TrueTouch® Driver for Linux (TTDL) PIP User Guide

Doc. No. 001-90126 Rev. *I

CONFIDENTIAL - RELEASED ONLY UNDER NONDISCLOSURE AGREEMENT (NDA)

Parade Technologies, Inc.
2858 De La Cruz Blvd.
Santa Clara, CA 95050
Phone (USA): +1.408.329-5540
http://www.paradetech.com

Copyrights

Trademarks

PSoC Designer™, Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. TrueTouch® is a registered trademark of Parade Technologies, Ltd. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

The Android Driver Source Code is free and open source and distributed according to the GNU General Public License version 2 (GPL v2). A copy of this license can be included in the distribution files (COPYING.txt).

Source Code

Other Source Code (software and/or firmware) is owned by Parade Technologies, Ltd. (Parade) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Parade hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Parade Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Parade integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Parade.

Disclaimer

PARADE MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Parade reserves the right to make changes without further notice to the materials described herein. Parade does not assume any liability arising out of the application or use of any product or circuit described herein. Parade does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Parade's product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Parade against all charges.

Use may be limited by and subject to the applicable Parade software license agreement.

Contents

Section A: TTDL Technical Description 7

1. Overview 8
- 1.1 TrueTouch Device Support 8
- 1.2 Android Touchscreen Device Driver 10
- 1.3 Resources 11
  - 1.3.1 TrueTouch® Documentation 11
  - 1.3.2 Public Reference Material 11
  - 1.3.3 Definitions, Acronyms, and Abbreviations 12

2. Technical and System Description 13
- 2.1 Module Design 13
- 2.2 System Architecture 15
- 2.3 TrueTouch Devices and Interfaces 16
- 2.4 Multi-Touch Signal Types in Android 17
  - 2.4.1 Multi-Touch Protocol A (MTA) 17
  - 2.4.2 Multi-Touch Protocol B (MTB) 17

3. TTDL Kit Contents 18

4. TTDL Modules and Program Flow 21
- 4.1 Module Types 21
  - 4.1.1 Core Module 21
  - 4.1.2 Worker Modules 22
  - 4.1.3 Adapter Module 23
- 4.2 Module Build Methods 23
  - 4.2.1 Built-In 23
  - 4.2.2 Loadable Kernel Module (LKM) 23
  - 4.2.3 Device 24
- 4.3 Driver Module Loading 24
  - 4.3.1 Automatically Loading Modules 24
  - 4.3.2 Manually Loading Modules 24
  - 4.3.3 Module Dependencies 25
- 4.4 Application Development Environment 25
  - 4.4.1 Characteristics 25
- 4.5 Program Flow 26
  - 4.5.1 Start Thread 26
  - 4.5.2 Interrupt Service Thread 27
  - 4.5.3 Device Access Flow 28
  - 4.5.4 Thread Flow 29

4.5.5 Callback Flow 30
4.5.6 Request Command Output 31
4.6 Error Handling 31
4.6.1 Dev_info() 31
4.6.2 Dev_err() 31
4.6.3 Dev_dbg() 31
4.6.4 Dev_vdg() 32
4.6.5 pt_debug() 32
4.6.6 Watchdog 32

Section B: Features and Modules 33

5. CapSense Button Keycode Signaling Module 34
5.1 Details 34
5.1.1 8-Button Support 34

6. Dynamic Debugging Module 35
6.1 Details 35

7. Easy Wakeup 36
7.1 Basic Easy Wakeup Details 36
7.2 Advanced Easy Wakeup Support 37
7.2.1 Gesture ID 38
7.2.2 Gesture Data 38

8. Firmware Upgrades (OTA Updates) 39
8.1 Using the Firmware Class for Device Firmware Loading 39
8.2 SWD Firmware Loader 39
8.3 Loading by Panel ID 40
8.3.1 Bin Files 40
8.3.2 Included Image Files 40

9. Manufacturing Tests 41
9.1 Basic Manufacturing Tests 41
9.1.1 Panel Scan 41
9.1.2 Get IDAC Data 42
9.1.3 Run Automatic Shorts Test 42
9.1.4 Run Opens Test 43
9.1.5 Run Calibration 43
9.1.6 Initialize Baselines Test 43
9.2 C_M/C_P In-system Manufacturing Tests 44
9.2.1 Mutual-capacitance (C_M) Panel Test 44
9.2.2 Self-capacitance (C_P) Panel Test 44
9.2.3 Mutual-capacitance (C_M) Buttons Test 45
9.2.4 Self-capacitance (C_P) Buttons Test 45
9.3 Range Checking C_M/C_P Manufacturing Test 46
9.3.1 Range Checking C_M/C_P Host Interface 46
9.3.2 Sample Input Range File 50
9.3.3 Sample Output Results File 52

10. Multi-Touch Signaling Module 53
10.1 Feature Details 53

11. Power Management 56
11.1 Linux Open/Close 56
11.2 Linux Notifier Power Management 56

12. Proximity Sensing Signaling Module 57
12.1 Overview 57
12.2 Proximity Details 57

13. Restore Parameters on Restart 59
13.1 Details 59

14. Stylus, Hover and Glove 60
14.1 Stylus Details 60
14.2 Hover Details 60
14.3 Glove Details 60

15. User Interfaces (sysfs, debugfs) 61
15.1 Details 61
15.1.1 Sysfs Node Path: /sys/kernel/debug 61
15.1.2 Sysfs Node Path: /sys/kernel/debug/x-0024/mfg_test 62
15.1.3 Sysfs Node Path: /sys/class/firmware/x-0024 62
15.1.4 Sysfs Node Path: /sys/bus/i2c/devices/x-0024 63
15.1.5 Sysfs Examples 67

16. Virtual Key Signaling 68
16.1 Virtual Key Overview 68
16.2 Virtual Key Map 68
16.3 Implementing Virtual Keys 69

Section C: Customization and Debugging 71

17. Driver Porting (Linux Board Configuration) 72
17.1 Overview 72
17.2 TTDL Porting Example 72

18. Driver Porting (Device Tree) 75
18.1 TTDL Porting 75

19. Firmware Updates 76
19.1 Remote Host 76
19.2 Automatic OTA Updates 76
19.2.1 Kernel Upgrade 76
19.2.2 ADB Push 77
19.2.3 OTA Upgrade With a .ihex File 77
19.2.4 Board Configuration File Change 77
19.2.5 Manual Touch Parameters Update Using Binary File (Driver.bin) 78
19.2.6 Create a New Kernel with *<project>.bin* File 82
19.2.7 Testing the OTA Feature Using the New Kernel 83
19.2.8 Summary Menuconfig System Generated Defines 84

20. IDAC Calibration 86
20.1 Testing With TTDL 86
20.1.1 Calibrate IDAC for Mutual Capacitance Sensors 86

20.1.2 Calibrate IDAC for Mutual Capacitance Buttons 86
20.1.3 Calibrate IDAC for Self Capacitance 87

21. Driver Without XRES Signal 88
21.1 Solution to Designs Without XRES Signal 88
21.2 Technical Notes 88

22. System Debugging 90
22.1 Parade's TrueTouch Host Emulator Tool 90
22.2 Logic Analyzer 90
22.3 Enable Linux Debug Macros 90
22.4 Helpful Linux Host Windows 90
22.4.1 logcat 90
22.4.2 getevent 90
22.4.3 kmsg 91
22.5 Android Display Settings 91

23. Mobile Tuner 92
23.1 Overview 92
23.2 Prerequisites 92
23.3 Install USB Driver for Target Android Device 93
23.4 Settings for Mobile Tuning via USB 95
23.4.1 TTHE 3.1 and Later 95
23.4.2 Finding the Driver Directory Paths 98
23.5 Settings for Mobile Tuning Via WiFi 99

24. Troubleshooting and FAQ 101

Section A: TTDL Technical Description

This section documents the technical design and features in the TrueTouch® Driver for Linux (TTDL) 3.x driver. Typically, the latest driver version that is released is recommended. However, the customer firmware may require a specific driver version (refer to the TrueTouch device release notes for compatibility requirements).

Section A contains the following chapters:

- Overview
- Technical and System Description
- TTDL Kit Contents
- TTDL Modules and Program Flow

1. Overview

Parade's TrueTouch® Driver for Linux™ (TTDL), formerly known as TrueTouch® Driver for Android™ (TTDA) enables developers to integrate Parade TrueTouch devices into mobile touchscreen applications. This driver targets Android implementations, but can be integrated into other Linux kernel-based products that support touchscreen input drivers also. The driver components are modular and can be integrated into products with either I^2C or SPI host to touchscreen device interfaces.

The code is developed under the open-source licensing terms of GPL v2.

1.1 TrueTouch Device Support

TTDL 3.x supports the following Packet Interface Protocol (PIP) TrueTouch devices (see 001-85948):

- TMA545/TMA54X Base v1/v2
- TMA568 Base v1/v2
- TMA445A Base v1 and Base.v1.1
- TMA448 Base v1/v2
- TT21XXX, TT31XXX Base.v1

TTDL 3.x provides support for the following functions:

- TrueTouch Packet Interface Protocol (PIP) host interface protocol
- 10 finger touch, passive stylus, glove, hover, proximity
- Bootloader (field upgrade)
- Up to 8 CapSense buttons
- Virtual keys
- In-system ("in-phone") manufacturing tests (TTDA 3.3 and above)
- In-system C_M/C_P manufacturing tests (TTDA 3.5 and above)
- Command and response via a standard file system interface (sysfs)
- EasyWake

Table 1-1. Driver-Supported Features

Driver Version	Product Family	Firmware Release	Linux Kernel (Compile Versions)	Driver-Supported Features												
				Bus Model	EZ Wake	Hover	Stylus	Glove	Proximity	Device Tree	Simplified Core Driver	In-System Tests	In-system C_M/C_P Tests	8 Button Support	Advanced EZ-Wake	C_M/C_P Range Checking
TTDA 3.0	TMA525	TSG5_M. Base.V1	3.0 - 3.6	✓	–	–	–	–	–	–	–	–	–	–	–	–
TTDA 3.0.1	TMA568	TSG5_L. Base.V1	3.2 - 3.6	✓	–	–	✓	-	-	✓	–	–	–	–	–	–
TTDA 3.1	TMA545	TSG5 Base.V1,2	3.2 - 3.6	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–
TTDA 3.2	TMA448 TMA5XX	TSG5 Base.V1,2	3.2 - 3.14	–	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–
TTDA 3.3 TTDA 3.4	TMA448 TMA5XX	TSG5 Base.V1,2	3.0 - 3.15	–	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–
TTDA 3.3 TTDA 3.4	TMA445A	TSG6M Base.V1	3.0 - 3.15	–	✓	–	–	✓	✓	✓	✓	✓	–	–	–	–
TTDA 3.5	TMA445A, TT21XXX, TT31XXX	TSG6M. Base.V1.1 TSG6L. Base.V1	3.0 - 3.15	–	✓	–	TSG6L	✓	✓	✓	✓	✓	✓	–	–	–
TTDL 3.6	TMA445A, TT21X, TT31X	TSG6M.Base.V 1.3 TSG6L.Base.V 1.3	3.0-3.15	–	✓	–	TSG6L	✓	✓	✓	✓	✓	✓	✓	✓	–
TTDL 3.7	TMA445A, TT21X, TT31X TT41X	TSG6M.Base.V 1.3 TSG6L.Base.V 1.3 TSG6XL.Base. V3	3.0-3.15	–	✓	–	TSG6L, 6XL	✓	✓	✓	✓	✓	✓	✓	✓	✓
TTDL3.8	TMA445A, TT21X, TT31X TT41X	TSG6M.Base.V 1.3 TSG6L.Base.V 1.3 TSG6XL.Base. V3	3.2-4.8.4	–	✓	–	TSG6L, 6XL	✓	✓	✓	✓	✓	✓	✓	✓	✓

For TTDA 3.0.1 to TTDA 3.2 versions and for kernel versions earlier than 3.2, the driver adapter modules must be modified to remove references to the Device Tree support kernel API calls. For kernel versions earlier than than 3.0, TTDL may be limited to using only the multi-touch protocol A module; otherwise, attempts to build for multi-touch protocol B may return an unresolved symbol “input_mt_init_slots”.

For TTDA 3.0 to TTDA 3.1 versions and for kernel versions later than 3.6, the multi-touch protocol B kernel API call to “input_mt_init_slots()” needs to have a third parameter added. This parameter can either be set to 0 or the kernel-defined constant INPUT_MT_DIRECT can be used. Additionally, the use of the macro “__dev_exit_p” must be removed in the adapter modules.

Note: Latest versions of the driver have more bug fixes.

1.2 Android Touchscreen Device Driver

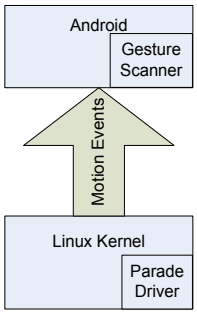
Android is a complete ecosystem designed specifically for mobile devices. This system includes a Java®-based user interface for which developers can create applications that can be installed on user products. Android uses a specialized version of the Linux operating system kernel. The Parade Android driver is included in the product Android platform build as part of the boot image, which contains the runtime kernel. TTDL 3.x is compatible with Android 2.3 (Kernel 2.36), 4.0 (Kernel 3.0.8), and 4.1+ (Kernel 3.2+).

The Linux kernel supports several driver types. The input subsystem type has separate classifications for input devices. TTDL 3.x is classified as a “Touch Device” in the input subsystem.

A touchscreen device driver services touchscreen device touch reports, parses the touch signals, packs a Linux event with the touch signals as motion signals, and then submits the packed event to the Linux event-handling system. This event-handling system passes the event information to the Android platform layer, which passes the event to requesting applications as a Motion Event.

The Android also includes a Gesture Scanner. The Motion Event information is sent to the Gesture Scanner, which can create a number of common gestures. Android applications that request gesture signals receive the gesture information from the Android Gesture Scanner.

Figure 1-1. Driver Overview



TTDL 3.x is based on the Linux bus driver and uses Linux module-based functions to provide the hardware bus and platform system interfaces. TTDL custom user modules can subscribe for data from the core functions, allowing you to create innovative designs quickly. Initially, the TTDL development used the Linux multi-touch protocol A signaling to support Android Gingerbread (GB 2.3) development platforms. Protocol B signaling was added to support the integration and testing on Android Ice Cream Sandwich (ICS 4.0+) development systems. Slot signaling associates each tracked (touch) ID with a slot number, making TTDL suitable for submission to the Linux repository for upstream releases into the global Linux community. TTDL reference platforms are the Texas Instruments (TI) PandaBoard hardware and the Qualcomm Snapdragon Kit.

1.3 Resources

1.3.1 TrueTouch® Documentation

Document Number	Document Title	Description
001-88715	TMA545 TrueTouch® Multi-Touch All-Points Touchscreen Controller	The device datasheets contain information on features, system design, touchscreen performance, electrical specifications, and packaging.
001-82145	TMA568 TrueTouch® Multi-Touch All-Points Touchscreen Controller	
001-89993	TMA448 TrueTouch® Multi-Touch All-Points Touchscreen Controller	
001-87143	TMA445A TrueTouch® Multi-Touch All-Points Touchscreen Controller	
001-93969	TT21X/31X TrueTouch® Multi-Touch All-Points Touchscreen Controller	
001-96852	TT41X TrueTouch® Multi-Touch All-Points Touchscreen Controller	
001-80396	TMA545 Technical Reference Manual (TRM)	Technical Reference Manuals contain register map details and device operational theory.
001-87666	TMA568 Technical Reference Manual (TRM)	
001-89995	TMA448 Technical Reference Manual (TRM)	
001-88195	TMA445A/TT21X/TT31X/TT41X Technical Reference Manual	
001-85948	TrueTouch® Communication Examples – Packet Interface Protocol (PIP)	Examples of how to use the built-in self-test interface on TrueTouch PIP devices.
001-63571	MTE User Guide	TrueTouch® Manufacturing Test Executive User Guide.
001-90764	Serial Wire Debug (SWD) Programming with TTDA User Guide	Includes prerequisites for the SWD firmware loader feature and provides guidance on usage.

