

# TS-7350 Single Board Computer Documentation

---

Brigham Young University Idaho

For Idaho National Laboratories

Revised: July 23, 2009

# Contents

<b>1</b>	<b>Overview of Single Board Computer (SBC)</b>	<b>2</b>
1.1	Technologic Systems TS-7350 SBC . . . . .	2
1.2	Official Documentation . . . . .	2
1.3	Specifications . . . . .	2
1.3.1	General Specifications . . . . .	2
1.3.2	Detailed Specifications . . . . .	3
1.4	Purchasing Information . . . . .	3
1.5	SD Card Information . . . . .	4
<b>2</b>	<b>Chucker Box (SBC) Configuration</b>	<b>5</b>
2.1	Preliminary . . . . .	5
2.2	Download SD Card Image . . . . .	5
2.3	Prepare SD Card . . . . .	5
2.4	Modify Fast Boot . . . . .	6
2.5	Configure SBC for DHCP . . . . .	7
2.6	Required Software Installation . . . . .	8
2.7	Download Cell Phone Control Scripts . . . . .	8
2.8	Bluetooth and Software Configuration . . . . .	8
2.9	Setup Auto Login and Auto Run Script . . . . .	10
<b>3</b>	<b>General Linux Instructions</b>	<b>12</b>
3.1	Mount a USB Thumb Drive . . . . .	12
3.2	Check Disk Space . . . . .	12
3.3	Servicing a Corrupted SD Card . . . . .	12
3.4	Simple vi Commands . . . . .	12
3.5	Optional, Extra, and Advanced . . . . .	13
3.5.1	Compile Kernel . . . . .	13
3.5.2	Install Debian Lenny . . . . .	13

# 1 Overview of Single Board Computer (SBC)

This section consists of basic, quick information regarding the Technologic Systems TS-7350 Single Board Computer (SBC) and it's hardware including the SD Card and it's partition layout. Most of this information has been extracted from the [TS-7350](#) homepage. In order to prevent duplicated information, following the first section, this document will focus mostly on INL application specific information.

## 1.1 Technologic Systems TS-7350 SBC

The TS-7350 is a compact full-featured Single Board Computer (SBC) based upon the Cirrus EP9302 200MHz ARM9 CPU, which provides a standard set of on-board peripherals. It features a programmable FPGA which is connected to a dedicated framebuffer and provides additional on-board peripherals. With a simple custom peripheral board and a custom FPGA load, your LCD touchscreen of choice can be integrated with the TS-7350. The TS-7350 runs Linux 2.6 out of the box.

## 1.2 Official Documentation

The TS-7350 has excellent online documentation and resources. In order to prevent duplicated and non-specific information, it is recommended that these resources are used.

- [TS-7350/TS-7370 Preliminary Manual](#)
- [TS-LCD-READY Getting Started Sheet](#)
- [Linux for ARM on TS-72XX User's Guide](#)
- [Getting Started with TS-Linux](#)
- [TS-7000 Yahoo User Group \(Support Forums\)](#)
- [TS-7350 Linux FTP Repository \(Docs, Binaries, Downloads\)](#)
- [Latest Factory 512MB SD Card Image](#)
- [TS-7350 Schematic](#)
- [TS-7350 Mechanical Drawing](#)

## 1.3 Specifications

This section consists of both overall and detailed technical specifications of the TS-7350.

### 1.3.1 General Specifications

The TS-7350 features a multi-purpose 200MHz ARM9 CPU. It allows development of multi-function embedded applications through its multiple peripheral interfaces, which includes on-board RAM, 10/100 ethernet, USB 2.0 host, serial ports, SD Card socket, A/D channels, digital I/O lines, temperature sensor, real time clock, and more.

The TS-7350 features a general purpose 40 pin header. The 40 pin header provides the following features:

- I2C Bus and SPI Bus
- 4 12-bit ADC
- 6 Latched Outputs
- 7 Buffered Inputs
- 3 Digital Inputs/Outputs
- 1 Console COM Port (requires TS-9445 mini-peripheral for RS-232 levels)
- External Reset, Power, Ground, Frame and JTAG signals

The default FPGA load provides additional peripherals, such as SD Card socket and serial ports. A video core is not included in the default load. In addition, the FPGA can be configured on the fly to either load a 16-bit PC/104 bus on the 64-pin PC/104 connector or use it as general purpose I/O lines.

