TSHARC UART Open Source Linux Driver Documentation

Document #:	AD-110028-001
Title:	**TSHARC UART Open Source Linux Driver Documentation**
Subtitle:	
Date:	12-August-2011
Description:	This document describes the reference TSHARC UART driver software developed for Linux.

Revision History

Version	Date	By	Description
001	01/03/2011	SG	• Initial Revision

Table of Contents

1.0 Introduction

The goal of the documentation presented is to provide instructions on how to quickly and easily setup and calibrate touch controller drivers for most embedded or desktop Linux configurations. After this setup and calibration, applications running on target machine will be aware of the position and touch state of the touch screen connected to the touch controller. We have provided source code and discuss some common open source touch libraries to enable the driver to function on any platform the driver source code or library source code is compiled for.

An installation script (“install.sh”), a controller test utility (“inputverify”) and a controller activation utility (“inputactivate”) are provided tools that were created to help simply the task of setting up a touch controller driver for the target computer. These tools and the libraries discussed in this documentation are only recommendations and there are a variety of many good open source solutions available. However, for non-USB interfaces, at a minimum the Microchip kernel module will need to be compiled from source code (unless a kernel module is already available from the mainstream Linux kernel tree such as the “hampshire” kernel module) and activated (by way of read bytes from the controller’s associated device path to the loaded Microchip kernel module).

After all of the documentation steps have been completed for the intended target platform, a kernel device path will appear under “/dev/input” that will send touch state information (in evdev format) to any application or touch library that reads this location. Normally, a library will be setup to read from this location and the application will process the touch state information in the form of mouse events interpreted from the GUI framework the application is using.

Preliminary Steps

In order for the included installation script and utilities to function, the provided source code will need to be compiled. To setup the environment for compiling source code for the target system, please see your Linux distributions documentation on how to best accomplish this. As a convenience, an “ubuntu10-10.sh” script is provided to setup all the necessary build dependencies such that the documentation steps may be directly followed on the Ubuntu 10.10 Linux platform. This script may be run from command prompt from within the installation directory as first step using the command “sh ./ubuntu10-10.sh”. Scripts for other Linux distributions will be added in future driver updates.

Extract installation files to a directory path that does not contain any spaces (to avoid confusing gcc or build tools) using a command such as "tar zxvf TSHARC-LINUX-UART-V102.tar.gz" (or by using nautilus or other file manager plug-in) followed by running the **"make”** command then the **“sh ./install-module.sh"** command at the root of the extracted files.

Root access

Many if not most of the described operations described in the documentation will require root access. On most Linux platforms a root command prompt may be obtained by running the command “su” followed by entering the root password. On Ubuntu platforms, instead of the “su” command, use the “sudo su” followed by entering a root password instead. For the purposes of more easily following the documentation, please run the commands described in this document with root access.

2.0 Automatic Installation of the driver

If all of the preliminary steps and libraries have been setup, there is an “install.sh” file provided with the driver bundle that will automatically install the driver on most systems. Run this installation script with root access from the installation directory using the command “sh ./install.sh”. If there is an error at any point in the installation script, please see the documentation section relevant to the installation failure under “Manual Installation of the driver”.

```
Command:
sh ./install.sh

Example output:

Note: If errors are found at any point in the script, please see online
documentation at www.microchip.com on how to setup target linux target
build dependencies to resolve these errors.

Building kernel source

make -C /lib/modules/2.6.35-24-generic-pae/build
SUBDIRS=/home/steve/Desktop/TSHARC-LINUX-UART-V102 modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.35-24-generic-
pae'

.
.
.
Installation complete.

To calibrate controller, please see Microchip online documentation for
details regarding this.
```

Please see the section below for information on how to automatically activate driver on reboot.

3.0 Manual Installation of the driver

3.1 Setting up UART controller on system

Getting touch communication to work over a UART interface involves two main parts. The first part is creating (compiling) a kernel touchscreen module that will take a set of bytes read from the UART interface and decoding these bytes into coordinates based on the touch protocol. The second part is creating an application that will register an open handle of a serial port with the kernel so that that UART driver may see the UART data stream such that it may be decoded and translated into coordinates.