1.3.2 Public Reference Material

Reference	Description
http://en.wikipedia.org/wiki/Fastboot	Fastboot standard
http://developer.android.com/guide/developing/tools/adb.html	Android debugging host tool
https://source.android.com/devices/input/touch-devices.html	Android standards for touch devices
http://pandaboard.org	TI PandaBoard information
http://www.gnu.org/licenses/gpl-2.0.html	Open-source licensing used by the TTDL
LINUX Files in Build Tree	
kernel/Documentation/input/multi-touch-protocol.txt	Linux multi-touch protocol description
kernel/include/linux/input.h	Linux event signal definitions: Defines all multi-touch event signals used by the driver
kernel/Documentation/firmware_class/README	Text file containing the firmware class use design

1.3.3 Definitions, Acronyms, and Abbreviations

Reference	Description
GPL	GNU General Public License
HID	Human Interface Device
I^2C	Inter-Integrated Circuit
NDA	Non-Disclosure Agreement
OTA	Over-the-Air (automatic firmware load at startup)
PIP	Packet Interface Protocol
SPI	Serial Peripheral Interface
TTDA	TrueTouch Driver for Android
TTDL	TrueTouch Driver for Linux
TTHE	TrueTouch Host Emulator
TTSP	TrueTouch Standard Product
Upstream	Refers to the Linux repository found at http://www.kernel.org

2. Technical and System Description

2.1 Module Design

TTDA/TTDL uses Linux modules for interfaces. These modules can be added or removed at runtime:

- TrueTouch Standard Product (TTSP) Bus Handler
  - Linux bus type; uses Linux-defined bus structure
  - Responsible for coordination of other modules
  - Creates a virtual bus that is represented to the Linux kernel
  - All other modules use the APIs provided by the bus module to register themselves
- Core Module
  - TTSP bus-defined core device structure (core device)
  - TTSP bus-defined core driver structure (core driver)
  - Responsible for interacting with the physical device (in terms of its host interface)
  - Needs helper functions provided by the core platform data to perform board-specific tasks, such as initialization, power-on and power-off, suspend and resume, and interrupt handling
  - Provides APIs to upper modules to access the physical device
- Adapter (I^2C/SPI) Modules
  - I^2C adapter device and driver
  - SPI adapter device and driver
  - TTSP bus-defined adapter structure (adapter device)
  - Linux host bus-defined driver structure
  - Responsible for interacting with the physical device (in terms of its bus interface)
  - Implement bus-specific read and write operations
- Multi-Touch Modules
  - TTSP bus-defined device structure (TTSP device)
  - TTSP bus-defined driver structure (TTSP driver)
  - Responsible for reporting touch events to Linux input subsystem
  - Two multi-touch modules for Linux's multi-touch protocol A (MTA) and B (MTB)
    - MTA module provides multi-touch signaling to Android using Protocol A
    - MTB module provides multi-touch signaling to Android using Protocol B

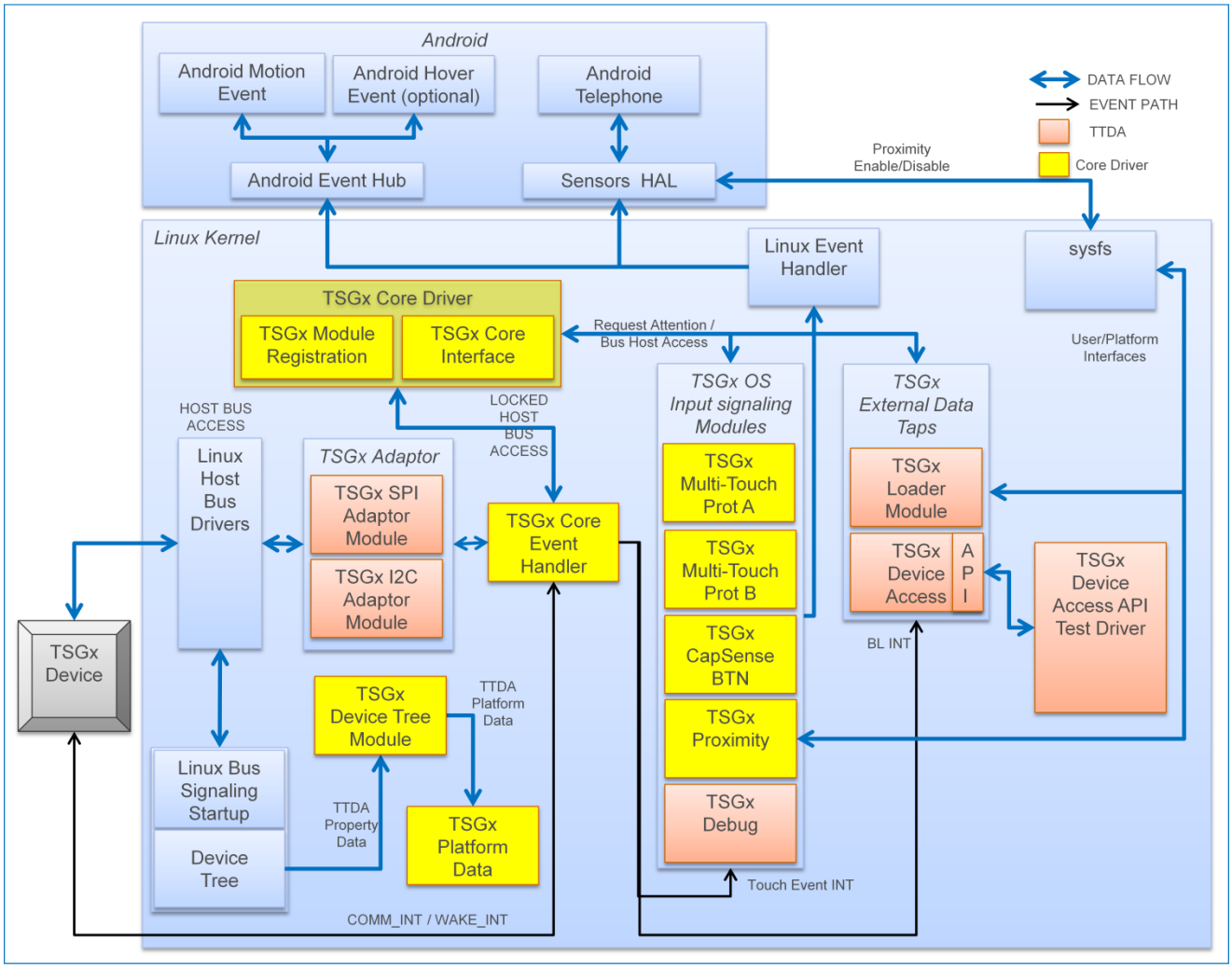
  Note: Do not run the MTA and MTB modules at the same time. Select and build with the one that the product's Android version supports. See Multi-Touch Signal Types in Android for details.

- CapSense Button Module
  - Responsible for reporting button events to the Linux input subsystem
- Debug Module
  - Print in human- or machine-readable format using a standard file system (sysfs) switch
  - TTHE interface support
  - Provides kernel log messages on touch and button events for debugging information
- Device Access Module
  - Includes TTHE interface
  - Provides access to gather information (SysInfo) about the physical device
  - Provides access to execute TrueTouch Application commands
- Loader (Firmware Upgrade) Module
  - Responsible for the upgrade of the touch application
  - Manual ADB loader
    - Uses firmware class with manual concatenate file to loader firmware class sysfs
  - Automatic Over-the-Air (OTA) loader
    - Optional method to allow automatic OTA at bootup
    - Runs once if a new firmware image has a greater revision than that currently in the TrueTouch device
    - Uses firmware-class with an optionally built-in firmware file (as a *.h file) or firmware binary file added in to the loader firmware class sysfs (as a *.bin file)
- Device Access API Test Module (TTDA 3.1 and later only)
  - Provides sysfs interface to emulate the normal Device Access sysfs interface
  - Uses the Device Access API to send the sysfs inputs to the Device Access and return Device Access responses to the emulation interface back to the user.
- Proximity Module (TTDA 3.1 to TTDL 3.7 only)
  - Compatible with TMA488/5XX Base.V2 firmware
  - Provides proximity NEAR/FAR reporting
- Device Tree Module (TTDA 3.0.1 and later)
  - Provides dynamic mapping of platform data for each module requiring such information and requiring TTDA bus device registration at startup.
  - Provides an API for access to, and parsing of, a Device Tree for the TTDL. The API is called by each Adapter Module.
  - Although this module has init and exit module functions, there is no probe function.

  Note: The Device Tree module must be built-in.

2.2 System Architecture

Figure 2-1. TTDL 3.2 – 3.7, TTDL3.8 System Data Flow



2.3 TrueTouch Devices and Interfaces

TTDL 3.X is developed using a TrueTouch TMA488/5XX/6XX development kit interfaced to TI PandaBoards or Qualcomm DragonBoards. The device register map and operational theory are available in the TrueTouch Technical Reference Manual documents.

TTDL 3.X supports PIP, which exposes the following:

- Touch reports
  - Touch ID: range [0..9]
  - X: range [0..xmax] (xmax read from system information)
  - Y: range [0..ymax] (ymax read from system information)
  - Z: range [0..255]
  - Touch major: range [0..255]
  - Touch minor: range [0..255]
  - Orientation: range [-128..127]
  - Hover: optional
  - Event ID: check for touch lift-off; do not signal lift-off touches
- Button reports - expanded to 8-button support in TTDL 3.6
- Operational interface for touch and CapSense button support
  - Touch report data will be converted to Linux multi-touch event signals.
  - Button report data will be converted to key codes.
  - Operation commands
  - A TTDL device user module provides a user interface to send operational commands and receive status via a standard file system (sysfs) interface.
  - The user can send (STORE) operational commands on the product command line interface using “echo” commands and receive the returned operational command status data (SHOW) using the “cat” command.
- Bootloader
  - In-field firmware upgrade using TTDL 3.x.
- Manufacturing test support
- Virtual key example code
  - Demonstrates how to set up a product baseline to assign touch area information to the Android platform so that it will automatically replace touch information in the defined key area with key codes.
  - Provides key code assignment to touch-geometry.

2.4 Multi-Touch Signal Types in Android

2.4.1 Multi-Touch Protocol A (MTA)

These signals are used in Android 2.0 and later and Android Éclair including revision 2.1. Protocol A signals include X and Y positions, and Z pressure for multiple touches. The Android Motion Event generator creates track information for each event report from the driver. Therefore, no hardware track ID information is included in the motion data. The X and Y position information is reported as absolute positions.

2.4.2 Multi-Touch Protocol B (MTB)

These signals are used in Android 4.0 Ice Cream Sandwich and later. Protocol B signals include the same X and Y positions and Z pressure as Protocol A signals; however, a hardware-generated track ID is also included in the interface. The X and Y position information is reported as absolute positions. Protocol B is required if your application is to be submitted to Linux.

3. TTDL Kit Contents

The TTDL kit includes the following:

- TTDL User Guide (this document)
- Release notes for the most current revision of the TTDL kit
- Source files
  - The “kernel” directory is the root of a tree structure of the files supplied by Parade to build the driver, mirroring the destination in Linux
  - Table 3-1 and Table 3-1 list the files included in TTDL 3.x. The board configuration file provided is for the TI PandaBoard reference platform, which is used for TTDL development. TTDL users must customize the board configuration file for specific custom design. See the Driver Porting section for more details.
- COPYING.txt
  - This is the required open source “GNU GENERAL PUBLIC LICENSE”. If you are not familiar with this license, read it completely. Parade can supply this file in accordance with the license.

Table 3-1. TTDA 3.2 and Above File List

File Name and Location	Description
..kernel/arch/arm/mach-omap2/board-omap4panda.c	Product specific configuration with Parade driver PandaBoard platform.
..kernel/arch/arm/boot/dts/omap4-panda.dts	Example Device Tree file that contains hardware configuration for the Parade driver PandaBoard platform. (TTDA 3.0.1 and later).
..kernel/arch/arm/boot/dts/apq8074-dragonboard.dtsi	Example Device Tree file that contains hardware configuration for the Parade driver MSM8074 Dragon Board platform. (TTDA 3.4).
..kernel/arch/arm/configs/msm8974_defconfig	Sample default configuration file for the MSM8074 Dragon Board platform. This file is modified to include the TTDL driver. (TTDA 3.4).
..kernel/drivers/input/touchscreen/cyttsp5_btn.c	Provides CapSense button Linux signaling. The buttons typically map to the Linux HOME, MENU, BACK, and SEARCH keys.
..kernel/drivers/input/touchscreen/cyttsp5_core.c	Provides device startup, interrupt service support, device data extraction, output command management, and device suspend and resume threads.
..kernel/drivers/input/touchscreen/cyttsp5_debug.c	Provides a debug module for printing to the kernel log the raw touch and button information.
..kernel/drivers/input/touchscreen/cyttsp5_device_access.c	Provides external user access to the device through the driver to provide an interface for writing commands to the device and to receive the corresponding responses.
..kernel/drivers/input/touchscreen/cyttsp5_devtree.c	Device Tree module that parses the Device Tree and

File Name and Location	Description
	creates platform data for devices and registers them with the bus. (TTDA 3.0.1 and later).
..kernel/drivers/input/touchscreen/cyttsp5_i2c.c	Provides an interface primitive function for performing I^2C transfers. Transfers are read or write operations. This module also provides bus binding for the TSG5 core module.
..kernel/drivers/input/touchscreen/cyttsp5_loader.c	Provides both manual and automatic firmware update capability.
..kernel/drivers/input/touchscreen/cyttsp5_mt_common.c	Common multi-touch functions for the TSG5 Multi-touch module.
..kernel/drivers/input/touchscreen/cyttsp5_mta.c	Multi-touch Protocol A module. Provides Linux Protocol A touch signaling.
..kernel/drivers/input/touchscreen/cyttsp5_mtb.c	Multi-touch Protocol B module. Provides Linux Protocol B touch signaling.
..kernel/drivers/input/touchscreen/cyttsp5_platform.c	Holds platform data for the core and primitive functions which are customizable.
..kernel/drivers/input/touchscreen/cyttsp5_proximity.c	Provides the proximity feature capability.
..kernel/drivers/input/touchscreen/cyttsp5_regs.h	TSG5 device register bit definitions.
..kernel/drivers/input/touchscreen/cyttsp5_spi.c	Provides an interface primitive function for performing SPI transfers. Transfers are either a read or write operation. This module also provides bus binding for the TSG5 core module.
..kernel/drivers/input/touchscreen/cyttsp5_test_device_access_api.c	Example module that shows how to interface to the TTDL 3.x device access through another driver.
..kernel/drivers/input/touchscreen/Kconfig	Local modified build file for PandaBoard/TTDL driver.
..kernel/drivers/input/touchscreen/Makefile	Local modified build file for PandaBoard/TTDL driver.
..kernel/drivers/input/touchscreen/Kconfig.dragonboard-apq8074	Local modified build file for MSM Dragon Board/TTDL driver (TTDA 3.4).
..kernel/drivers/input/touchscreen/Makefile.dragonboard-apq8074	Local modified build file for MSM Dragon Board/TTDL driver (TTDA 3.4).
..kernel/drivers/input/touchscreen/Kconfig.patch	Local modified build file for PandaBoard/TTDL driver in patch file format to ease integration. (TTDA 3.5)
..kernel/drivers/input/touchscreen/Makefile.patch	Local modified build file for PandaBoard/TTDL driver in patch file format to ease integration. (TTDA 3.5)
..kernel/include/linux/cyttsp5_core.h	Defines the Core module platform data structure.
..kernel/include/linux/cyttsp5_device_access-api.h	Calling interfaces for the device access through the TTDL from another driver.
..kernel/include/linux/cyttsp5_platform.h	Holds platform data for the core, loader, and primitive functions which are customizable.

Not included in driver distribution – Contact your local Parade FAE for more details on these files.	
..kernel/Makefile	Kernel root build. PandaBoard build modifiers to include Parade driver.
<android>/system/usr/idc/cyttsp5_mt.idc	Sample file that tells Android how to interpret multi-touch signals sent by Parade driver. Needs to be pushed into

File Name and Location	Description
	Android /system/usr/idc.
<android>/system/usr/keylayout/cyttsp5_mt.kl	Sample virtual key layout file. Needs to be pushed into Android /sys/usr/keylayout if virtual key is to be used.
<android>/system/usr/keychars/cyttsp5_mt.kcm	Sample virtual key configuration file. Needs to be pushed into Android /sys/usr/keychars if virtual key is to be used.
<kernel>/firmware/cyttsp5_fw.bin	TTDL firmware binary file for automatic driver flashing at startup based on firmware revision information.