The TS-7350 computers are extremely rugged and reliable. Temperature operation is fanless and range is industrial standard at lower CPU speeds. Power consumption is approximately 2W (400mA @ 5VDC). Power input range is flexible from 5VDC to 28VDC.

The TS-7350/TS-7370 provides multiple COM ports using both TTL (2 total) and RS-232 (5 total) levels. Two optional RS-485 ports are available with DMX/RDM support. The EP9302 processor provides two COM ports (TTL Console at the JTAG header and RS-232 at COM1 header). The remaining COM ports are provided by the FPGA through the proprietary XUART core and are 9-bit serial capable. Please refer to the manual for further information.

### 1.3.2 Detailed Specifications

- 200MHz ARM9 CPU
- 32MB SDRAM (64-128MB opt)
- User-programmable 5K LUT FPGA
- Flexible 64-pin FPGA-PC/104 connector
- 8MB RAM Framebuffer, no video core
- Able to drive TFT-LCDs via custom FPGA
- 1 10/100 ethernet port
- 2 USB 2.0 (12Mbit/s max)
- 1 SD Card slot (up to 6MB/s DMA)
- 5 RS-232, 2 TTL COM ports
- 40-pin header with ADC, SPI, I2C, DIO...
- Optional Temp Sensor, RTC, RS-485
- Fanless -40 to +70C, +85C 166Mhz
- 5-28VDC Power Input
- Boots Linux 2.6 in about 1 second
- Debian Linux is default on SD Card
- Requires TS-9445 peripheral for Console
- Supports out-of-the-box [Eclipse IDE](#)

### 1.4 Purchasing Information

This section consists of the ordering information to duplicate the chucker box prototype delivered to INL. There are two total expenses: (1) development expense which includes the first time expense development kit and TS-7350 and (2) production or single chucker box expense which only includes the TS-7350.

Part Number	Description	Price (QTY1)
TS-7350-32	TS-7350 (200MHz ARM9; 32MB RAM)	\$159
OP-BBRTC	Battery Backed <a href="#">Real Time Clock</a>	\$10
OP-ROHS-NC	Built 100% RoHS compliant	\$0
OP-TMPSENSE	On-Board Temperature Sensor	\$3
KIT-LCDR	<a href="#">TS-LCD-READY Development Kit</a>	\$150
TOTAL W/ DEVEL KIT:	First Time Expense	\$322
TOTAL W/O DEVEL KIT:	Production Expense	\$172

Table 1: Parts List for Prototype Chucker Box.

## 1.5 SD Card Information

The SD Card contains the operating system for the TS-7350 and is inserted in the SD Card slot located on the bottom of the board. The SD cards shipped with and for the TS-7350 contain a special four partition scheme. The first partition is a VFAT partition. On cards larger than 1GB, this partition contains Eclipse and other tools, documentation, and so forth. On cards smaller than 1GB this partition is empty. The second partition contains a raw image of the kernel to be booted on the board, while the third partition contains an initial ramdisk (initrd or Fast Boot) which the kernel uses as its root filesystem until the fastboot shell exits. The fourth partition contains the Debian Linux distribution, and contains the root filesystem that is used when the fastboot shell exits and the full boot commences.

- /dev/tssdcarda: the whole device with MBR bootup code
  - /dev/tssdcarda1: 1.5GB vfat partition with Eclipse IDE or 4MB empty
  - /dev/tssdcarda2: 4MB for the uncompressed Linux Kernel Image
  - /dev/tssdcarda3: 4MB for the uncompressed initial ramdisk with busybox filesystem (Fast Boot)
  - /dev/tssdcarda4: complete Debian Linux filesystem JFS type

## 2 Chucker Box (SBC) Configuration

The purpose of this section is to walk the SBC programmer through step by step on how to create and tweak a specialized Linux OS for the INL Wireless Testbed. For development purposes, the best solution is to have another Debian Linux PC with an SD card reader attached.

### 2.1 Preliminary

There are several steps to take. This assumes you have a Debian based PC with internet connection.