3.1.1 Compiling driver for current kernel version

Before we can create the necessary kernel touch module, we will need to setup our development system and/or Linux target so that we may be able to successfully compile the provided Microchip kernel touchscreen module code. The way this is done is different depending on the Linux distribution used.

If the compile and installation is successful, you should see output similar to the following:

```
Command:
make

Example output:
make -C /lib/modules/2.6.35-24-generic-pae/build
SUBDIRS=/home/steve/Desktop/TSHARC-LINUX-UART-V102 modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.35-24-generic-
pae'
  CC [M]  /home/steve/Desktop/TSHARC-LINUX-UART-V102/mchptsharc.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/steve/Desktop/TSHARC-LINUX-UART-V102/mchptsharc.mod.o
  LD [M]  /home/steve/Desktop/TSHARC-LINUX-UART-V102/mchptsharc.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.35-24-generic-
pae'
gcc -o inputactivate inputactivate.c
gcc -o inputverify inputverify.c

Command:
sh ./install-module.sh

Example output:
Installing kernel module
`mchptsharc.ko' -> `/lib/modules/2.6.35-24-generic-
pae/kernel/drivers/input/touchscreen/mchptsharc.ko'
```

A kernel touch module will have been created in the same directory it was built with a ".ko" file extension after a successful build using the "make" command. The "./install-module.sh" script's responsibility is to install the compiled touch module by copying it to a location where it may be successfully loaded.

It is often necessary (but not always necessary since a kernel module may automatically load from activation) to load the kernel module before activating the processing of touch packets. To load the kernel module, enter the command “modprobe mchptsharc”.

```
Command:
modprobe mchptsharc

Example output:
No output expected
```

To verify the kernel module has successfully loaded, run the command “lsmod | grep mchp” and a single line of text should appear indicating the loaded module.

```
Command:
lsmod | grep mchptsharc

Example output:
Mchptsharc     2321    0
```

Please note that after this step the cursor is not yet expected to move. Even though the kernel touch module has been loaded, we will still need to send the kernel module bytes before we may have coordinates returned using the device path.

3.1.2 Setting up a udev rule

For every input device that is activated (enumerated), a kernel evdev path will be created in the order of enumeration somewhere under the path “/dev/input” that will send touch state information in evdev format to any application or touch library that reads this location (for example, “/dev/input/event5”). Sometimes after a reboot or a hotplug, this evdev path will change. This change can cause our setting to not be applied to the intended input device. Fortunately, Linux provides a mechanism in the form of udev rules to consistently map this location. To install the udev rule for the controller, copy the file “10-tsharcuart.rules” to the “/etc/udev/rules.d” directory. The following line is an example of how to do this.

```
Command:
cp 10-tsharcuart.rules /etc/udev/rules.d

Example output:
No output expected
```

If a recent kernel with “inotify” support is used, udev will automatically detect this new file and apply this new rule file. If this is not the case, reboot to computer to apply the rule file.

Now all controllers that are activated with the name “Microchip TSHARC Serial TouchScreen” will be mapped at location “/dev/input/by-id/Microchip_TSHARC_UART_Touchscreen_event”. Verification of

this udev rule being applied is described under “Activating UART Driver and determining event ID” in the upcoming documentation.