4. TTDL Modules and Program Flow

This chapter describes the TTDL Module types and a high level program flow within these modules. This includes the Linux bus type defines and implementation, build methods, introduction to the Multi-Touch module, CapSense button keycode signaling module, proximity module, and the debug module. It also documents virtual key signaling, user interfaces via sysfs and debugfs nodes, implementation of TTDL data types and control structure, as well as application development environment.

For TTDA 3.2 and later, this method is simplified to use the binding of the host bus instead of the special TTDA bus. Also, the MTA/B, Button, and Proximity worker modules are combined with the core module to form a Core Driver module and are conditionally included in the build. The loader and device access modules remain loadable modules.

4.1 Module Types

4.1.1 Core Module

The Core Driver module combines all the functions of the Multi-Touch (MTA/B), CapSense Button (BTN), Proximity, and Core modules. The Core Driver is built as a single module and each adapter (I2C and SPI) is built as a separate module. The feature modules will be buildable as loadable modules however; they will not be registered with separate Linux devices. They will share the same device as the Core Driver module. They will continue to subscribe for event service for interrupt callback notification.

Core provides service for:

- Device startup
- Device interrupt pin
  - The interrupt service will read the current mode from the device and call back the subscribers to run within the context of the ISR to ensure that handling is complete before the interrupt service complete signal is sent to the OS by the ISR. The subscriber call back can run inline or send a waking signal to another thread. Subscriptions will be set up and served using Linux List objects. Wait on event queues with conditional wake up is the preferred method of synchronizing threads
  - The interrupt service will base the current mode on device conditions. New modes are only added if they can be determined by reading the device
- Concurrent access protection
  - Interlocks are provided
  - Request access
    - Core provides access to each module in the order of request
    - Core blocks other requester(s) until access is relinquished by accessing module
- Mode-protected read/write access calls to allow subscribed modules access to adapter primitives. Core translates read/write requests into appropriate adapter read/write adapter calls
- Platform data
  - Driver default Easy Wakeup Gesture ID (TTDA 3.1 and later)
- Platform data configurable function calls
  - Initialization
    - Platform-specific startup initialization code
  - Hard reset

- Includes reset and wait blocking capability
- Includes soft reset as default if hard reset function is not defined

☐ Wakeup

- Provide hard wakeup interrupt pin strobing
- Alternative includes power cycling for power management instead of using device low power mode

- Suspend/Resume handling
  - ☐ Suspend thread
    - Core puts the device into Deep Sleep.
      i. Sets the Deep-Sleep by writing the HID Set Power (SLEEP) command to the device
    - Core puts the device into the Easy Wakeup low-power mode
      i. *Sysfs* object for user selection of wake gesture
      ii. Sets interrupt properties to allow waking host
      iii. Writes the Easy Wakeup command to device
  - ☐ Resume thread
    - Core wakes the device from Deep Sleep
      i. Wakes by writing the HID Set Power (ON) command to the device
    - Device wakes host on Easy Wake gesture recognition
      i. Ping command to verify device
- The Core Driver exposes a set of module interface commands to subscriber modules through the following structure:
  ☐ `struct cyttsp5_core_nonhid_cmd`

4.1.2 Worker Modules

Provides an interface to OS signaling and OS file system objects

- Subscribes to Core Module for interrupt event signaling. Provides interrupt attention callback function to access device data. Typically the callback is called inline without thread rescheduling so that it is ensured that data is read before the core releases the interrupt back to the kernel (IRQ_HANDLED).
- Multi-touch event signaling
  - Sends multi-touch signal usage, including min/max signal values, to kernel to set capabilities at startup
    i. Uses platform data to configure signals to be used based on product requirements
  - Runtime touch positions
    i. Sends each touch value as part of its Linux signal
- CapSense button keycode signaling
- Proximity signaling (TTDA 3.1 and later)

Provides optional device access through user space

- User interfaces
- Sysfs interface standard
  - Show
  - Store

- Firmware Class standard
    - Loading object
    - Data object

4.1.3 Adapter Module

- Provides an interface to the device communication bus
- I^2C
- SPI
- Exposes read/write primitives for Core access to device
- Adapter devices launched by adding adapter device structure to system launch list for the bus type
- Adapter drivers launched as loadable kernel module (LKM) with TTSP Bus driver registration call in the module_init() function
- In TTDA 3.0.1 and later, the Adapter Modules are updated to provide optional dynamic parsing of the Device Tree to see if matching tree properties are provided for the TTDL

  The normal board configuration calls to initiate the host bus for the adapter and to register the TTDL modules using statically-defined platform data. If the device tree compile-time option is used, this data is replaced with a system call to provide binding of the Device Tree. The Device Tree allows removing the platform data from the board configuration and adding it into the Device Tree in the form of property fields. The Device tree also includes property fields used by the system to initiate the host bus. The adapter Modules add calls to dynamically allocate a platform data structure for the module, parse the Device Tree to access the platform data content for each TTDL module using system parsing calls, and copy the Device Tree data to the allocated structure. The calls include references to the dynamically allocated platform data.

4.2 Module Build Methods

4.2.1 Built-In

- Use the standard build with the “-y” option to create “.o” files
- Included in the built-in ”.o” intermediate linker/loader files

4.2.2 Loadable Kernel Module (LKM)

- Uses Linux loadable kernel modules for touch driver components
- These modules are built with the “-m” option to create “.ko” files.
- The “.ko” files can be loaded by hand/script using the insmod command line directive.
- The “.ko” files can be loaded at system startup by adding insmod directives in the Android “init.rc” file.
- Three types of LKM touch drivers

1. Core: System Information and COMM_INT interrupt service handling. Interrupt attention and device mode handling.

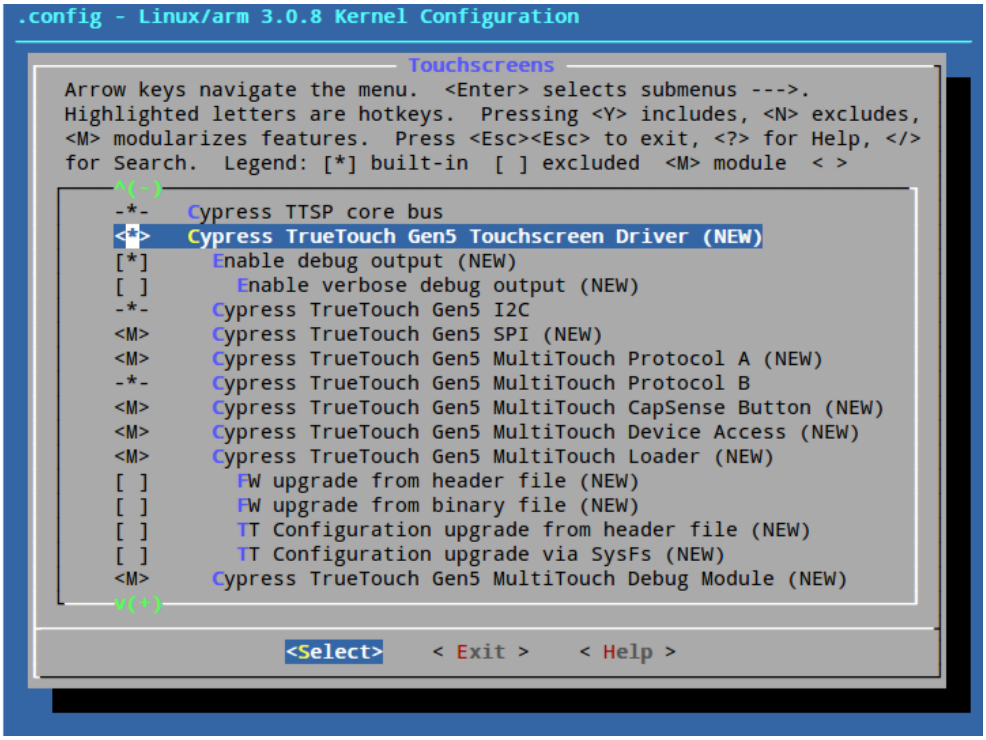
- The core reads the length bytes (TTDL 3.x) on each interrupt.
- Subscribing modules use the status information provided by the core.
    - Eliminates redundant read of status bytes by each subscriber.

2. Worker: Provides device information to system OS signal. This type of module can include multi-touch signaling, user sysfs driver control, and firmware class device loader modules.

- Interprets the length bytes (TTDL 3.x) read by the core.
- Reads any required additional touch record bytes.

3. Adapter: System communication bus interface module.

Figure 4-1 Module Selection (TTDA 3.0 Example)



4.2.3 Device

Refer to *Part-Specific Support* in TrueTouch Device Support.

4.3 Driver Module Loading

Modules can be built to load automatically or manually. Modules that are compiled as “built-in” are loaded automatically at startup, while loadable kernel modules (LKM) must be loaded manually or with a script. Parade’s build instructions always make the bus driver “built-in” to ensure that it loads first. The commands in this section are example commands for TI PandaBoard reference platform.

4.3.1 Automatically Loading Modules

To build automatically loading modules:

1. Under the kernel directory

```
make panda_defconfig
make
```

2. Reboot to load modules.

4.3.2 Manually Loading Modules

To build manually loading modules:

1. Under the kernel directory

```
make
```

2. Push the generated .ko files to the running Android system using adb command.

4.3.3 Module Dependencies

4.3.3.1 TTDA 3.1 and Earlier

There is no dependency for module loading. Modules can be inserted in any order. However, when the device is probed, there is module dependency. MTA/MTB/BTN/Loader/Debug/Device Access modules depend on Core module. Core module depends on I^2C/SPI modules. When all are probed, to remove modules, MTA/MTB/BTN/Loader/Debug/Device Access modules need to be removed before Core module can be removed. Similarly, Core module needs to be removed before removing I^2C/SPI module. **Note:** In TTDA 3.1, the Proximity module must be compiled as built-in. See the **Error! Reference source not found.** section for more details.

Table 4-1 lists the modules that must be launched before activating features.

Table 4-1. Activation Dependencies

	Launched Modules							
	Core	**I^2C**	**SPI**	**MTA/MTB**	**BTN**	**Loader**	**Debug**	**Proximity**
Device	✓	✓	or ✓					
Touches	✓	✓	or ✓	✓				
CapSense Buttons	✓	✓	or ✓		✓			
Touches and CapSense Buttons	✓	✓	or ✓	✓	✓			
Touch Report Debug Prints	✓	✓	or ✓				✓	
Proximity	✓	✓	or ✓					✓

Module loading/unloading guidelines:

- The bus driver should always be compiled as “built-in.”
- End users will never load/unload driver modules.
- The debug module is not loaded automatically (unless it is compiled as “built-in”).
- Mobile product developers want to be able to load the debug module on any product.
- If OTA firmware is distributed in the kernel build, the loader module is loaded automatically.
- The dynamic load and unload of modules is limited to the multi-touch (MTA/MTB), CapSense button (BTN), and Debug (dbg) user modules.

4.3.3.2 TTDA 3.2 and Later

The Core Driver module includes the Multi-touch (MTA or MTB), CapSense Button, and IR replacement proximity functions. The Core module must be loaded first and then any or all of the loadable modules (Loader, Device Access, and Debug Monitor) can be loaded. The loadable modules can be loaded in any order after the Core driver has been loaded.

4.4 Application Development Environment

Applications use simple “echo” and “cat” file system interface commands. To write commands, either “echo” a single value or a quoted string and redirect to the file system interface (such as group data). To read the status, perform a “cat” and the status data is displayed at the command line with line separation between each status byte returned.

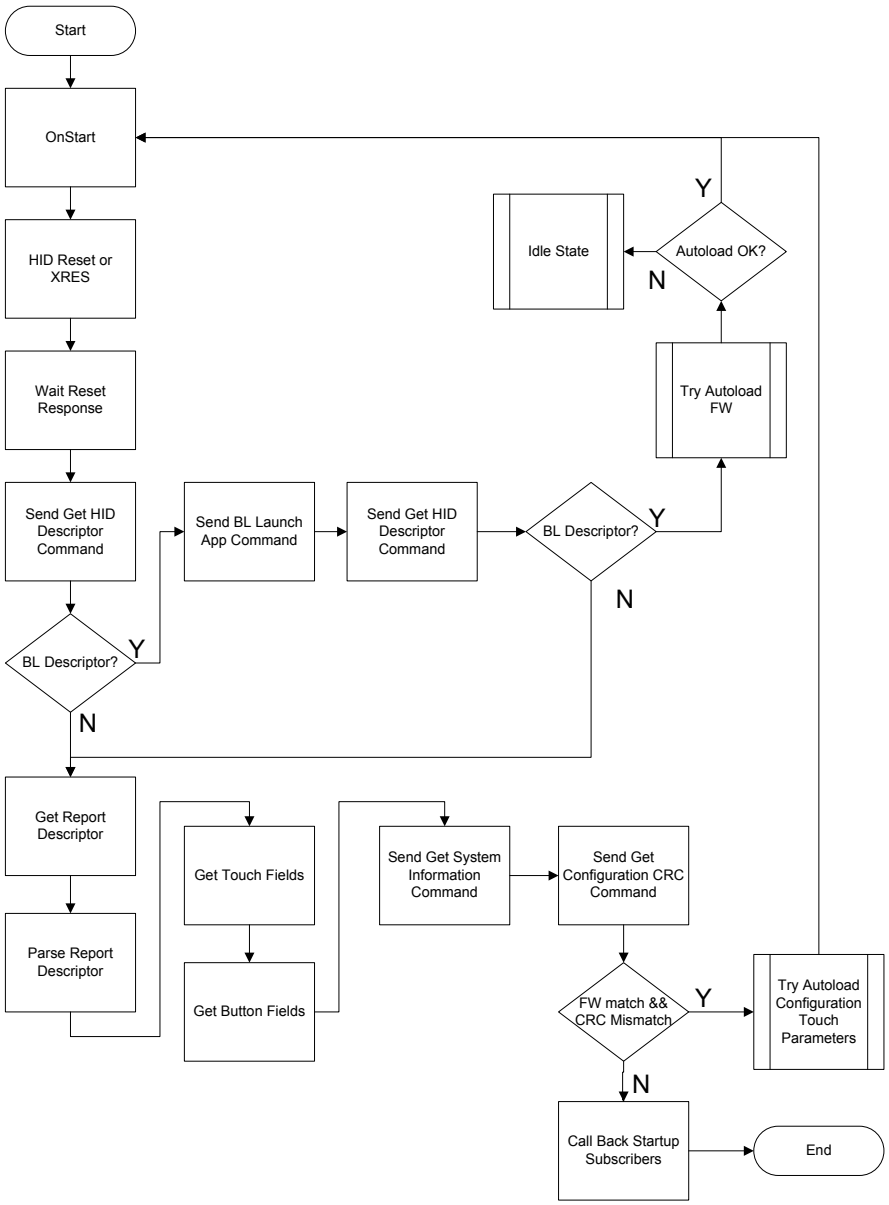
4.4.1 Characteristics

One purpose of the Module architecture is to allow alternate user modules to be created that can run in parallel with the standard set of modules. The alternate modules can subscribe for the same information. For example, a Debug module can be created that subscribes for interrupt attention and can dump the raw touch data to the command display without changing the normal touch flow.

