support
<p>ts-x11-image-tsimx6-20160512161729.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/fido/ts-x11-image-tsimx6-20160512161729.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Added 100kohm pullups to the onboard/offboard SPI chip selects.
<p>ts-x11-image-tsimx6-20161116215413.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/jethro/ts-x11-image-tsimx6-20161116215413.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Updated to Yocto Jethro ■ Updates to QT 5.5 ■ Updated to 4.1.15 based on Freescale/NXP's imx_4.1.15_1.0.0_ga. ■ Added improved support for TS-TPC-7990 ■ New tshwctl with crossbar support.
<p>ts-x11-image-tsimx6-20170301225516.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/ts-x11-image-tsimx6-20170301225516.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Updated to Yocto Morty 2.2.1 with the same imx_4.1.15_1.0.0_ga kernel ■ Includes QT 5.7.1 ■ Included additional alsa utilities
<p>ts-x11-image-tsimx6-20170731205110.rootfs.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/yocto/morty/ts-x11-image-tsimx6-20170731205110.rootfs.tar.bz2)</p>	<ul style="list-style-type: none"> ■ Updated to Morty 2.2.2 ■ Included QT Quick 1.x/2.x support ■ Added support for TS-TPC-7990 REV C in kernel and ts4900-utils ■ Updated kernel <ul style="list-style-type: none"> ■ Fixed issue with ttyMAX* UARTs losing data or requiring the user to transmit before it continues to receive again ■ Fixed issue with ttyMAX* loopbacks dropping the first character ■ Added wilc3000 support for TS-TPC-7990 REV C WIFI

15.5.2 Debian Changelog

Image	Changes
debian-armhf-wheezy-20140929.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-wheezy-20140929.tar.bz2)	<ul style="list-style-type: none"> ■ Initial Release
debian-armhf-wheezy-20141125.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-wheezy-20141125.tar.bz2)	<ul style="list-style-type: none"> ■ Updated kernel with significant fixes, see github (https://github.com/embeddedarm/linux-3.10.17-ixmx6/commits/master) for more information. ■ Included first TS-7970 FPGA
debian-armhf-jessie-20160825.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20160825.tar.bz2)	<ul style="list-style-type: none"> ■ New kernel - 3.10.53 (from freescale's 3.10.53_1.1.0_ga) instead of 3.10.17. <ul style="list-style-type: none"> ■ Fixed CAN dropped frames (just under 1% of frames were dropped on 3.10.17) ■ Fixed reported UART RX fifo overflows ■ GPU fixes ■ Kernel includes compiled in splash screen for quick graphical response on boot ■ TS-TPC-8950 support added ■ New FPGA (crossbar added, bluetooth fixed, and max3100 implemented) ■ Added bluez, wireless-tools, usbutils, nfs-common, and pciutils into the image. ■ Added Openssh server (generates on first boot)
debian-armhf-jessie-20150526.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20150526.tar.bz2)	<ul style="list-style-type: none"> ■ First update to Debian Jessie
debian-armhf-jessie-20151008.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20151008.tar.bz2)	<ul style="list-style-type: none"> ■ Included kernel support for TS-7970 REV A ■ Updated to latest TS-4900 FPGA (20150603) ■ Included openssh, generates keys on first boot. Remove /etc/ssh/*key* to regenerate. ■ Included latest ts4900-utils with TS-7970 support.
debian-armhf-jessie-20160512.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20160512.tar.bz2)	<ul style="list-style-type: none"> ■ Fixed TS-7970 ttyMAX uarts (requires FPGA update) ■ Fixed resolv.conf symlink to use resolvd ■ Updated to 3.14.52 kernel ■ Corrected TS-TPC-8950 calibration
debian-armhf-jessie-20160512.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20160512.tar.bz2)	<ul style="list-style-type: none"> ■ Moved to 4.1.15 kernel ■ Updated Debian to latest Jessie changes ■ Added latest ts4900-utils with improved TS-TPC-7990 support.
debian-armhf-jessie-20170123.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20170123.tar.bz2)	<ul style="list-style-type: none"> ■ Added support for TS-7970 REV D hardware ■ Added support for TS-7990 REV B hardware
debian-armhf-jessie-20170306.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20170306.tar.bz2)	<ul style="list-style-type: none"> ■ Fixed resolv.conf symlink ■ Added nfs-common ■ Cleaned up old temporary files
debian-armhf-jessie-20170327.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20170327.tar.bz2)	<ul style="list-style-type: none"> ■ Fixed regression in TS-TPC-8950 support ■ Adds root.version to list image date
debian-armhf-jessie-20170419.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20170419.tar.bz2)	<ul style="list-style-type: none"> ■ Fixed issue of missing U-boot splash screen disabling the backlight on REV B boards. ■ Fixed potential issue with WIFI not being recognized. ■ Added support for #TS-DC799-SILO board.
debian-armhf-jessie-20170731.tar.bz2	

(ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/debian/debian-armhf-jessie-20170731.tar.bz2)

- Added support for TS-TPC-7990 REV C in kernel and ts4900-utils
- Updated kernel
 - Fixed issue with ttyMAX* UARTs losing data or requiring the user to transmit before it continues to receive again
 - Fixed issue with ttyMAX* loopbacks dropping the first character
 - Added wilc3000 support for TS-TPC-7990 REV C WIFI

15.5.3 Arch Linux Changelog

Image	Changes
arch-armhf-20140929.tar.gz (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/arch/arch-armhf-20140929.tar.gz)	Initial Release

15.5.4 Ubuntu Linux Changelog

Image	Changes
ubuntu-armhf-16.04-20160407.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu/ubuntu-armhf-16.04-20160407.tar.bz2)	<ul style="list-style-type: none"> ■ Initial Release
ubuntu-armhf-16.04-20160818.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu/ubuntu-armhf-16.04-20160818.tar.bz2)	<ul style="list-style-type: none"> ■ Bumped from 3.14.52 to 4.1.15 kernel. This adds support for the TS-TPC-7990. ■ Added more common packages, mmc, can-utils, etc.
ubuntu-armhf-16.04-20170306.tar.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu/ubuntu-armhf-16.04-20170306.tar.bz2)	<ul style="list-style-type: none"> ■ Updated ts4900-utils for final TS-7970/TS-TPC-7990 ■ Added TS-TPC-7990 REV B support

15.5.5 Ubuntu Core Linux Changelog

Image	Changes
ubuntu-core-16-2016-12-22.img.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu-core/ubuntu-core-16-2016-12-22.img.bz2)	<ul style="list-style-type: none"> ■ Initial Release
ubuntu-core-16-2017-04-21.img.bz2 (ftp://ftp.embeddedarm.com/ts-socket-macrocontrollers/ts-4900-linux/distributions/ubuntu-core/ubuntu-core-16-2017-04-21.img.bz2)	<ul style="list-style-type: none"> ■ Fixed boot scripts to work with core updates ■ Added support for the TS-4900 carrier board specific device trees. ■ Added support for the TS-7970 ■ Added support for the TS-TPC-7990 ■ Updated Kernel <ul style="list-style-type: none"> ■ Added ttyMAX* uart support ■ Added fixes from our main Linux kernel to the RTC driver ■ Fixed ethernet timing

15.6 TS-7970 Errata

Issue	Status	Description
RS232 prevents booting on REV A	Workarounds available, fixed in REV B	Early TS-7970 REV A boards may fail to boot if RS232 is connected before the board is powered. A small amount of RS232 idle negative voltage leaks from the transceiver to the FET controlling the switched 5V. The FET will not toggle while the output has a negative voltage, so the 5V rail never comes up. If this is a concern or if the issue is seen, we can rework the board to have a 2.5ohm resistor from SW_5V to 5V_A on U47 pins 4 and 5. REV A boards shipped after 01/13/2016 include this fix.
Boot is prevented if the USB Device port (not host) is plugged in without the console enable jumper.	Fixed in REV C	If the console jumper is not installed the silabs has USB VBUS, but no data signals. This puts the USB device into a locked up state while it waits to communicate on this bus. Due to this lockup it is unable to monitor voltages and turn on the SW_5V to the reset of the board. The fix is to disconnect VBUS from the silabs which is done on the REV C PCB. A cable without VCC can be made to work around the issue, or submit an RMA.

16 Product Notes

16.1 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.

16.2 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 (<https://www.embeddedarm.com/support/rma.php>) to arrange for any repair service and to obtain repair charge information.

Retrieved from "<https://wiki.embeddedarm.com/w//index.php?title=TS-7970&oldid=9096>"