1. Open the terminal (If you're in a GNOME GUI environment, the location of the terminal is in Applications → Accessories → Terminal) and login as super user (root)

```
su
```

2. Download the required software for dealing with the JFS filesystem

```
apt-get install jfsutils gparted
```

3. Make a directory for TS-7350 downloads and development files and then change into that directory

```
mkdir /TS7350  
cd /TS7350
```

### 2.2 Download SD Card Image

1. Download the latest .dd image from the TS-7350 Linux Repositories for the 512MB SD Card media (if you have a larger capacity card, this guide will be walking you through how to enlarge the last partition to make more room)

```
wget ftp://ftp.embeddedarm.com/ts-arm-sbc/ts-7350-linux/binaries/ts-images/512mbsd-latest.dd.bz2
```

2. Extract the .dd image from the bz2 archive

```
bzip2 -d 512mbsd-latest.dd.bz2
```

### 2.3 Prepare SD Card

1. Insert the SD Card to be used in the SBC into the SD Card reader attached to the USB port of the computer.
2. Find out which device the SD card was mounted on and take note of the location (ie /dev/sdb)

```
fdisk -l
```

3. Unmount the SD Card (this is just in case it was automounted by Linux)

```
umount /dev/sdb*
```

4. Use the dd command to copy the bitwise information from the 512mbsd-latest.dd to the SD Card. **WARNING: THIS WILL DESTROY ANY DATA ON THE SD CARD OR DEVICE! Make sure you know which device the SD card is.** Note: this step will take some time depending on the R/W speed of the SD Card (approx. 15 to 20 minutes)

```
dd if=512mbsd-latest.dd of=/dev/sdb
```

5. Unmount the SD Card, remove it from the SD card reader, insert it again, unmount the automatically mounted drives, and then check to make sure the filesystems are sanitary

```
umount /dev/sdb*  
[remove card]  
[insert card]  
umount /dev/sdb* && jfs_fsck -v -f /dev/sdb4 && e2fsck /dev/sdb3
```

- This step is optional. Use parted to expand sdb4 to fit a larger capacity card. If you are using a 512MB card, skip this step. Resize the filesystem on partition to start at start and end at end megabytes. Get the partition start

```
parted resize /dev/sdb4 7 531
```

## 2.4 Modify Fast Boot

- Boot to the fast boot partition. Refer to official documentation for instructions on how to do this if you need.
- Use vi command to edit the linuxrc-sdroot file located on the root directory.

```
vi linuxrc-sdroot
```

- Replace the line that reads “fsck /dev/tssdcard4” with “jfs\_fsck /dev/tssdcard4”
- The final file contents should look like this:

```
#!/bin/sh
# Copyright (c) 2007, Technologic Systems.
# All rights reserved.
#
# Roots to SD flash card, assumes partition #3 (/dev/sdcard0/disc0/part3)
# symlink to /linuxrc and run "save" to use

export PATH=/bin:/sbin:/sbin:/mnt/root/bin:/mnt/root/sbin:/mnt/root/usr/bin:/mnt/root/usr/sbin
:/mnt/root/usr/local/bin:/mnt/root/usr/local/sbin
export LD_LIBRARY_PATH=/lib:/usr/lib
export CONSOLE=/dev/ttyAM0

mount -t proc none /proc
mount -t sysfs none /sys
mount -t tmpfs none /dev
mdev -s
mkdir /dev/pts /dev/shm
mount -t devpts none /dev/pts
mount -t tmpfs none /dev/shm

setconsole $CONSOLE
stty -F $CONSOLE ospeed 115200 > /dev/null 2>&1
hostname ts7350
check-usb-update >/dev/null 2>&1 </dev/null &

echo ">> Booting to SD Card..." > $CONSOLE

(
peekpoke 16 0x600ff0d6 0x3
insmod /tssdcard.ko
mdev -s
if [ -e /dev/tssdcarda4 -a -e /mnt/root/notrootfs ]; then
mount -o ro /dev/tssdcarda4 /mnt/root
jfs_fsck /dev/tssdcarda4
mount -o remount,rw /mnt/root
fi
) > /dev/null 2>&1

if [ -e /mnt/root/notrootfs -o -e /mnt/root/fastboot -o ! -e /mnt/root/sbin/init ]; then

if [ -e /mnt/root/fastboot ]; then
echo ">> Directory '/fastboot' found. Booting to initrd instead..." > $CONSOLE
else
echo ">> SD Card failed. Booting to initrd..." > $CONSOLE
fi

(
ifconfig lo 127.0.0.1 up
route add -net 127.0.0.0 netmask 255.0.0.0 lo
ifconfig eth0 192.168.0.50 up
#route add default gateway 192.168.0.1 eth0
/sbin/telnetd
```

```

insmod /ts7000_nand.ko
insmod /tsuart1.ko
insmod /tsuart7350.ko
mdev -s

if [ -e /dev/mtdblock3 -a -e /mnt/root/notrootfs ]; then
    mount -t yaffs2 -o ro /dev/mtdblock3 /mnt/root
fi
) > /dev/null 2>&1 &

(
    export BOOTTIME='eptime'
    export ENV=/shinit
    exec /bin/sh -i < $CONSOLE > $CONSOLE 2>&1
)

wait
killall busybox telnetd > /dev/null 2>&1
echo ">> Booting Linux..." > $CONSOLE
cd /mnt/root
pivot_root . ./initrd
./bin/mount -n --move ./initrd/sys ./sys
./bin/mount -n --move ./initrd/proc ./proc
exec ./usr/sbin/chroot . ./sbin/init < .$CONSOLE > .$CONSOLE 2>&1

else

#verify if there is a kernel and issue bootloader command. if not, then:

cd /mnt/root
pivot_root . ./initrd
./bin/mount -n --move ./initrd/sys ./sys
./bin/mount -n --move ./initrd/proc ./proc
exec ./usr/sbin/chroot . ./sbin/init < .$CONSOLE > .$CONSOLE 2>&1

fi