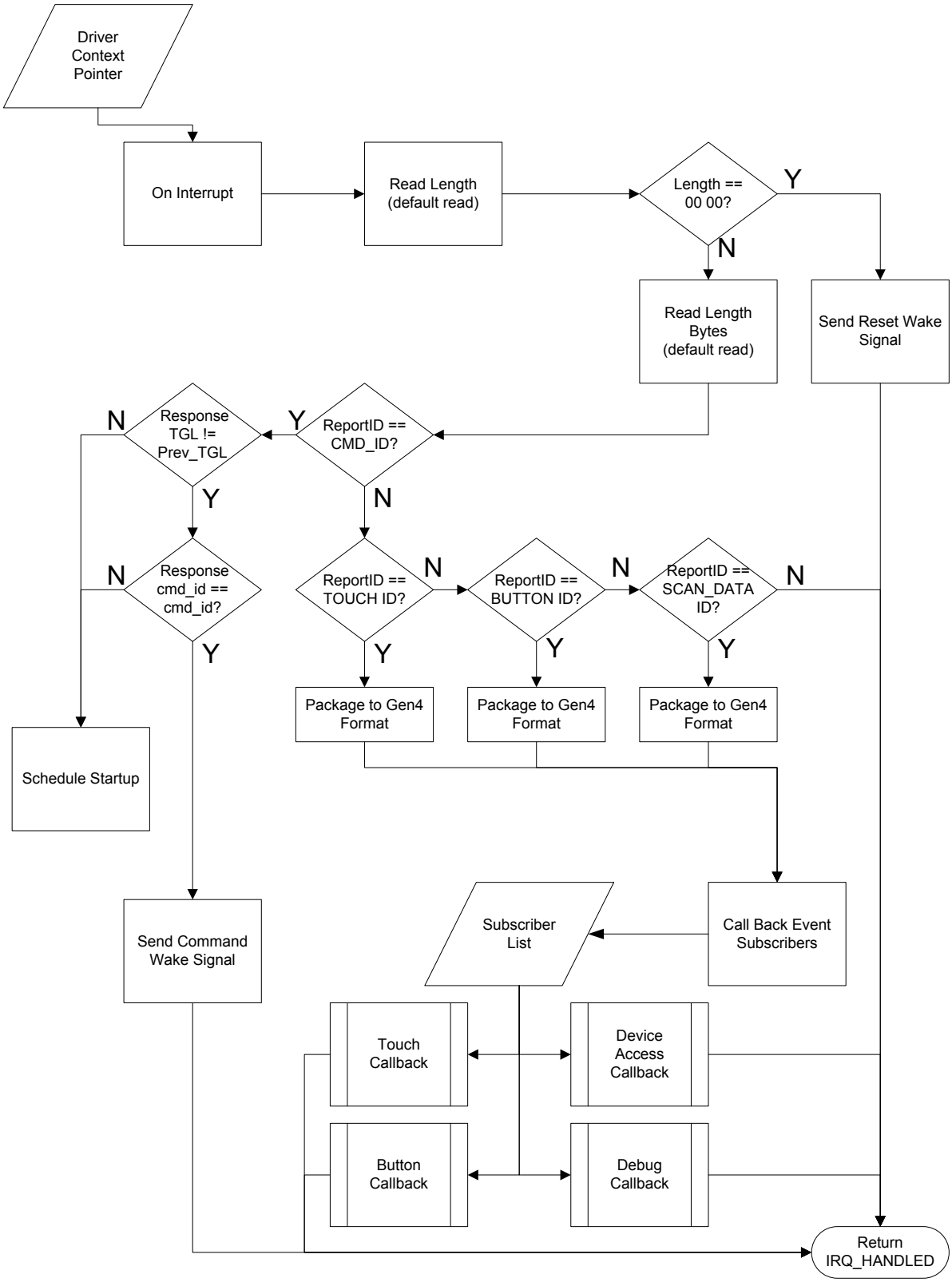
This chapter illustrates the flow diagrams software design.

4.5 Program Flow

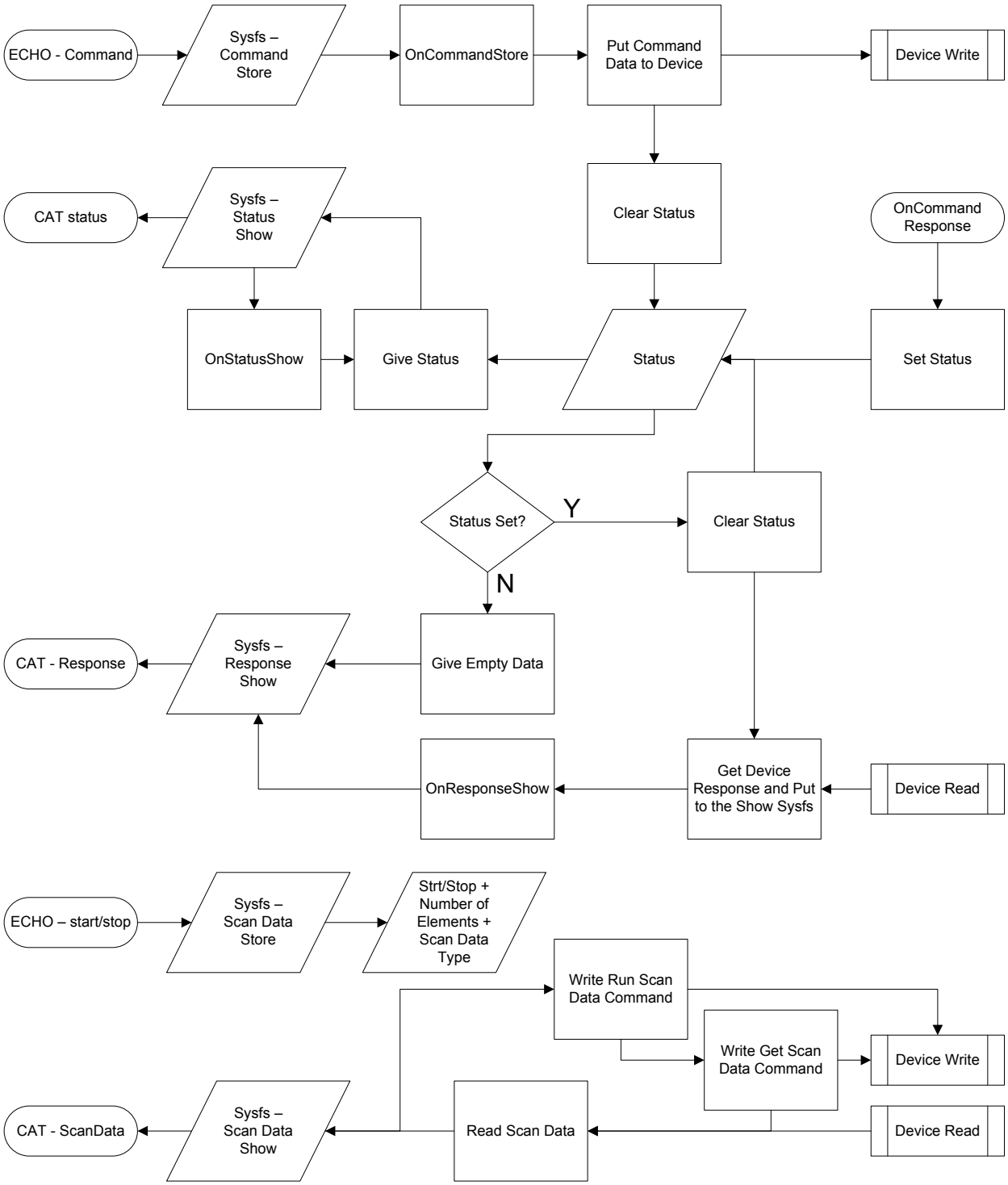
4.5.1 Start Thread



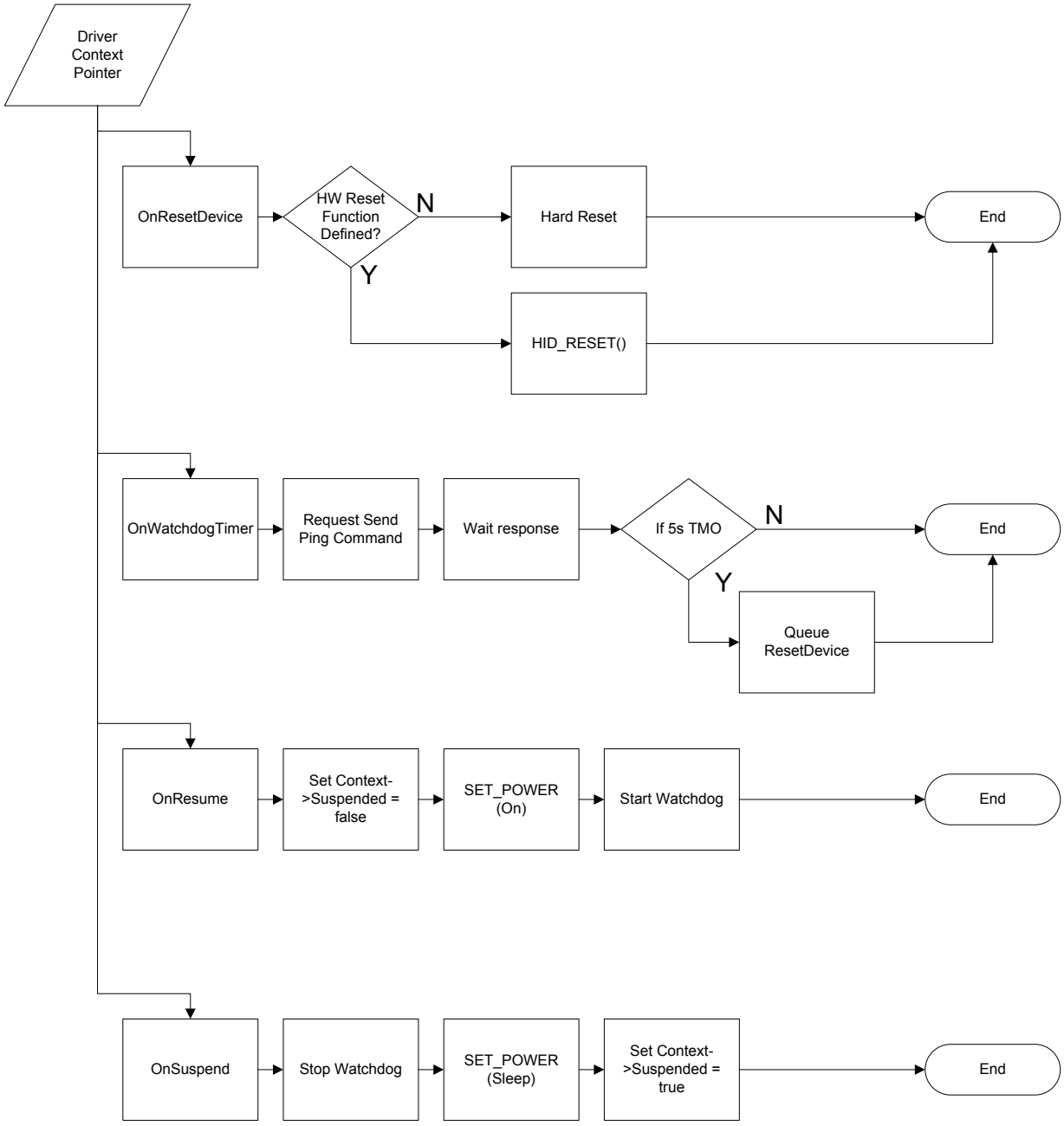
4.5.2 Interrupt Service Thread



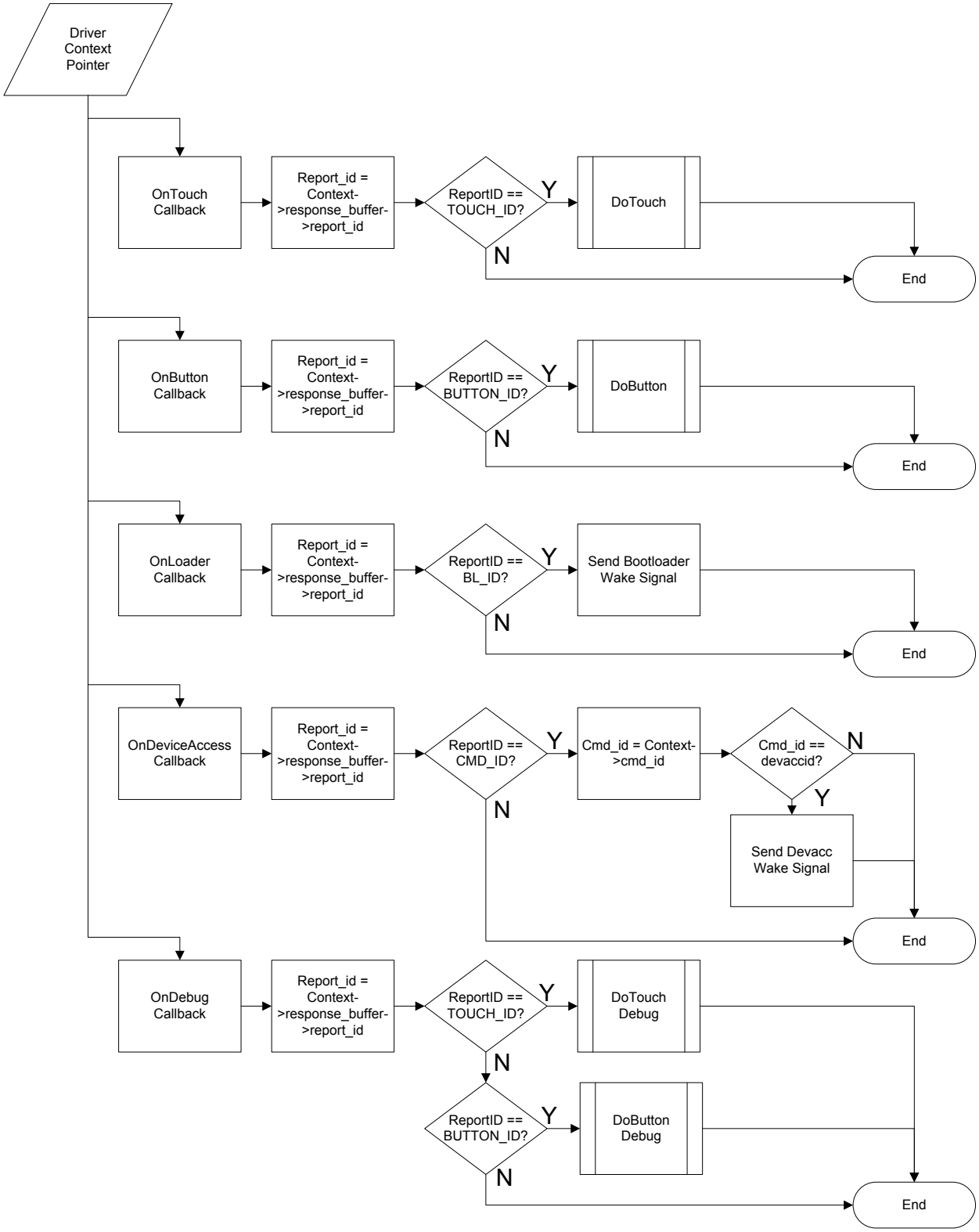
4.5.3 Device Access Flow



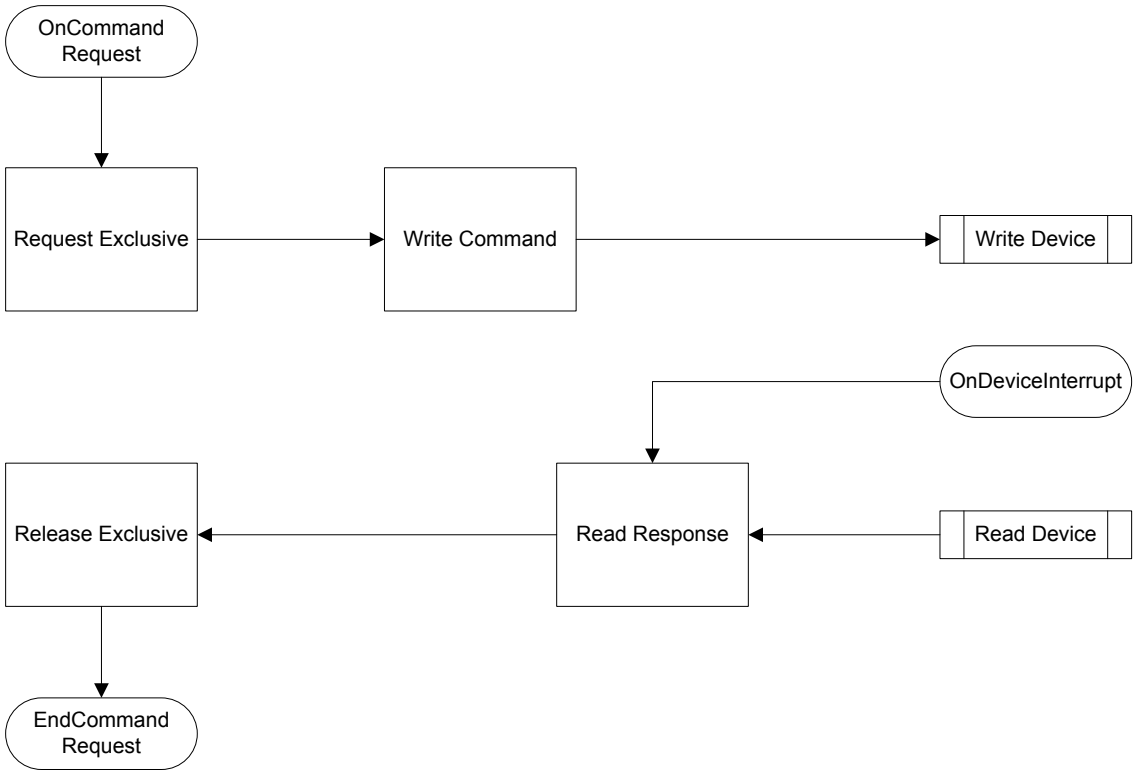
4.5.4 Thread Flow



4.5.5 Callback Flow



4.5.6 Request Command Output



4.6 Error Handling

It is important to consider testability when designing your system software. Proper Linux error handling is required. Whenever a value is returned from any function, the value is tested for error. If an error is returned, the calling function prints an error message to the kernel log.