3.1.3 Finding the touch controller’s device path

A controller’s device path is the path in which data may be read from the touch controller. In order to finish configuring a touch controller, this path needs to be discovered if it is not known already (Usually, the path is either "/dev/ttyS0" or "/dev/ttyS1"). To help automatically discover the controller, the “inputverify” tool that is included with the driver may be used to discover the path of the controller. To detect the device path, run the command “./inputverify –f –p TSHARC” from the installation directory. With the exception of the EVDEV protocol, during the execution of this command the touch screen will need to be touched (to better clarify, touch screen, press enter, release touch after command completes).

```
Command:
./inputverify –f –p TSHARC

Example output:
Microchip Touchscreen Controller Utility V1.00

Command-line options selected:
Find mode enabled.
Protocol set to TSHARC
Found controller at device path: /dev/ttyS0
```

3.1.4 Activating UART Driver and determining event ID

1. If a udev rule was installed, this step may be skipped. Otherwise, before activating the driver, if udev rule has not been installed, first look at the list of existing event IDs. This can be done by running the command “ls /dev/input”

2. If the udev is Using the device path of the controller, run the "inputactivate" command with root access as shown below:

```
Command:
./inputactivate –tsharc /dev/ttyS0 &

Example output:
No output expected
```

After touching the touch screen, the cursor will now move if running this command under the Desktop (X11) Linux environment.

3. If a udev rule was installed, verify the udev mapped evdev path appeared by running the command “ls /dev/input/mctouchscreen”.

```
Command:
ls /dev/input/by-id/Microchip_TSHARC_UART_Touchscreen_event

Example output:
/dev/input/by-id/Microchip_TSHARC_UART_Touchscreen_event
```

4. If a udev rule was installed, this step may be skipped. Otherwise, at this point the Linux system will now be outputting Linux formatted touch packets (evdev format). Enter the command "ls /dev/input" and the event id that was not previously seen in the directory listing will be the new event ID. Usually, the highest event ID is the correct ID.

The full series of steps to determine event id without a udev rule installed is shown below.

```
Command:
ls /dev/input

Example output:
by-id   event0  event2  event4  js0     mice    mouse1
by-path event1  event3  event5  js1     mouse0  mouse2

Command:
./inputactivate –tsharc /dev/ttyS0 &

Example output:
[1] 2252

Command:
ls /dev/input

Example output:
by-id   event0  event2  event4  event6  js1     mouse0  mouse2
by-path event1  event3  js0     mice    mouse1  mouse3
```

In the above example, the new event id is "event6".

3.1.5 Automatic activation after a reboot

To have the driver automatically activate after a reboot, a startup script can be created to automatically activate the controller for every startup. Usually, this file is created in the directory "/etc/init.d". Please create the file "touchstart" with the following contents with the bolded path change to the path on the target system..

```
modprobe mchptsharc
killall -9 inputactivate
/usr/local/bin/inputactivate –tsharc /dev/ttyS0 &
```

To cause the script to automatically run during bootup, we need to run the following commands:

```
Command:
runlevel

Example output:
N 2

Command (the following command may need to be modified slightly
depending on target distribution):
ln –s /etc/init.d/touchstart /etc/rc2.d/S01touchstart

Command:
chmod 755 /etc/init.d/touchstart

Example output:
No output expected
```

3.1.6 Setting up automatic installation on kernel upgrade / update using DKMS

Very often an updated kernel is installed on Linux system to address security fixes and add enhancements. All of the Linux kernel module drivers must be rebuilt at this point in order to work. In other words, if Linux modules were compiled before the update, it would cease to work after the upgrade (by design). In order to address this problem a mechanism call dynamic kernel module support (DKMS) can be used to automatically compile and install all kernel modules that are not already part of the Linux kernel tree.

A short summary of example dkms commands is shown below. In order for any of these commands to work correctly, the kernel source files, Makefile and "dkms.conf" files need to be present in the "TSHARC-LINUX-UART-V102" directory.

Registering a kernel source package with DKMS:
dkms add -m TSHARC-LINUX-UART -v V102

Building a kernel source package registered with DKMS:
dkms build -m TSHARC-LINUX-UART -v V102

Installing a kernel source package registered with DKMS:
dkms install -m TSHARC-LINUX-UART -v V102

Uninstalling a kernel source package registered with DKMS:
dkms uninstall -m TSHARC-LINUX-UART -v V102

Un-registering a kernel source package with DKMS:
dkms remove -m TSHARC-LINUX-UART -v V102 --all

4.0 Common Open Source Calibration Methods

Up to this point, we have been setting things up such that the system reads the controller data and decodes this into controller's raw coordinates. Raw coordinates are coordinates that the controller calculates and communicates as a result of a touch. These raw coordinates will be accurate according to its electrical connections from the touch screen to the controller. However, to make these raw coordinates match up with the touched area of the display, a touch calibration will often be necessary.