```

- Exit vi, set the SBC to skip the fast boot (boot directly to Debian Linux which is about 15 seconds), and then save all the changes.

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

## 2.5 Configure SBC for DHCP

This section will review how to setup the SBC to get a dynamically assigned IP address which will allow it to connect to the internet and network.

- First, edit the `/etc/udev/rules.d/z25_persistent-net.rules` file to reflect that we will be using the `eth0` connection. The final file should look like this:

```

# This file was automatically generated by the /lib/udev/write_net_rules
# program, probably run by the persistent-net-generator.rules rules file.
#
# You can modify it, as long as you keep each rule on a single line.
# MAC addresses must be written in lowercase.

SUBSYSTEM=="net", DRIVERS=="?* ", ATTRS{address}=="00:d0:69:41:e4:41", NAME="eth0"
#SUBSYSTEM=="net", DRIVERS=="?* ", ATTRS{address}=="00:d0:69:4f:76:d8", NAME="eth1"

```

- Next, in the file `/etc/network/interfaces`, comment out anything that isn't `eth0` and also assign DHCP to `eth0`. The file should look like this:

```

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

auto lo
iface lo inet loopback

```

```
auto eth0
iface eth0 inet dhcp
#iface eth0 inet static
# address 192.168.0.50
# network 192.168.0.0
# netmask 255.255.255.0
# broadcast 192.168.0.255
# gateway 192.168.0.1

#auto eth1
#iface eth1 inet dhcp
#iface eth1 inet static
# address 192.168.1.50
# network 192.168.1.0
# netmask 255.255.255.0
# broadcast 192.168.1.255
```

- The final step will be to essentially release the static IP address that the SBC currently has on it, and then force it to obtain an IP address via DHCP. Run the following command:

```
/etc/init.d/networking restart
```

## 2.6 Required Software Installation

This section consists of how to install the software required for the bluetooth control as well as any additional troubleshooting tools.

1. To install the software, boot to the full Debian Linux partition and type the following into the terminal:

```
apt-get install bluetooth bluez bluezutils python2.5 bluez_hcidump
```

## 2.7 Download Cell Phone Control Scripts

The easiest way to download the cell phone control script is to download it onto a USB drive, mount the USB drive on the SBC, copy the files onto a known directory, and then unmount the USB drive.