The kernel log print primitives are: dev_info, dev_err, dev_dbg, and dev_vdbg.

4.6.1 Dev_info()

Dev_info() is called whenever something is to be printed to the kernel log unconditionally. This includes registration information. The format of the call is:

dev_info(struct device *dev, char *fmt, __func__, arglist…)

where:

struct device is defined by Linux

char *fmt is a format string starting with “%s:” for __func__

arglist are all items specified in *fmt

4.6.2 Dev_err()

Dev_err() is called for any error condition to provide a record in the kernel log. The format of the call is the same as dev_info(). The kernel log can be printed dynamically for real time monitoring of program flow.

4.6.3 Dev_dbg()

Dev_dbg() is called to report information to the kernel log to assist in debugging. The format of the call is the same as dev_info(). This is primary debug information and will be limited to allow debug logging without affecting real time performance (such as simple X, Y reporting on each touch). This call is conditionally compiled by placing the following define at the beginning of the code file:

```
#define DEBUG = y
```

4.6.4 Dev_vdg()

Dev_vdg() is called to report additional information to the kernel log to assist in debugging. This call is the same as dev_dbg() but is conditionally compiled for verbose debug by placing the following define at the beginning of the code file:

```
#define VERBOSE_DEBUG
```

4.6.5 pt_debug()

TTDL version 3.8 added a macro that can be called with 4 different debug levels:

- DL_QUIET
- DL_ERROR
- DL_WARN
- DL_INFO
- DL_DEBUG

The macro will print the debug message to kmsg based on the user controlled system wide debug level. Pt_debug() utilizes the dev_dbg() method to print out the message, if and only if the passed in level is greater or equal to the system debug level. The system level can be controlled with the “drv_debug” sysfs node.

4.6.6 Watchdog

Driver watchdog is a timer that updates periodically to check for touch device failure and trigger a device reset for recovery if necessary. When an ESD event happens, touch device may not react or malfunction.

In TTDA 3.0, a Linux system timer is used to implement a watchdog to periodically ping the device using the PIP defined operational NULL command. If the device fails to respond, then the driver queues a restart.

To disable the watchdog, set the constant CY_WATCHDOG_TIMEOUT to 0 in the *cyttsp5_regs.h* file:

```
#define CY_WATCHDOG_TIMEOUT 0
```

To enable watchdog, set the constant CY_WATCHDOG_TIMEOUT to non-zero in the *cyttsp5_regs.h* file. The CY_WATCHDOG_TIMEOUT constant defines the interval before a watchdog reset occurs, in milliseconds (For example, 1000 sets a 1-second interval):

```
#define CY_WATCHDOG_TIMEOUT 1000
```

During development, this value may be set to a larger timeout period (such as 5000) to reduce the watchdog activity during product startup. The default value of 1000 (1 sec) is long enough to have low impact on the product during normal activity and short enough to allow device failure detection and reset in a reasonably short human time frame.

Section B: Features and Modules

This section documents information about the features and modules included in the latest release of TTDL. Some features that have been deprecated may still be included in this section but the version the feature was introduced and deprecated will be listed.

Section B contains the following chapters sorted alphabetically:

- CapSense Button Keycode Signaling Module
- Dynamic Debugging Module
- Easy Wakeup
- Firmware Upgrades (OTA Updates)
- Manufacturing Tests
- Multi-Touch Signaling Module
- Power Management
- Proximity Sensing Signaling Module
- Restore Parameters on Restart
- Stylus, Hover and Glove
- User Interfaces (sysfs, debugfs)
- Virtual Key Signaling

5. CapSense Button Keycode Signaling Module

Module Name	BTN Module
Files	cyttsp5_btn.c
Introduced in Version	TTDL 3.0 8-Button support - TTDA 3.6
Deprecated in Version	n/a

5.1 Details

The BTN module subscribes to the Core module for touch event interrupt notification. The Core calls the subscription callback for the subscribing BTN module.

- Provides the method to map keycodes to buttons in touch module platform data.
  - The Core module provides an array of button-to-key code conversion values. A copy of the array is attached to the system information read by the Core module. This array is part of the Core module platform data and is made available to the BTN module when the BTN module requests a pointer to the system information.
- Adds button to keycode signaling setup in touch module probe function.
- This is implemented as a separate TTSP module that subscribes for the interrupt events and uses the status information read by the core. If debug is enabled, the module also reads the I^2C bus for the button difference report data.
- The module keeps a record of which buttons are pressed and released and reports key press and release when it detects a change in button condition. Multiple buttons can change state on each interrupt event.

5.1.1 8-Button Support

The PIP definition for number of buttons supported by a device firmware has been increased from 4 to 8 buttons. This change includes the increase in the number of supported buttons in the System Information and the number of buttons reported in the button report descriptor. The Button report is updated to both report the button press and release status of up to 8-buttons and the signal data for up to 8-buttons. The TTDL allows the assignment of Key Codes to 8-buttons in the platform data in either the “cyttsp5_btn_keys[]” array for board configuration support or in the “cy,btn_keys” field of the Device Tree structure.

6. Dynamic Debugging Module

Module Name	Debug Module
Files	cyttsp5_debug.c
Introduced in Version	TTDL 3.0
Deprecated in Version	n/a

6.1 Details

The Debug module subscribes to the Core module for interrupt notifications for touch events. . The Core calls the subscription callback for the subscribing Debug module. The Debug module formats the touch record information and prints the information to the kernel log. The format of the touch data is in either human readable or terse machine readable format. The machine readable format is suitable for the touch interface of the TTHE Mobile Tuner.

The Debug module should only be built as LKM and not built-in because it will degrade normal touch performance. The Debug module should be loaded by the user on demand and removed when the debug session ends.

- Subscribes for touch interrupts
- On-touch interrupt
  - Increments an interrupt count
  - Reads complete touch record and prints raw or formatted byte content onto the kernel log along with the interrupt count. Default is raw bytes suitable for TTHE Mobile Tuner monitoring for TTHE touch tracking and line drawing. This information is printed with the string preface tag “cyttsp5_OpModeData”.
  - Prints the sensor data onto the kernel log with the string preface tag “cyttsp5_sensor_monitor”. This is activated automatically when the start *Sensor Data Mode* Configuration and test command is sent by TTHE when using the device access Group 15.
  - Provides the sysfs interface `cyttsp5_interrupt_count` to show or store the interrupt count number
    i. SHOW: current interrupt count
    ii. STORE: reset interrupt count
  - Provides the sysfs interface `cyttsp5_formated_output` to select raw or formatted byte output
    i. SHOW: current raw or formatted selection
    ii. STORE: set raw or formatted selection

7. Easy Wakeup

Feature Name	Easy Wakeup
Files	
Introduced in Version	Basic Easy Wakeup – TTDL 3.1 Advanced Easy Wakeu – TTDL 3.6
Deprecated in Version	n/a

7.1 Basic Easy Wakeup Details

The Core module is updated to include the ability for the touch device to wake up the host on recognition of one or more specified gestures. An operational parameter is added to allow selection of either a predefined wakeup gesture (allows device to wake up the host) or use deep sleep (host must wake the device) when the product enters low-power state. A user-to-kernel system interface is added to allow the host to tell the driver what the selection is for the wake gesture or if deep sleep is specified.

- The initial easy wakeup gesture value can be set in `_cyttsp5_core_platform_data` struct in the Linux Board Config file. An example of initializing easy wakeup gesture as 0xF is shown in the following code as part of the Linux Board Config.

  Note: Different firmware might have different values for the various easy wakeup gestures.

```
static struct cyttsp5_core_platform_data _cyttsp5_core_platform_data = {
    .irq_gpio = CYTTSP5_I2C_IRQ_GPIO,
    .rst_gpio = CYTTSP5_I2C_RST_GPIO,
    .xres = cyttsp5_xres,
    .init = cyttsp5_init,
    .power = cyttsp5_power,
#ifdef CYTTSP5_DETECT_HW
    .detect = cyttsp5_detect,
#endif
    .irq_stat = cyttsp5_irq_stat,
    .sett = {
            NULL,  /* Reserved */
            NULL,  /* Command Registers */
            NULL,  /* Touch Report */
            NULL,  /* Data Record */
            NULL,  /* Test Record */
            NULL,  /* Panel Configuration Record */
            NULL,  /* &cyttsp5_sett_param_regs, */
            NULL,  /* &cyttsp5_sett_param_size, */
            NULL,  /* Reserved */
            NULL,  /* Reserved */
            NULL,  /* Operational Configuration Record */
            NULL,  /* &cyttsp5_sett_ddata, *//* Design Data Record */
            NULL,  /* &cyttsp5_sett_mdata, *//* Manufacturing Data Record */
            NULL,  /* Config and Test Registers */
            &cyttsp5_sett_btn_keys,    /* button-to-keycode table */
    },
    .loader_pdata = &_cyttsp5_loader_platform_data,
    .flags = CY_CORE_FLAG_WAKE_ON_GESTURE,  /* TTDL 3.5- */
    .flags = CY_CORE_FLAG_NONE,             /* TTDL 3.6+ */
    .easy_wakeup_gesture = CY_CORE_EWG_TAP_TAP
```

```
            | CY_CORE_EWG_TWO_FINGER_SLIDE,
};
```

- The easy wakeup gesture flags and macro definitions are in the *cyttsp5_core.h* file.

```
enum cyttsp5_core_platform_flags {
    CY_CORE_FLAG_NONE = 0x00,
    CY_CORE_FLAG_WAKE_ON_GESTURE = 0x01, /* TTDL 3.5- enable device to wake on gesture
*/
                                        /* TTDL 3.6+ not used; simply set
                                         * easy_wakeup_gesture to one or more gestures
                                         * to enable wake on gesture; set to
                                         * CY_CORE_EWG_NONE to disable wake on gesture
                                         */
    CY_CORE_FLAG_POWEROFF_ON_SLEEP = 0x02, /* TTDA 3.2+ power off for sleep*/
    CY_CORE_FLAG_RESTORE_PARAMETERS = 0x04, /* TTDA 3.3 restore RAM params on restart */
};

enum cyttsp5_core_platform_easy_wakeup_gesture {/* TTDA 3.3 */
    CY_CORE_EWG_NONE = 0x00,
    CY_CORE_EWG_TAP_TAP = 0x01,
    CY_CORE_EWG_TWO_FINGER_SLIDE = 0x02,
    CY_CORE_EWG_RESERVED = 0x03,
    CY_CORE_EWG_WAKE_ON_INT_FROM_HOST = 0xFF,
};
```

- The easy wakeup gesture can also be changed using a sysfs interface during runtime. The sysfs path is

```
/sys/bus/ttsp5/devices/main_ttsp_core.cyttsp5_i2c_adapter/easy_wakeup_gesture
```

Note: Setting easy_wakeup_gesture to 0 or 255 disables easy wakeup functionality. Instead, the regular deep-sleep option will be applied.

7.2 Advanced Easy Wakeup Support

Easy Wakeup has been redefined for the PIP specification 1.6 and above such that the host driver no longer selects the wakeup gesture id. Instead, the firmware is programmed to recognize a number of gestures and on waking, reports which of the gestures woke up the device.

For firmware that complies with PIP spec 1.5 and below, the TTDL tells the device which of the following gestures can wake up the device:

- One finger double tap (gesture ID 01h)
- Two finger vertical slide (gesture ID 02h)

For firmware that complies with PIP 1.6 and above, the firmware will ignore the host value and wake on any of the following gestures:

- Single finger slide (decreasing TX direction) (gesture ID 05h)
- Single finger slide (increasing TX direction) (gesture ID 06h)
- Single finger slide (decreasing RX direction) (gesture ID 07h)
- Single finger slide (increasing RX direction) (gesture ID 08h)
- ASCII character recognition. (gesture ID in the range 20h to 7Fh)

To support the advanced Easy Wakeup, a new constant must be defined in the build:

#define EASYWAKE_TSG6

Place this “#define” at the beginning of the cyttsp5_regs.h file to ensure that it is defined for all source files.

The base firmware supports the finger slide gestures (ID 05h to 08h), but no character recognition. Custom firmware can be requested that can support 4 to 5 letter and/or numeric characters. If character recognition is enabled in the firmware, then the Wakeup Event report will return the character ascii value in the range 0x20 to 0x7F as the Gesture ID.

For tuning and debugging purposes the TTDL adds two sysfs objects to allow reporting both the waking gesture id and the gesture touch data.

7.2.1 Gesture ID

The Gesture ID for the most recent Wakeup Event report can be printed with the Gesture ID sysfs object. The path to this object is the same path as for the Standard Custom User Commands (Section **Error! Reference source not found.Error! Reference source not found.**). For example:

```
> cd /sys/bus/i2c/devices/4-0024
> cat easy_wakeup_gesture_id
```

This will print the waking gesture ID as a hex number.

7.2.2 Gesture Data

In addition to the Gesture ID, the Wakeup Event report returns touchdown information such as X-position in 2-bytes and Y-position in another 2-bytes:

- For the One finger double tap (Gesture ID 01h), two positions are returned. The first data position is the first touchdown point (X1 and Y1) and the second position is the second touchdown point (X2 and Y2).
- For the Single finger slide gestures (Gesture ID 05h to 08h), two positions are returned. The first position is the touchdown point (X1 and Y1) and the second position is the lift off point (X2 and Y2).

For example with (X1, Y1) at (0x0111, 0x0122) and (X2, Y2) at (0x0211, 0x0222):

```
> cd /sys/bus/i2c/devices/4-0024
> cat easy_wakeup_gesture_show
0x11                            // X1 (LSB)
0x01                            // X1 (MSB)
0x22                            // Y1 (LSB)
0x01                            // Y1 (MSB)
0x11                            // X2 (LSB)
0x02                            // X2 (MSB)
0x22                            // Y2 (LSB)
0x02                            // Y2 (MSB)
(8 bytes)
```

8. Firmware Upgrades (OTA Updates)

Feature Name	Firmware Upgrades
Files	cyttsp5_loader.c
Introduced in Version	SWD in TTDA 3.2.1 and TTDA 3.3.1 only Loading by Panel ID - TTDA 3.4
Deprecated in Version	n/a

8.1 Using the Firmware Class for Device Firmware Loading

See section User Interfaces (sysfs, debugfs) for details on sysfs node usage.

- The Loader module starts up the firmware class.
- The firmware class creates two special sysfs objects: loading and data.