For embedded machines, the basic usage of the "tslib" touch screen library will be described. For desktop machines, the evdev X11 module (used by xorg.conf modification) and evdev dynamic calibration methods will be described.

4.1 Tslib (recommended reading for embedded Linux)

4.1.1 Setting up Tslib library

The Tslib library is an open source touch library mostly aimed at embedded systems. The Tslib libraries come with a calibration application ("ts_calibrate") and a test application ("ts_test") that utilize the graphics framebuffer. The framebuffer is a device that abstracts the graphics hardware so that software may access graphics capabilities using a common interface across various hardware platforms. The framebuffer interface is very common on embedded devices.

A Tslib installation script, the Tslib version 1.0 source bundle, and patch files are provided to ensure the code functions correctly with Microchip touchscreen controllers. Before running this startup script, please ensure the "autotools" packages are installed such as "autoconf" and "automake" that this script depends upon. An example usage of this script is shown below.

```
Command:
sh ./install-tslib.sh

Example output:
Extracting files
tslib-1.0/
tslib-1.0/m4/
tslib-1.0/m4/external/
tslib-1.0/m4/external/PLACEHOLDER
tslib-1.0/m4/internal/
tslib-1.0/m4/internal/visibility.m4
tslib-1.0/plugins/
.
.
.
make[2]: Leaving directory `/home/steve/Desktop/TSHARC-LINUX-UART-
V102/tslib-1.0'
```

```
make[1]: Leaving directory `/home/steve/Desktop/TSHARC-LINUX-UART-
V102/tslib-1.0'
Installation complete.

Next environmental variables will need to be setup in order to
run 'ts_calibrate' and 'ts_test' tslib utilities.

For example if evdev path is at /dev/input/event5,the minimum
environmental variables will need to be set by using the following
commands.

export TSLIB_TSDEVICE=/dev/input/event5
export TSLIB_PLUGINDIR=/usr/local/lib/ts
```

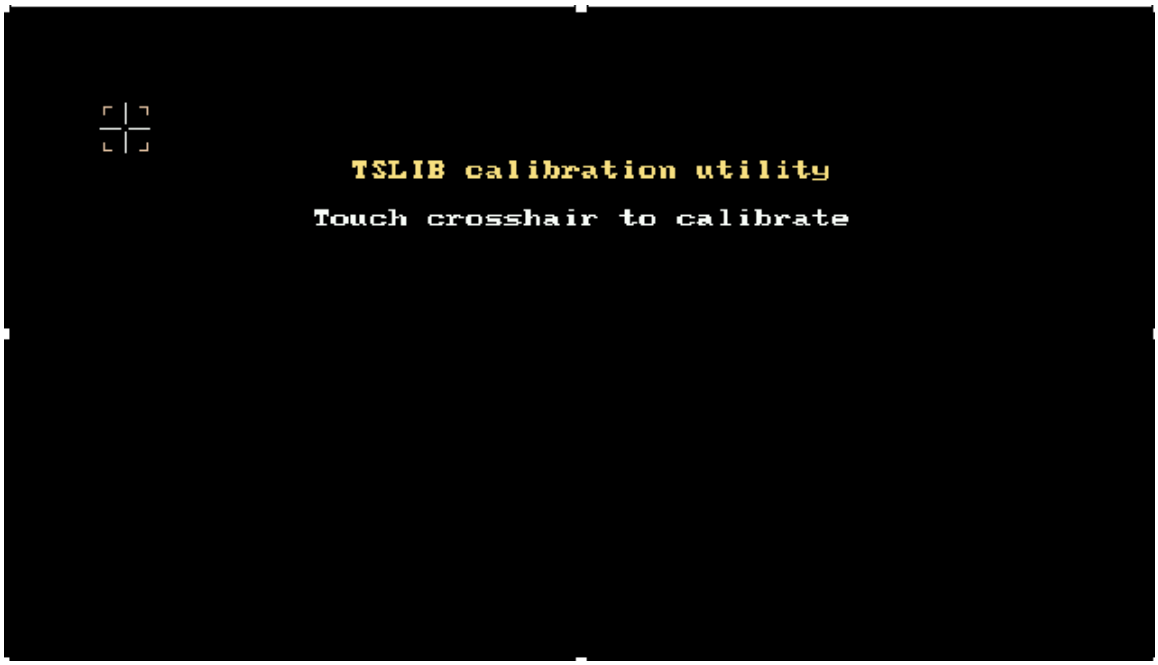
If everything is setup correctly, the “ts_calibrate” application can be used to calibrate the display and the “ts_test” application can be used to test the calibration. However, before we can use these utilities, we need to setup a couple environmental variables. At a minimum, we need to set “TSLIB_TSDEVICE” and “TSLIB_PLUGINDIR” for the utilities to work correctly with our Tslib configuration.