1. Make a directory to mount the USB drive to, find out which drive is your USB drive by using `fdisk -l`, and then mount it to the created directory.

```
mkdir /media/usbdrive
fdisk -l
mount -t vfat /dev/sdxx /media/usbdrive
```

2. Moving files between the USB Flash drive to the SBC is easy using the `cp` command.

```
cp /media/usbdrive /home/destination
```

3. Unmount the USB Flash drive from the SBC

```
umount /dev/sdb*
```

## 2.8 Bluetooth and Software Configuration

This section deals with the configuration of the system software which includes bluetooth. For more information on bluetooth commands, refer to [this guide](#) online.

1. After installing the required packages for bluetooth, plugin the USB Bluetooth Dongle and enable bluetooth by typing

```
hciconfig hci0 up
```

2. The bluetooth device should now be enabled (usually indicated by a blue light). Next step is getting the bluetooth device into a discoverable mode. **UNFORTUNATELY**, the exact steps to this section were unable to be included due to time constraints and the complexity involved. Everybody from the BYU-Idaho team contributed something to this part, and nobody documented how they were able to accomplish this task. Basically, it was beat with six hammers until it worked. The file of most interest was `/etc/bluetooth/hcid.conf`. Also, the `/usr/share/doc/bluez-utils/examples/passkey-agent` file needed to be compiled and installed to use pairing password. The `hcid.conf` file ended up looking like this:

```
#
# HCI daemon configuration file.
#

# HCID options
options {
    # Automatically initialize new devices
    autoinit yes;

    # Security Manager mode
    # none - Security manager disabled
    # auto - Use local PIN for incoming connections
    # user - Always ask user for a PIN
    #
    security auto;

    # Pairing mode
    # none - Pairing disabled
    # multi - Allow pairing with already paired devices
    # once - Pair once and deny successive attempts
    pairing multi;

    # Default PIN code for incoming connections
    #passkey "1234";
    passkey "0000";
#auth disable;
#encrypt disable;

}

# Default settings for HCI devices
device {
    # Local device name
    # %d - device id
    # %h - host name
    name "%h-%d";

    # Local device class
    class 0x3e0100;

    # Default packet type
    #pkt_type DH1,DM1,HV1;

    # Inquiry and Page scan
    iscan enable; pscan enable;
    discovto 0;

    # Default link mode
    # none - no specific policy
    # accept - always accept incoming connections
    # master - become master on incoming connections,
    # deny role switch on outgoing connections
    lm accept;

    # Default link policy
    # none - no specific policy
    # rswitch - allow role switch
    # hold - allow hold mode
    # sniff - allow sniff mode
    # park - allow park mode
```

```
lp rswitch , hold , sniff , park ;  
}
```

## 2.9 Setup Auto Login and Auto Run Script

1. Now that the device is discoverable, the next step is to setup the SBC to automatically login as root. The first thing to be done is to install mingetty

```
apt-get install mingetty
```

2. Edit the `/etc/inittab` file to look like the following (the most important line being `1:2345:respawn:/sbin/mingetty --autologin root ttyAM0`):

```
# /etc/inittab: init(8) configuration.  
# $Id: inittab,v 1.91 2002/01/25 13:35:21 miquels Exp $  
  
# The default runlevel.  
id:2:initdefault:  
  
# Boot-time system configuration/initialization script.  
# This is run first except when booting in emergency (-b) mode.  
si::sysinit:/etc/init.d/rcS  
  
# What to do in single-user mode.  
~:S:wait:/sbin/sulogin  
  
# /etc/init.d executes the S and K scripts upon change  
# of runlevel.  
#  
# Runlevel 0 is halt.  
# Runlevel 1 is single-user.  
# Runlevels 2-5 are multi-user.  
# Runlevel 6 is reboot.  
  
10:0:wait:/etc/init.d/rc 0  
11:1:wait:/etc/init.d/rc 1  
12:2:wait:/etc/init.d/rc 2  
13:3:wait:/etc/init.d/rc 3  
14:4:wait:/etc/init.d/rc 4  
15:5:wait:/etc/init.d/rc 5  
16:6:wait:/etc/init.d/rc 6  
# Normally not reached, but fallthrough in case of emergency.  
z6:6:respawn:/sbin/sulogin  
  
# What to do when CTRL-ALT-DEL is pressed.  
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now  
  
# Action on special keypress (ALT-UpArrow).  
#kb::kbrequest:/bin/echo "Keyboard Request—edit /etc/inittab to let this work."  
  
# What to do when the power fails/returns.  
pf::powerwait:/etc/init.d/powerfail start  
pn::powerfailnow:/etc/init.d/powerfail now  
po::powerokwait:/etc/init.d/powerfail stop  
  
# /sbin/getty invocations for the runlevels.  
#  
# The "id" field MUST be the same as the last  
# characters of the device (after "tty").  
#  
# Format:  
# <id>:<runlevels>:<action>:<process>  
#  
# Note that on most Debian systems tty7 is used by the X Window System,  
# so if you want to add more getty's go ahead but skip tty7 if you run X.  
#  
#1:2345:respawn:/sbin/getty 38400 tty1  
#1:2345:respawn:/bin/login -f root tty1 </dev/tty1 >/dev/tty1 2>&1  
#2:23:respawn:/sbin/getty 38400 tty2  
#3:23:respawn:/sbin/getty 38400 tty3  
#4:23:respawn:/sbin/getty 38400 tty4
```

```
#5:23:respawn:/sbin/getty 38400 tty5
#6:23:respawn:/sbin/getty 38400 tty6

1:2345:respawn:/sbin/mingetty --autologin root ttyAM0

# Example how to put a getty on a serial line (for a terminal)
#
#T0:23:respawn:/sbin/getty -L ttyAM0 115200 vt100
#T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100
#T1:23:respawn:/sbin/getty -L ttyS1 9600 vt100

# Example how to put a getty on a modem line.
#
#T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3
```