  When a '1' is written to one of the loading objects, the firmware class starts the loader (typical paths shown here)
  - For PIP1.x to FLASH: `echo 1 > /sys/bus/ttsp5/devices/2-0024/manual_upgrade`
  - For PIP2.x to FLASH: `echo 1 > /sys/bus/ttsp5/devices/2-0024/pip2_manual_upgrade`
  - For PIP2.x to SRAM: `echo 1 > /sys/bus/ttsp5/devices/2-0024/pip2_manual_ram_upgrade`

  Start the firmware class in order to accept the binary file:

  ```
  echo 1 > /sys/class/firmware/2-0024/loading
  ```

  The contents of a file can be copied to the firmware class by concatenating a file to the data sysfs.

  ```
  cat <file> > /sys/class/firmware/cyttsp5_loader.main_ttsp_core/data
  ```

  When the "cat" operation is complete, the loader can be stopped by writing '0' to the loading object

  ```
  echo 0 > /sys/class/firmware/cyttsp5_loader.main_ttsp_core/loading
  ```
- When this is done, the firmware class will copy the file contents into a kernel space memory block and invoke a driver callback function with a (u8 *) pointer to the kernel space memory block and a size value indicating how many bytes were copied.
- The format of the loader file will be binary:
  - ☐ The content will be according to the existing driver conversion tool.
  - ☐ To support OTA, the loader startup sequence uses either an included platform data firmware image or a firmware class to detect if a firmware binary file is saved into the firmware class default directory.

8.2 SWD Firmware Loader

IMPORTANT: TTDA 3.2.1 and TTDL 3.3.1 are limited distributions, which enables SWD programming in the driver, in addition to bootloading. This distribution of code is only compatible with specific TrueTouch part numbers. Please work directly with a Parade Design Services (CDS) engineer or Field Application Engineer (FAE) to ensure that all prerequisites are met before using this limited distribution driver. For guidance on prerequisites and usage of the SWD firmware loader feature in TTDA 3.2.1, refer to Parade specification 001-90764.

The Loader is updated to include the ability to acquire the device via the SWD Interface. The Loader provides SWD word serializer for write operations including writing the acquisition key and writing the load firmware data to the device. The Loader also provides SWD word deserializer to read response data from the device.

The SWD Loader expects a load file in HEX format, so the Loader is updated to read and parse firmware from either a HEX file compiled as an image in the driver or available as a file in the file system and accessible via Linux firmware class.

Implementing SWD Loader on a product requires proper GPIO pin wiring to support the SWD electrical interface. Additionally, the product Linux build environment must allow direct “bit-bang” of the SWD Clock and I/O pins by the SWD Loader. If these requirements cannot be met, then the SWD Loader is not available for that product. Standard TTDL Loader must be used instead, limiting the device firmware to what will fit on the device along with the bootloader firmware.

8.3 Loading by Panel ID

The TTDL adds another structural layer over the loader structures used to match the required load files to Panel IDs. The mechanism will be backward compatible to allow single load file implementations using the previous one file method that does not use the Panel ID.

8.3.1 Bin Files

The new method will provide a standard naming for load files that include a reference to Panel ID. These have a fixed naming convention: for example, for Panel ID 0, the file name to use is cyttsp5_fw00.bin; for Panel ID 1, the file name to use is cyttsp5_fw01.bin; and so on.

8.3.2 Included Image Files

To handle the multiple include file images files, the following structures are updated to allow both the original single bin file and the new multiple bin file implementations.

```
/* Embedded Firmware image pointer structure */
struct cyttsp5_touch_firmware {
	const uint8_t *img;
	uint32_t size;
	const uint8_t *ver;
	uint8_t vsize;
	uint8_t panel_id;		/* Panel ID (if no Panel ID then this is set to 0xFF) */
};

/* Embedded TT Config image pointer structure */
struct cyttsp5_touch_config {
	struct touch_settings *param_regs;
	struct touch_settings *param_size;
	const uint8_t *fw_ver;
	uint8_t fw_vsize;
	uint8_t panel_id;		/* Panel ID (if no Panel ID then this is set to 0xFF) */
};

/* Embedded image structure containing pointers to:
 * Either Single image structures or
 * Multiple image structures lists
 */
struct cyttsp5_loader_platform_data {
	struct cyttsp5_touch_firmware *fw;	/* No Panel ID; Firmware image pointer */
	struct cyttsp5_touch_config *ttconfig;	/* No Panel ID; TT Config image pointer */
	struct cyttsp5_touch_firmware **fws;	/* Panel ID; Firmware image list pointer */
	struct cyttsp5_touch_config **ttconfigs;/* Panel ID; TT Config image list pointer */
	u32 flags;
};
```

9. Manufacturing Tests

Feature Name	Manufacturing Tests
Files	cyttsp5_device_access.c
Introduced in Version	Basic Manufacturing Tests – TTDA 3.3 C_M/C_P In-system Manufacturing Tests - TTDL 3.5 Range Checking C_M/C_P Manufacturing Test – TTDA 3.7
Deprecated in Version	n/a

Manufacturing Tests are user application interfaces to allow simple scripting to perform the following tests: get Panel Scan data, get IDAC data, run Automatic Shorts test, run Opens test, Calibrate sensors, and Initialize Baselines. Similar to the TTDA 3.2, the TTDA 3.3 is released as TTDA 3.3 and TTDA 3.3.1. The TTDA 3.3.1 release includes a loader module that can be optionally built to use an SWD Loader instead of the standard BL Loader (see the caution note in section 8.1 before you plan to use the SWD Loader in your product design).

9.1 Basic Manufacturing Tests

The Manufacturing Tests are provided as sysfs object interfaces (debugfs objects in TTDA 3.5) that are collected under a common path such as /path/to/mfg_test/auto_shorts (for example, /sys/bus/i2c/devices/4-0024/mfg_test/auto_shorts as described in **Error! Reference source not found.** and /sys/kernel/debug/4-0024/mfg_test/auto_shorts for TTDA 3.5). The complete set of added interfaces is:

- `/path/to/mfg_test/panel_scan`
- `/path/to/mfg_test/get_idac`
- `/path/to/mfg_test/auto_shorts`
- `/path/to/mfg_test/opens`
- `/path/to/mfg_test/calibrate`
- `/path/to/mfg_test/baseline`

To understand the data that is returned by the Manufacturing Tests, refer to the Application Note: AN85948 Touchscreen Manufacturing Testing with TMA448/545/568.

9.1.1 Panel Scan

The Panel Scan test will stop the normal touch scanning using the firmware Stop Scanning command. Then, the test will perform a single scan using the firmware Execute Panel Scan. The test will call the firmware Retrieve Panel Scan for as many times as necessary to extract a complete set of scan data. This data is returned as a single contiguous block of data with the length and format information. The user application can parse this data based on the touch device part type and panel configuration. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.1.1.1 Panel Scan Host Interface

Panel Scan:

```
> echo "ID" > /path/to/mfg_test/panel_scan // where: ID is Data ID for Retrieve Panel Scan
                                           // from TRM
                                           // [0=Raw-MC, 1=Base-MC, 2=Diff-MC, etc.]
> cat /path/to/mfg_test/panel_scan
Status S     //where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1      // where: S1 is retrieve status byte [0=pass, 1=fail]
ID      // where: ID is Data ID
```

```
EE      // where: EE is actual number of elements (7:0)
EE      // where: EE is actual number of elements (15:8)
FF      // where: FF is format of the data
        // (Sign type, matrix, data element size)
DD      // where: DD is the data in the same byte order as read from the device
DD
:
```

9.1.2 Get IDAC Data

The Get IDAC Data test will stop the normal touch scanning using the firmware Stop Scanning command. Then, the test will perform successive calls to the firmware Retrieve Data Structure command. Each call is to get the next block of the data as required until all the data has been read from the device. This data is returned as a single contiguous block of data with the length and format information. The user application can parse this data based on the touch device part type and panel configuration. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.1.2.1 Get IDAC Data Host Interface

Get IDAC:

```
> echo "ID" > /path/to/mfg_test/get_idac   //where: ID is the scan mode data type to
retrieve
> cat /path/to/mfg_test/get_idac
Status S                                    //Where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1          // where: S1 is retrieve status byte [0=pass, 1=fail]
ID          // where: ID is the type of scan mode data to retrieve
LL          // where: LL is length of returned data (7:0)
LL          // where: LL is length of returned data (15:8)
FF          // where: FF is format of the data (matrix)
DD          // where: DD is the data in the same byte order as read from the device
DD
:
```

9.1.3 Run Automatic Shorts Test

The Run Automatic Shorts Test will stop the normal touch scanning using the firmware Stop Scanning command. Then the test will perform the firmware Automatic Shorts command. Then the test will call the firmware Retrieve Shorts Data command. If the test summary result is a pass then the summary results are returned to the user application. If the summary results are a fail, then the test will return the summary results along with the actual return data which provides information to identify which sensor(s) is/are shorted. The user application can parse this data based on the product panel configuration. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.1.3.1 Run Automatic Shorts Host Interface

```
Automatic Shorts:
> cat /path/to/mfg_test/auto_shorts
Status S   // Where: S is [-1=bad command, 0=pass, >0=Internal Linux communication error]
== Below data is sent by the driver if Status S is pass ==
S1         // where: S1 is Run Automatic Shorts Self Test status byte
           // [0=pass, 1=fail, ff=bad command]
S2         // where: S2 is self-test summary result [0=pass, 1=fail, ff=interpret results]
== Below data is sent by the driver if S2 is not pass ==
S3         // where: S3 is Get Automatic Shorts Self Test Results
           // status byte [0=pass, 1=fail]
ID         // where: ID is Auto Shorts Self-Test ID (4)
LL         // where: LL is length of returned data (7:0)
LL         // where: LL is length of returned data (15:8)
DD         // where: DD is the data in the same byte order as read from the device
DD
:
```

9.1.4 Run Opens Test

The Run Opens Test will stop the normal touch scanning using the firmware Stop Scanning command. Then the test will perform the firmware Opens command. Then the test will call the firmware Retrieve Opens Data command. If the test summary result is a pass then the summary results are returned to the user application. If the summary result is a fail, then the test will return the summary results along with the actual return data which provides information to identify which sensor(s) is/are open. The user application can parse this data based on the product panel configuration. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.1.4.1 Run Opens Test Host Interface

Opens:

```
> cat /path/to/mfg_test/opens
Status S    // Where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1          // where: S1 is Run Opens Self Test Command byte
            // [0=pass, 1=fail, ff=bad command]
S2          // where: S2 is test result byte [0=pass, 1=non-zero, ff=interpret results]
== Below data is sent by the driver if S2 is not a pass  ==
S3          // where: S3 is Get Opens Self- Test Results Command result byte
            // [0=pass, 1=fail, ff=interpret results]
ID          // where: ID is Opens Self-Test ID (3)
LL          // where: LL is length of returned data (7:0)
LL          // where: LL is length of returned data (15:8)
DD          // where: DD is the data in the same byte order as read from the device
DD
:
```

9.1.5 Run Calibration

The Run Calibration Test will stop the normal touch scanning using the firmware Stop Scanning command. Then the test will perform the firmware Calibration command. If the Initialize Baseline flag is set by the user in the call to this test, then the test will call the firmware Initialize Baseline command. The test will return the pass/fail results for both the Calibrate and Initialize Baseline commands. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.1.5.1 Run Calibration Host Interface

Calibrate:

```
> echo "ID,BB" > /path/to/mfg_test/calibrate
            // where: ID is the scan mode in the range
            //  [0=MC, 1=BTN, 2=SC]
            // where: BB is Perform baseline initialize flag
            //  [0=no-init, 1=Perform]
> cat /path/to/mfg_test/calibrate
Status S    // Where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1          // where: S1 is calibration status byte [0=pass, 1=fail]
== Below is valid if Perform Baseline Initialization after Calibration is Perform ==
S2          // where: S2 is initialize baseline status [0=pass, 1=fail]
```

9.1.6 Initialize Baselines Test

The Initialize Baselines Test will stop the normal touch scanning using the firmware Stop Scanning command. Then the test will perform the firmware Initialize Baseline command. The test will return the pass/fail results for Initialize Baseline command. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.1.6.1 Initialize Baseline Host Interface

Init Baselines:

```
> echo "ID" > /path/to/mfg_test/baseline
            // where: ID is the scan mode in the range:
            // [1=MC, 2=BTN, 3=MC+BTN,4=SC, 5=MC+SC,
            // 6=BTN+CS, 7=MC+BTN+SC]
> cat /path/to/mfg_test/baseline
```

```
Status S   // Where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1         // where: S1 is initialize baseline status [0=pass, 1=fail]
```

9.2 C_M/C_P In-system Manufacturing Tests

Manufacturing Tests are provided as debugfs object interfaces that are collected under a common path such as /path/to/mfg_test/cm_panel (for example, /sys/kernel/debug/4-0024/mfg_test/cm_panel as described in **Error! Reference source not found.**). The complete set of added interfaces is:

- `/path/to/mfg_test/cm_panel`
- `/path/to/mfg_test/cp_panel`
- `/path/to/mfg_test/cm_button`
- `/path/to/mfg_test/cp_button`

To understand the data that is returned by the Manufacturing Tests, refer to the application note: AN85948 Touchscreen Manufacturing Testing with TMA448/545/568 for Mutual-capacitance (C_M) and Self-capacitance (C_P) testing.

9.2.1 Mutual-capacitance (C_M) Panel Test

The C_M Panel Scan test will stop the normal touch scanning using the firmware Stop Scanning command. Then, the test will perform the firmware C_M Panel Self-Test. The test will return the C_M data for the entire panel. This data is returned as a single contiguous block of data with the length. The user application can perform range checking on this data to verify the panel C_M. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.2.1.1 C_M Panel Host Interface

C_M Panel Test:

```
> cat /path/to/mfg_test/cm_panel
Status S   //where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1         // where: S1 is Run CM Panel Self Test Command byte
           // [0=pass, 1=fail, ff=bad command]
S2         // where: S2 is test result byte [0=pass, 1=non-zero, ff=interpret results]
== Below data is sent by the driver if S2 is not a pass  ==
S3         // where: S3 is Run CM Panel Self Test Results Command result byte
           // [0=pass, 1=fail]
ID         // where: ID is CM Panel Self-Test ID (5)
LL         // where: LL is length of returned data (7:0)
LL         // where: LL is length of returned data (15:8)
DD         // where: DD is the Panel CM data in the byte order read from the device
DD
:
```

9.2.2 Self-capacitance (C_P) Panel Test

The C_P Panel Scan test will stop the normal touch scanning using the firmware Stop Scanning command. Then, the test will perform the firmware C_P Panel Self-Test. The test will return the C_P data for the entire panel. This data is returned as a single contiguous block of data with the length. The user application can perform range checking on this data to verify the panel C_P. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.2.2.1 C_P Panel Host Interface

C_P Panel Test:

```
> cat /path/to/mfg_test/cp_panel
Status S   //where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1         // where: S1 is Run CP Panel Self Test Command byte
```

```
            // [0=pass, 1=fail, ff=bad command]
S2          // where: S2 is test result byte [0=pass, 1=non-zero, ff=interpret results]
== Below data is sent by the driver if S2 is not a pass  ==
S3          // where: S3 is Run CP Panel Self Test Results Command result byte
            // [0=pass, 1=fail]
ID          // where: ID is CP Panel Self-Test ID (5)
LL          // where: LL is length of returned data (7:0)
LL          // where: LL is length of returned data (15:8)
DD          // where: DD is the Panel CP data in the byte order read from the device
DD
:
```

9.2.3 Mutual-capacitance (C_M) Buttons Test

The C_M Buttons Scan test will stop the normal touch scanning using the firmware Stop Scanning command. Then, the test will perform the firmware C_M Buttons Self-Test. The test will return the C_M data for each Mutual-capacitance or Hybrid button. This data is returned as a single contiguous block of data with the length. The user application can perform range checking on this data to verify the C_M buttons. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.2.3.1 C_M Button Host Interface

C_M Button Test:

```
> cat /path/to/mfg_test/cm_button
Status S    //where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1          // where: S1 is Run CM Button Self Test Command byte
            // [0=pass, 1=fail, ff=bad command]
S2          // where: S2 is test result byte [0=pass, 1=non-zero, ff=interpret results]
== Below data is sent by the driver if S2 is not a pass  ==
S3          // where: S3 is Run CM Button Self Test Results Command result byte
            // [0=pass, 1=fail]
ID          // where: ID is CM Button Self-Test ID (5)
LL          // where: LL is length of returned data (7:0)
LL          // where: LL is length of returned data (15:8)
DD          // where: DD is the Panel CM data in the byte order read from the device
DD
:
```

9.2.4 Self-capacitance (C_P) Buttons Test

The C_P Buttons Scan test will stop the normal touch scanning using the firmware Stop Scanning command. Then, the test will perform the firmware C_P Buttons Self-Test. The test will return the C_P data for each Self-capacitance or Hybrid button. This data is returned as a single contiguous block of data with the length. The user application can perform range checking on this data to verify the C_P buttons. Finally, the test will call the firmware Resume Scanning command to restore normal touch processing.