export TSLIB_TSDEVICE=/dev/input/by-id/Microchip_TSHARC_UART_Touchscreen_event
export TSLIB_PLUGINDIR=/usr/local/lib/ts

Other environmental values that can be set include the following:

export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_CONSOLEDEVICE=none
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_CONFFILE=/etc/ts.conf

After setting the environmental values, we now can run “ts_calibrate”. The images below shows what can be expected to see after running this calibration.



To test the calibration, run the “ts_test” command. The image below demonstrates the test screen.

4.2 Configuring X-Windows (recommended reading for desktop Linux)

X11(also referred to as X-Windows or Xorg) is a protocol that is used for graphical applications and input most often used on desktop Linux systems. Windows managers on Linux use X11 to define a general interface to interacting with graphical application. In this section we will describe how to configure our input devices with X11 so that the resulting coordinate events returned by X11 are in a calibrated form.

4.2.1 Acquiring minimum/maximum values of X/Y axes

The "inputverify" tool may be used to monitor input events from a device path.

We will want to use the "inputverify" tool with the activated controller's evdev path to get the minimum and maximum values for the X and Y axises of the touch screen so we will later be able to calibrate the touch sensor.

```
inputverify -d /dev/input/by-id/Microchip_TSHARC_UART_Touchscreen_event
-p EVDEV
```

Test if X and Y axes are swapped

Touch along the left edge of the touch screen and look at the second column of numbers which represents the y-axis. The values in these middle column should have the biggest change in value and the left column should be mostly near the same value. If this is not the case, that means the touch screen is wired to the controller such that the x and y axes are swapped.

To test for range with axes swapped, please follow these steps:

Touch the middle of the left edge of the touchscreen and write down the Y position as the min_y value.
Touch the middle of the right edge of the touchscreen and write down the Y position as the max_y value.
Touch the middle of the top edge of the touchscreen and write down the X position of the min_x value.
Touch the middle of the bottom edge of the touchscreen and write down the X position of the max_x value.

To test for range without axes swapped, please follow these steps:

Touch the middle of the left edge of the touchscreen and write down the X position as the min_x value.
Touch the middle of the right edge of the touchscreen and write down the X position as the max_x value.
Touch the middle of the top edge of the touchscreen and write down the Y position of the min_y value.
Touch the middle of the bottom edge of the touchscreen and write down the Y position of the max_y value.

We will use these values for our calibration using either the evdev X11 module or evdev dynamic calibration. If the minimum x value is greater than the maximum x value and/or the minimum y value is greater than the maximum y value, this is not uncommon and evdev will use this information to determine which axes are flipped.

4.2.2 Evdev X11 dynamic calibration

Newer version of Xorg includes the "Xinput" protocol which allows for dynamic calibration. Dynamic calibration enables a touch screen calibration to be tested without first editing a configuration file.