3. The next step is to add the cell phone control script and set it to boot on startup. You'll need to transfer the file by some means to the SBC (i.e. USB Drive). After the script is saved to a location on the SBC, add the script to the boot-up file. Do this by adding the following line to the `/root/.bash_profile` file:

```
python /location/of/controlscript.py

# To shutdown board automatically when script is finished , use this line:
# python /location/of/controlscript.py ; poweroff
```

## 3 General Linux Instructions

This section consists of useful commands when working with the TS-7350 SBC. These commands were used often when configuring and developing the SBC for INL. For more linux commands and instructions, please refer to the [Getting Started with TS-Linux](#) guide from Technologic Systems.

### 3.1 Mount a USB Thumb Drive

1. Make a directory to mount the USB drive to, find out which drive is your USB drive by using `fdisk -l`, and then mount it to the created directory.

```
mkdir /media/usbdribe
fdisk -l
mount -t vfat /dev/sdxx /media/usbdribe
```

2. Moving files between the USB Flash drive to the SBC is easy using the `cp` command.

```
cp /media/usbdribe /home/destination
```

### 3.2 Check Disk Space

1. To check disk space usage, use the `df` command

```
df -lH
```

### 3.3 Servicing a Corrupted SD Card

1. If somehow the SD Card filesystem becomes corrupted and the SBC will no longer boot to Debian Linux from it, then these steps must be performed. Be sure that the `jfsutils` package has been installed as instructed in the preliminary section of this document. Insert the SD Card into the SD Card reader, unmount what is automatically mounted, and perform the sanitary checks.

```
umount /dev/sdb*
jfs_fsck -v -f /dev/sdb4
e2fsck /dev/sdb3
```

### 3.4 Simple vi Commands

The following list will get you started quickly in editing with `vi`. If you need more information, refer to this guide: <http://www.cs.fsu.edu/general/vimanual.html>. Another alternative editor installed on the SBC is `nano`.

- `j` - down
- `h` - left
- `l` - right
- `k` - up
- `i` - insert
- `a` - insert after current
- `r` - replace
- `esc` - escape out of editing
- `shift+colon` (`↑ + :`) - command (`w` is write, `q` is quit)

## 3.5 Optional, Extra, and Advanced

### 3.5.1 Compile Kernel

1. The following link is a resource on the technologic systems website which outlines the overall process to compile a kernel for the TS-7200 boards and can be adapted to the TS-7350.

- [Linux 2.6 for TS-7200 Computers](#)

### 3.5.2 Install Debian Lenny

1. The following guide was created by a TS-7390 user and outlines the steps required to install Debian Lenny (the newest version at the time of this writing) onto the TS-7390. It is adaptable to run on the TS-7350.

- <http://ted.openavr.org/Lenny-on-ts7390>