9.2.4.1 C_P Button Host Interface

C_P Button Test:

```
> cat /path/to/mfg_test/cm_button
Status S    //where: S is [-1=bad command, 0=pass]
== Below data is sent by the driver if Status S is pass ==
S1          // where: S1 is Run CP Button Self Test Command byte
            // [0=pass, 1=fail, ff=bad command]
S2          // where: S2 is test result byte [0=pass, 1=non-zero, ff=interpret results]
== Below data is sent by the driver if S2 is not a pass  ==
S3          // where: S3 is Run CP Button Self Test Results Command result byte
            // [0=pass, 1=fail]
ID          // where: ID is CP Button Self-Test ID (5)
LL          // where: LL is length of returned data (7:0)
LL          // where: LL is length of returned data (15:8)
DD          // where: DD is the Panel CP data in the byte order read from the device
```

```
DD
:
```

9.3 Range Checking C_M/C_P Manufacturing Test

The TTDA 3.7 adds range checking capability to the TTDL C_M/C_P manufacturing test capability. This is a new test for C_M/C_P as an alternative test to the C_M/C_P test described in the CM/CP In-system Manufacturing Tests above. This test takes in range limits from a user specified file in standard CSV (Comma Separated Values) format and returns pass/fail indications and detailed range checking results. These results are also in CSV format and can be redirected by the user to a file for use in spreadsheet analysis. These files are compatible with the Data Analyzer tool used by the Parade Manufacturing Test Engineering support group to create limits and statistically analyze testing results.

9.3.1 Range Checking C_M/C_P Host Interface

The Range Checking C_M/C_P test is initiated by writing the test control options to a system (sysfs) object located in the driver device path "/path/to/sysfs". This path is typically in the form of "/sys/dev/i2c/devices/4-0024" where the "4-" is the I2C host bus (bus=4 for PandaBoard development kit) and the "0024" is the TrueTouch device I2C address in hex format (0x24 is typical in sample firmware)..

9.3.1.1 Run the Test

The test is run by echoing command options to the test sysfs object: cmcp_test. Then the same object is read using a cat operation. If the cat returns success status, then the test results can be read from the debugfs object: cmcp_results using another cat operation and redirect the output to a file. The result data in the file is in CSV format and is suitable for analyzing in a spreadsheet tool.

In order to perform range checking of the firmware generated C_M/C_P results data, the cmcp_test takes range information from a CSV formatted file specified either with the firmware class or as loaded manually using other sysfs objects: cmcp_threshold_loading and cmcp_threshold_data. The loading of the file is controlled with commands echoed to the _loading object and by cat'ing the file to the _data object. If a file has not been loaded either automatically or manually, then range checking is not performed.

```
> cd /sys/bus/i2c/devices/4-0024
> echo "X, Y, Z" > cmcp_test              // X, Y, and Z are as defined below in:
                                          // Table 9-1. CM/CP Range Testing Options
```

Table 9-1. C_M/C_P Range Testing Options

X=Test ID	Y = Full or Basic Range Checking (Optional if Z is not specified)	Z = Calibrate When Testing (Optional)
0 = Perform all Tests	0 = Full (default)	0 = Calibrate When Testing (default)
1 = C_M Panel with Gradient	1 = Basic	1 = No Calibration
2 = C_P Panel		
3 = C_M Button		
4 = C_P Button		

9.3.1.2 Get the Test Status

At any time after starting the test run, the current status of the testing can be observed by cat'ing the test sysfs object:

```
> cat cmcp_test                         // get the test status
```

9.3.1.2.1 Status – Not Ready

In order to support a polling mechanism, the cmcp_test will respond to a cat read with a non-ready status:

```
> cat cmcp_test                         // get the test status
Status 0                                // response - test running, results not ready
```

9.3.1.2.2 Status – Ready – Test Results Available

When the cat responds with a ready status, test results available, then the host cat read the cmcp_threshold_results for the complete test results.

```
> cat cmcp_test                          // get the test status
Status 1                                 // response – test finished, results available
```

9.3.1.2.3 Status – Ready – Test Fail

If the test fails to run completely, then the cat responds with a ready signal, but not test results are available. The ready signal is an error code.

```
> cat cmcp_test                          // get the test status
Status N                                 // response – test not run, no results available
                                         // followed by error string text
```

Table 9-2. C_M/C_P Range Check Test Status Codes

Code	Status	Corrective Action	Error String
0	Not Ready	Wait and try again	N/A – no error
1	Ready – Test Complete, all threshold range checking passed.	N/A – no action needed.	N/A – no error
2	Ready – Test Aborted Mismatch:	Retry with corrected input range file	“Input cmcp threshold file mismatches with FW”
3	Ready – Test Aborted Invalid Item - Button	Retry with corrected Firmware	“FW doesn't support button!.”
4	Ready – Test Aborted Invalid Item	Retry with corrected command parameters	“Wrong test item or range check input! Only support below items: 0 - Cm/Cp Panel & Button with Gradient (Typical) 1 - Cm Panel with Gradient 2 - Cp Panel 3 - Cm Button 4 - Cp Button Only support below range check: 0 - Full Range Checking (default) 1 - Basic Range Checking(TSG5 style)”
5	Ready – Test Aborted Unsupported Test ID	Retry with corrected command parameters	“get self test ID not supported!”
6	Ready – Test completed, threshold failures.	Read response at cmcp_threshold_results	N/A – no error

9.3.1.3 Example Test Run Initiations

```
> echo "0" > cmcp_test                   // perform all tests with full range checking
                                         // and with calibration
> echo "0, 1" > cmcp_test                // perform all tests with basic range checking
                                         // and with calibration
> echo "0, 0, 1" > cmcp_test             // perform all tests with full range checking
                                         // and no calibration
> echo "1, 0, 0" > cmcp_test             // perform CM Panel test with full range
```

```
// checking and with calibration
```

9.3.1.4 Input the Range Limits – Input Range Limit File

The range limits for the range checking feature of the C_M/C_P Test are entered with a file. The file is formatted as CSV and provides the named ranges (Table 9-3. CM/CP Range Testing Limits). If a range is not specified in the file, then the checking corresponding to that range is not checked. If no range limit file is specified or loaded, then no range checking is performed.

Table 9-3. C_M/C_P Range Testing Limits

Item	Name	Description	Range Checking		Panel		Button	
			Full	Basic	C_M	C_P	C_M	C_P
1	CM_EXCLUDING_COL_EDGE	Gradient Neighbor Uniformity	✓	✓	✓	—	—	—
2	CM_EXCLUDING_ROW_EDGE	Gradient Neighbor Uniformity	✓	✓	✓	—	—	—
3	CM_GRADIENT_CHECK_COL	Gradient Neighbor Uniformity	✓	✓	✓	—	—	—
4	CM_GRADIENT_CHECK_ROW	Gradient Neighbor Uniformity	✓	✓	✓	—	—	—
5	CM_RANGE_LIMIT_ROW	Panel Sensor Row Uniformity	✓	—	✓	—	—	—
6	CM_RANGE_LIMIT_COL	Panel Sensor Column Uniformity	✓	—	✓	—	—	—
7	CM_MIN_LIMIT_CAL	Calculated Calibration Level Limit	✓	—	✓	—	—	—
8	CM_MAX_LIMIT_CAL	Calculated Calibration Level Limit	✓	—	✓	—	—	—
9	CM_MAX_DELTA_SENSOR_PERCENT	Avg Sensor v. Sensor Calibration Level	✓	—	✓	—	—	—
10	CM_MAX_DELTA_BUTTON_PERCENT	Avg Button v. Button Calibration Level	✓	—	—	—	✓	—
11	PER_ELEMENT_MIN_MAX_TABLE_BUTTON	Per-element Button Range Check	✓	✓	—	—	✓	—
12	PER_ELEMENT_MIN_MAX_TABLE_SENSOR	Per-element Sensor Range Check	✓	✓	✓	—	—	—
13	CP_MAX_DELTA_SENSOR_RX_PERCENT	Avg RX Sensor v. Calibration Level	✓	—	—	✓	—	—
14	CP_MAX_DELTA_SENSOR_TX_PERCENT	Avg TX Sensor v. Calibration Level	✓	—	—	✓	—	—
15	CP_MAX_DELTA_BUTTON_PERCENT	Avg Button v. Calibration Level	✓	—	—	—	—	✓
16	MIN_BUTTON	Avg Button Range Check	✓	—	—	—	—	✓
17	MAX_BUTTON	Avg Button Range Check	✓	—	—	—	—	✓
18	PER_ELEMENT_MIN_MAX_RX	Per-element RX Sensor Range Check	✓	✓	—	✓	—	—
19	PER_ELEMENT_MIN_MAX_TX	Per-element TX Sensor Range Check	✓	✓	—	✓	—	—

There are two ways to input the range checking limits file: automatic, manual.

9.3.1.4.1 Automatic Load of Input Range Limit File

For this method, the Linux firmware class input API is used to allow the Linux to read the file from the file system and return the raw bytes from the file and the number of bytes. The byte data will be the comma separated ASCII values from the file.

For this method, the input file is named “cyttsp5_thresholdfile.csv” is either built into the host file system in the firmware class directory or is copied there as part of a user instruction during host startup. When the driver starts up, it will try to access this file. If the file is present, then it is read and the contents sent to the driver. The driver parses the data for use later during CM/CP test range checking.

9.3.1.4.2 Manual Load of Input Range Limit File

It is possible to manually enter the contents of an input range file. The driver provides two sysfs objects: cmcp_threshold_loading and cmcp_threshold_data. A file is copied to the file system from user space and then these

objects are used to read this file and send the contents to the driver. The driver parses this data same as for the automatic method above. If the automatic method was also used as part of driver startup, its parsed data will be replaced with the parsed contents from this manual file operation.

Copy the file (such as cyttsp5_thresholdfile.csv) to a user accessible file system location such as:
/data/cyttsp5_thresholdfile.csv.
Then perform the following user command line operations to read the file and load the contents to the driver:

```
> cd /sys/bus/i2c/devices/4-0024           // same sample path as described above
                                           // in paragraph 9.3.1
> echo 1 > cmcp_threshold_loading          // prepare driver to receive file contents
> cat /data/cyttsp5_thresholdfile.csv > cmcp_threshold_data  // read the file and copy its
                                                            // contents to the driver
> echo 0 > cmcp_threshold_loading          // direct driver to parse the loaded file range
                                           // limit contents
```

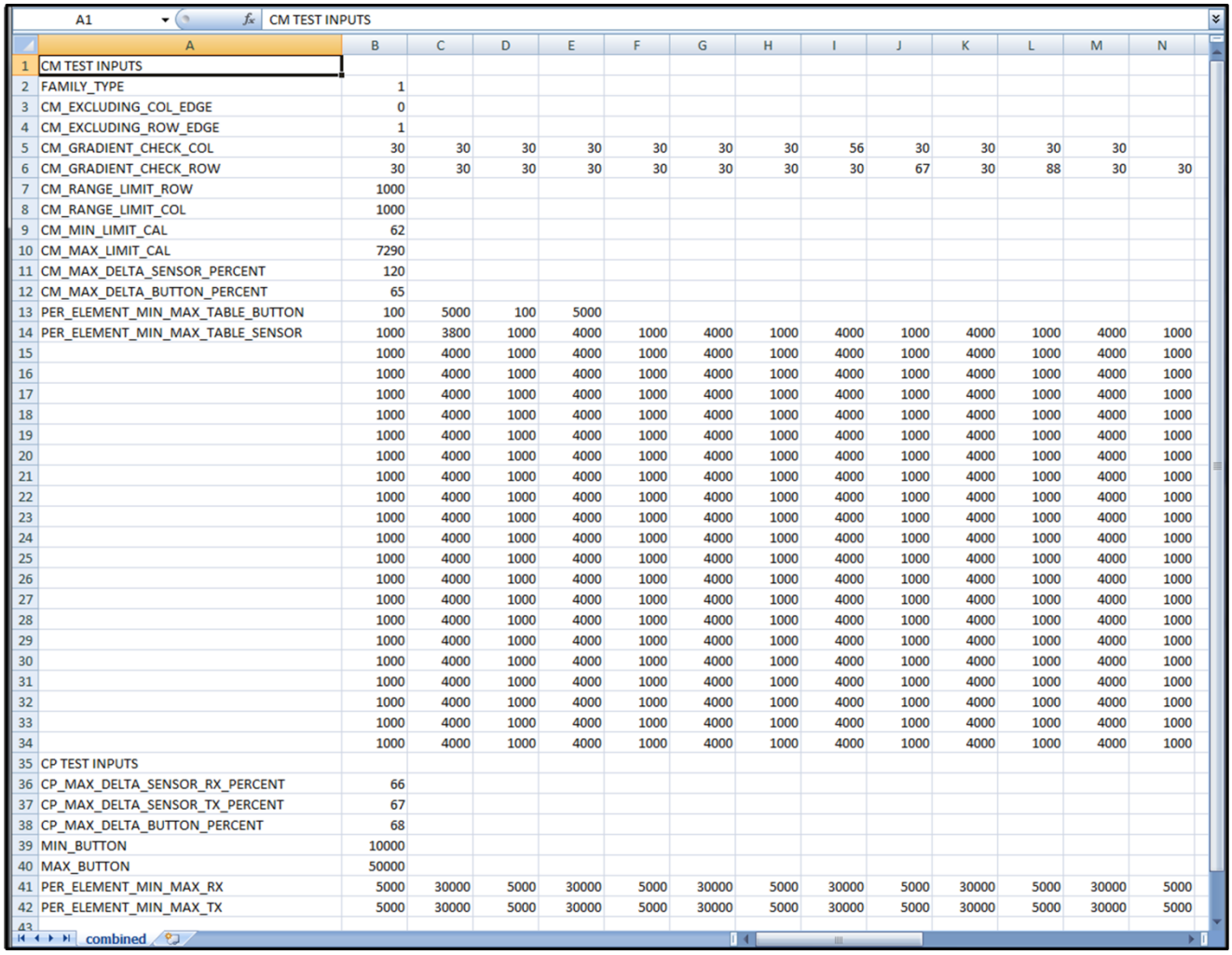
9.3.1.5 Get the Results

If the CM/CP Threshold Test ran completely and successfully, then the test range limit checking results can be read and redirected to a file for extraction and analysis. This can be done with a simple user cat command (note the out file can be any name such as “outputfile.csv”):

```
> cd /sys/kernel/debug/4-0024              // path to system debugfs and driver folder
> cat cmcp_results > /data/outputfile.csv  // write the range test results to a file
                                           // suitable for spreadsheet analysis
```