The minimum requirements for evdev dynamic calibration are as follows:

- Xserver version 1.5 and above
- evdev version 2.3.0 or higher

To perform a dynamic calibration manually, please follow these steps.

1. Find the ID of the touch device to configure by running the command “xinput list”.

2. Set the raw mode of the device every time before a dynamic calibration is performed (otherwise we may be calibrating an already calibrated touch device) using the following command:

```
xinput set-int-prop <id> “Evdev Axis Calibration” 32 0 4095 0 4095
xinput set-int-prop <id> “Evdev Axes Swap” 8 0

Example output:
No output expected
```

3. To set the dynamic calibration values, run the following command using the value for the previous step.

```
xinput set-int-prop <id> “Evdev Axis Calibration” 32 min_x max_x min_y
max_y
```

4. If the X and Y axises were determined to be swapped, we will need to run this additional command.

```
xinput set-int-prop <id> “Evdev Axes Swap” 8 1
```

5. To verify the settings have been set correctly, touch the touch screen and the mouse cursor should move to the touched area.

4.2.2.1 Making dynamic calibration settings permanent

1. Create a “99-calibration.conf” file in the home directory with the following contents

```
Section "InputClass"
        Identifier "calibration"
        MatchProduct "Microchip TSHARC Serial TouchScreen"
        Option "Calibration" "min_x max_x min_y max_y"
EndSection
```

2. If the X and Y axises were determined to be swapped, we will alter the calibration option line shown above.

```
        Option "SwapAxes" "1"
```

3. Copy this file to the "xorg.conf.d" directory.

```
Command:
find / -name "xorg.conf.d"

Example output:
/usr/share/X11/xorg.conf.d

Command using above example:
cp 99-calibration.conf /usr/share/X11/xorg.conf.d

Example output:
No output expected
```

4. Reboot computer.

4.2.3 X11 module files

X11 module files can be used to configure how various devices interface with the graphical environment and the driver the device is to be associated with. The file is that usually responsible (if a file is present) is the file "/etc/X11/xorg.conf". We will use this file to configure input devices to send mouse events such that the cursor is calibrated to the touch screen. Before setting up the system using an X11, please verify that the steps detailed in section 3.1.5 has been completed so that a usable evdev path is available after a reboot.

4.2.4 Create missing "xorg.conf" file

Most devices on X11 are automatically used and configured. For this reason, the configuration file "/etc/X11/xorg.conf" is often missing by default. In the case of configuring touch calibration, we will need to generate a configuration file so we may set the calibration setting appropriately.

If a "xorg.conf" file does not exist, one can be created using the command "X -configure" with root access without xorg running. To apply the detected settings, run the command "cp /root/xorg.conf.new /etc/X11/xorg.conf". It is recommended to test these detected settings before proceeding.

In our example distribution Ubuntu 10.10, we create this missing "xorg.conf" file using the following steps.

1. Reboot the system to get to the grub menu.
2. If the grub menu does not appear by default during a boot, hold the left shift button shortly after reboot and the grub menu will appear.
3. Selected the appropriate menu item and press the "e" key (usually the default selection is correct)

4. Move the cursor down to the line starting with the word “linux” and add the parameter “single” to the end of the line.
5. Boot using this modified grub entry.
6. Enter the command “X -configure” and then copy the generated “xorg.conf.new” to the “/etc/X11” directory with the file name “xorg.conf”.
7. Reboot

The following demonstrates these commands:

```
Command:
X –configure

Example output:
X.Org X Server 1.9.0
Release Date: 2010-08-20
X Protocol Version 11, Revision 0
Build Operating System: Linux 2.6.24-28-server i686 Ubuntu
Current Operating System: Linux microchip-SK21V10 2.6.35-25-generic
#44-Ubuntu SMP Fri Jan 21 17:40:48 UTC 2011 i686
Kernel command line: BOOT_IMAGE=/boot/vmlinuz-2.6.35-25-generic
root=UUID=b738431a-45cd-4161-bd4e-97f1476768ba ro quiet splash single
Build Date: 09 January 2011  12:14:58PM
xorg-server 2:1.9.0-0ubuntu7.3 (For technical support please see
http://www.ubuntu.com/support)
Current version of pixman: 0.18.4
        Before reporting problems, check http://wiki.x.org
        to make sure that you have the latest version.
Markers: (--) probed, (**) from config file, (==) default setting,
        (++) from command line, (!!) notice, (II) informational,
        (WW) warning, (EE) error, (NI) not implemented, (??) unknown.

Command:
cp xorg.conf.new /etc/X11/xorg.conf

Example output:
No output expected
```

4.2.5 Evdev X11 module

To using the evdev X11 module, please follow these steps:

5. Edit the “/etc/X11/xorg.conf” file such that it includes the following lines at the end of the file.

```
Section "InputDevice"
        Identifier "evdev”
```

```
        Driver “evdev”
        Option “Calibration” “min_x max_x min_y max_y”
        Option "Device" "/dev/input/by-id/Microchip_TSHARC_UART_Touchscreen_event”
EndSection
```

6. Next, edit the same file so that it includes the following text below after the line: Section “ServerLayout”

```
        InputDevice “evdev”
```

7. To calibrate the touchscreen, replace the “min_x”, “min_y”, “max_x”, “max_y”, with the values discovered in the “Acquiring minimum/maximum values of X/Y axes” section above.

8. If the X and Y axises were determined to be swapped, we will after the calibration option line shown above.

```
        Option “SwapAxes” “1”
```

9. Reboot computer.

The new touch calibration will now be applied and the cursor will move to the touched region of the touch screen.

5.0 Uninstallation of the driver

To uninstall driver, run this uninstall script with root access from the installation directory using the command “sh ./uninstall.sh”.

```
Command:
sh ./uninstall.sh

Example output:
Deactivating Module
Unloading Module
Removing Utilities
removed `/usr/local/bin/inputactivate'
removed `/usr/local/bin/inputverify'
Removing udev rule
removed `/etc/udev/rules.d/10-tsharcuart.rules'
Removing DKMS components
.
.
.
```

```
Uninstall Complete.
```

6.0 Troubleshooting

6.1 "inputverify" utility freezes while searching for UART device path

The find mechasm for UART path makes the assumption that the hardware timeouts are supported on the target hardware (which is usually the case). However, if the hardware timeout cannot be set, the utility may hang while looking for the attached port. If this is the case, please check for the path by manually testing each logical path the controller may be located at such "/dev/ttyS0", "/dev/ttyS1", "/dev/ttyS2" and so on by using a command similar to the following "./inputverify –d /dev/ttyS0 –p TSHARC".

6.2 Error opening "fbdevice" using Tslib utilities on desktop Linux

While Tslib is mostly geared towards embedded uses, it can also be used on the desktop as well. Sometimes though, when running "ts_calibrate" or "ts_test" utilizes an error regarding opening "fbdevice", is displayed. This error can usually be corrected by adding an additional kernel mode parameter "vga=xxx" (see table below for value of xxx) by editing the kernel parameters within the grub menu.

1. Reboot the system to get to the grub menu.
2. If the grub menu does not appear by default during a boot, hold the left shift button shortly after reboot and the grub menu will appear.
3. Selected the appropriate menu item and press the "e" key (usually the default selection is correct)
4. Move the cursor down to the line starting with the word "linux" and add a parameter such as "vga=771" to the end of the line to setup framebuffer graphics for 800x600 display resolution.
5. To avoid any display drive conflicts, it may be necessary to run the tslib utilities without Xorg running.

By adding this parameter, this enables the use of the kernel framebuffer interface so the tslib utilities may display graphics correctly. To use a resolution other than 800x600, please refer to table below which discusses a subset of the other available framebuffer display resolutions.

Depth	800x600	1024x768	1280x1024
8 bit	vga=771	vga=773	vga=775
16 bit	vga=788	vga=791	vga=794
24 bit	vga=789	vga=792	vga=795