9.3.2 Sample Input Range File

9.3.2.1 Spreadsheet Partial View



A1 | CM TEST INPUTS

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	CM TEST INPUTS													
2	FAMILY_TYPE	1												
3	CM_EXCLUDING_COL_EDGE	0												
4	CM_EXCLUDING_ROW_EDGE	1												
5	CM_GRADIENT_CHECK_COL	30	30	30	30	30	30	30	56	30	30	30	30	
6	CM_GRADIENT_CHECK_ROW	30	30	30	30	30	30	30	30	67	30	88	30	30
7	CM_RANGE_LIMIT_ROW	1000												
8	CM_RANGE_LIMIT_COL	1000												
9	CM_MIN_LIMIT_CAL	62												
10	CM_MAX_LIMIT_CAL	7290												
11	CM_MAX_DELTA_SENSOR_PERCENT	120												
12	CM_MAX_DELTA_BUTTON_PERCENT	65												
13	PER_ELEMENT_MIN_MAX_TABLE_BUTTON	100	5000	100	5000									
14	PER_ELEMENT_MIN_MAX_TABLE_SENSOR	1000	3800	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
15		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
16		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
17		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
18		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
19		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
20		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
21		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
22		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
23		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
24		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
25		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
26		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
27		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
28		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
29		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
30		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
31		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
32		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
33		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
34		1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000	4000	1000
35	CP TEST INPUTS													
36	CP_MAX_DELTA_SENSOR_RX_PERCENT	66												
37	CP_MAX_DELTA_SENSOR_TX_PERCENT	67												
38	CP_MAX_DELTA_BUTTON_PERCENT	68												
39	MIN_BUTTON	10000												
40	MAX_BUTTON	50000												
41	PER_ELEMENT_MIN_MAX_RX	5000	30000	5000	30000	5000	30000	5000	30000	5000	30000	5000	30000	5000
42	PER_ELEMENT_MIN_MAX_TX	5000	30000	5000	30000	5000	30000	5000	30000	5000	30000	5000	30000	5000
43														

combined

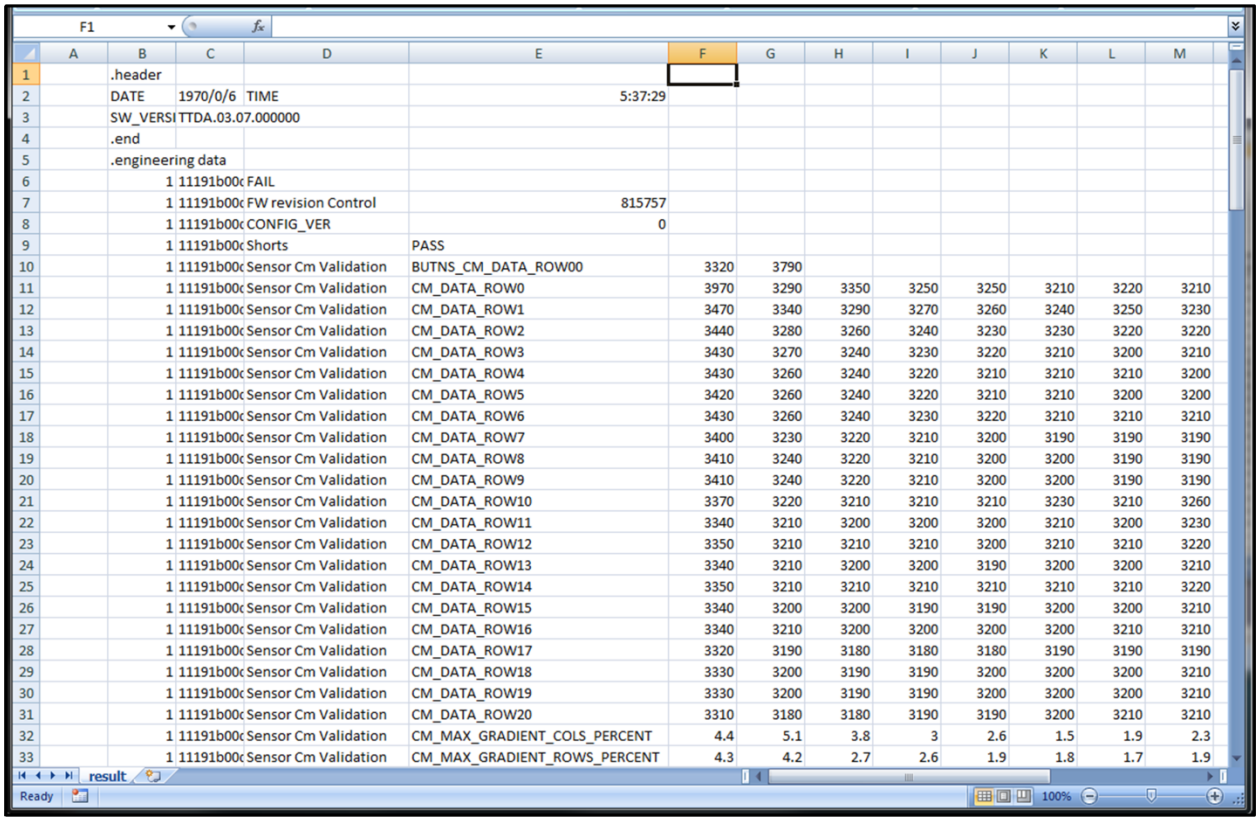
9.3.2.2 Text View

```
File  Edit  Format  View  Help
CM TEST INPUTS,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
FAMILY_TYPE,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_EXCLUDING_COL_EDGE,0,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_EXCLUDING_ROW_EDGE,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_GRADIENT_CHECK_COL,30,30,30,30,30,30,30,56,30,30,30,30,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_GRADIENT_CHECK_ROW,30,30,30,30,30,30,30,30,67,30,88,30,30,30,30,30,30,30,30,30,30,,,,,,,,,,,,,,,,,,,,
CM_RANGE_LIMIT_ROW,1000,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_RANGE_LIMIT_COL,1000,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_MIN_LIMIT_CAL,62,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_MAX_LIMIT_CAL,7290,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_MAX_DELTA_SENSOR_PERCENT,120,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CM_MAX_DELTA_BUTTON_PERCENT,65,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
PER_ELEMENT_MIN_MAX_TABLE_BUTTON,100,5000,100,5000,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
PER_ELEMENT_MIN_MAX_TABLE_SENSOR,1000,3800,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,100
0,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,1000,4000,,,,,,,,,,,,,,,,,
CP TEST INPUTS,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CP_MAX_DELTA_SENSOR_RX_PERCENT,66,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CP_MAX_DELTA_SENSOR_TX_PERCENT,67,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
CP_MAX_DELTA_BUTTON_PERCENT,68,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
MIN_BUTTON,10000,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
MAX_BUTTON,50000,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
PER_ELEMENT_MIN_MAX_RX,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,50
00,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000
PER_ELEMENT_MIN_MAX_TX,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,5000,30000,50
00,30000,,,,,,,,,,,,,,,,,
```

9.3.3 Sample Output Results File

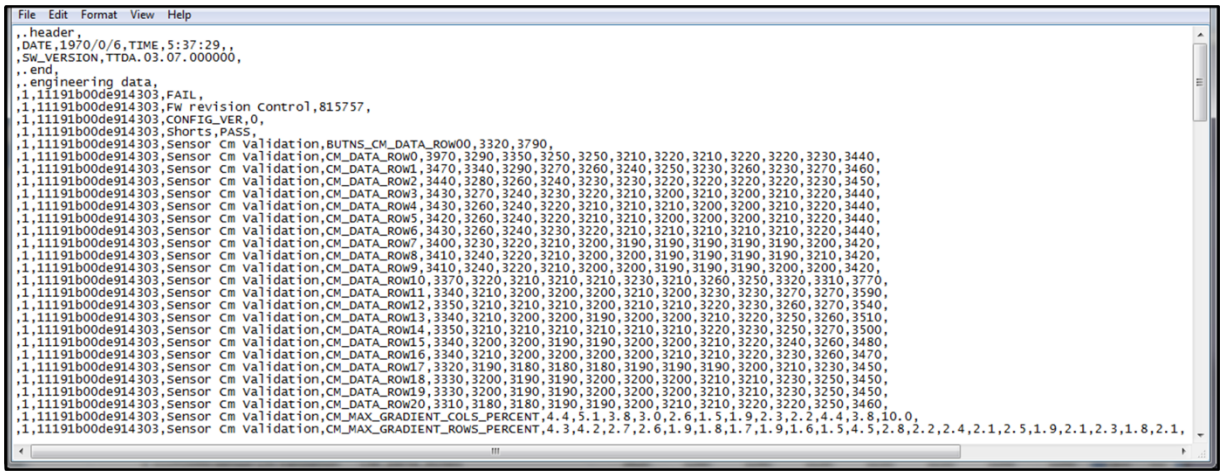
Refer to 001-63571 TrueTouch Manufacturing Test Kit User Guide Section 5 Data Log Analysis for general descriptions of the test output log file formats.

9.3.3.1 Spreadsheet Partial View



	A	B	C	D	E	F	G	H	I	J	K	L	M
1		.header											
2		DATE	1970/0/6	TIME	5:37:29								
3		SW_VERSI	TTDA.03.07.000000										
4		.end											
5		.engineering data											
6		1	11191b00(	FAIL									
7		1	11191b00(	FW revision Control	815757								
8		1	11191b00(	CONFIG_VER	0								
9		1	11191b00(	Shorts	PASS								
10		1	11191b00(	Sensor Cm Validation	BUTNS_CM_DATA_ROW00	3320	3790						
11		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW0	3970	3290	3350	3250	3250	3210	3220	3210
12		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW1	3470	3340	3290	3270	3260	3240	3250	3230
13		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW2	3440	3280	3260	3240	3230	3230	3220	3220
14		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW3	3430	3270	3240	3230	3220	3210	3200	3210
15		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW4	3430	3260	3240	3220	3210	3210	3210	3200
16		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW5	3420	3260	3240	3220	3210	3210	3200	3200
17		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW6	3430	3260	3240	3230	3220	3210	3210	3210
18		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW7	3400	3230	3220	3210	3200	3190	3190	3190
19		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW8	3410	3240	3220	3210	3200	3200	3190	3190
20		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW9	3410	3240	3220	3210	3200	3200	3190	3190
21		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW10	3370	3220	3210	3210	3210	3230	3210	3260
22		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW11	3340	3210	3200	3200	3200	3210	3200	3230
23		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW12	3350	3210	3210	3210	3200	3210	3210	3220
24		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW13	3340	3210	3200	3200	3190	3200	3200	3210
25		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW14	3350	3210	3210	3210	3210	3210	3210	3220
26		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW15	3340	3200	3200	3190	3190	3200	3200	3210
27		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW16	3340	3210	3200	3200	3200	3200	3210	3210
28		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW17	3320	3190	3180	3180	3180	3190	3190	3190
29		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW18	3330	3200	3190	3190	3200	3200	3200	3210
30		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW19	3330	3200	3190	3190	3200	3200	3200	3210
31		1	11191b00(	Sensor Cm Validation	CM_DATA_ROW20	3310	3180	3180	3190	3190	3200	3210	3210
32		1	11191b00(	Sensor Cm Validation	CM_MAX_GRADIENT_COLS_PERCENT	4.4	5.1	3.8	3	2.6	1.5	1.9	2.3
33		1	11191b00(	Sensor Cm Validation	CM_MAX_GRADIENT_ROWS_PERCENT	4.3	4.2	2.7	2.6	1.9	1.8	1.7	1.9

9.3.3.2 Text Partial View



```
File  Edit  Format  View  Help
,.header,
,DATE,1970/0/6,TIME,5:37:29,,
,SW_VERSION,TTDA.03.07.000000,
,.end,
,.engineering data,
,1,11191b00de914303,FAIL,
,1,11191b00de914303,FW revision Control,815757,
,1,11191b00de914303,CONFIG_VER,0,
,1,11191b00de914303,Shorts,PASS,
,1,11191b00de914303,Sensor Cm Validation,BUTNS_CM_DATA_ROW00,3320,3790,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW0,3970,3290,3350,3250,3250,3210,3220,3210,3220,3220,3230,3440,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW1,3470,3340,3290,3270,3260,3240,3250,3230,3260,3230,3270,3460,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW2,3440,3280,3260,3240,3230,3230,3220,3220,3220,3220,3230,3450,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW3,3430,3270,3240,3230,3220,3210,3200,3210,3200,3210,3220,3440,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW4,3430,3260,3240,3220,3210,3210,3210,3200,3200,3210,3220,3440,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW5,3420,3260,3240,3220,3210,3210,3200,3200,3200,3210,3220,3440,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW6,3430,3260,3240,3230,3220,3210,3210,3210,3210,3210,3220,3440,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW7,3400,3230,3220,3210,3200,3190,3190,3190,3190,3190,3200,3420,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW8,3410,3240,3220,3210,3200,3200,3190,3190,3190,3190,3210,3420,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW9,3410,3240,3220,3210,3200,3200,3190,3190,3190,3200,3200,3420,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW10,3370,3220,3210,3210,3210,3230,3210,3260,3250,3320,3310,3770,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW11,3340,3210,3200,3200,3200,3210,3200,3230,3230,3270,3270,3590,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW12,3350,3210,3210,3210,3200,3210,3210,3220,3230,3260,3270,3540,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW13,3340,3210,3200,3200,3190,3200,3200,3210,3220,3250,3260,3510,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW14,3350,3210,3210,3210,3210,3210,3210,3220,3230,3250,3270,3500,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW15,3340,3200,3200,3190,3190,3200,3200,3210,3220,3240,3260,3480,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW16,3340,3210,3200,3200,3200,3200,3210,3210,3220,3230,3260,3470,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW17,3320,3190,3180,3180,3180,3190,3190,3190,3200,3210,3230,3450,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW18,3330,3200,3190,3190,3200,3200,3200,3210,3210,3230,3250,3450,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW19,3330,3200,3190,3190,3200,3200,3200,3210,3210,3230,3250,3450,
,1,11191b00de914303,Sensor Cm Validation,CM_DATA_ROW20,3310,3180,3180,3190,3190,3200,3210,3210,3220,3220,3250,3460,
,1,11191b00de914303,Sensor Cm Validation,CM_MAX_GRADIENT_COLS_PERCENT,4.4,5.1,3.8,3.0,2.6,1.5,1.9,2.3,2.2,4.4,3.8,10.0,
,1,11191b00de914303,Sensor Cm Validation,CM_MAX_GRADIENT_ROWS_PERCENT,4.3,4.2,2.7,2.6,1.9,1.8,1.7,1.9,1.6,1.5,4.5,2.8,2.2,2.4,2.1,2.5,1.9,2.1,2.3,1.8,2.1,
```

10. Multi-Touch Signaling Module

Module Name	MT Module
Files	cyttsp5_mt_common.c, cyttsp5_mta.c, cyttsp5_mtb.c
Introduced in Version	TTDL 3.0
Deprecated in Version	n/a

10.1 Feature Details

Refer to the Linux multi-touch protocol, Linux event signal definitions, and the Android definition for multi-touch signaling. Links to these documents are available in the TrueTouch® Documentation section.

Protocol A (required for Alpha deliverables and Gingerbread support):

1. ABS_MT_TRACKING_ID
2. ABS_MT_POSITION_X
3. ABS_MT_POSITION_Y
4. ABS_MT_PRESSURE
    a. Z or 0 if hover.
5. ABS_MT_TOUCH_MAJOR
    a. Touch major or 1 if touch major is 0 and Z is greater than 0. If Z is 0, then touch major is 0.
6. ABS_MT_TOUCH_MINOR
    a. Touch minor or 1 if touch minor is 0 and Z is greater than 0. If Z is 0, then touch minor is 0.
7. ABS_MT_ORIENTATION
8. Input reports are signaled with either BTN_TOOL_FINGER if the touch type is Finger or Glove or with BTN_TOOL_PEN if the touch type is Stylus. These signals are used for both the button press and button release signaling for the touches.
9. The following Linux API calls are made to synchronize multi-touch signaling:
    a. Input_mt_sync: after each touch set of signals
    b. Input_sync: after all the touches. Results in Linux event handling

Protocol B (requires format for Linux Submission – ICS and later):

1. ABS_MT_TRACKING_ID – this value is assigned by the Linux event handler
2. ABS_MT_POSITION_X
3. ABS_MT_POSITION_Y
4. ABS_MT_PRESSURE
    a. Z or 0 if hover (optional).

5. ABS_MT_TOUCH_MAJOR
    a. Touch major or 1 if touch major is 0 and Z is greater than 0. If Z is 0, then touch major is 0.
6. ABS_MT_TOUCH_MINOR
    a. Touch minor or 1 if touch minor is 0 and Z is greater than 0. If Z is 0, then touch minor is 0.
7. ABS_MT_ORIENTATION
8. ABS_MT_SLOT
    a. ABS_MT_TOOL_TYPE
        i. MT_TOOL_FINGER: Finger or Glove
        ii. MT_TOOL_PEN: Stylus
9. The following Linux API call is made to synchronize multi-touch signaling:
    a. Input_sync: after all the touch signals. Results in Linux event handling.

The following tables provide device to O/S signal mapping. All device signals are translated and sent to the O/S as type "int".

Note: TTDL obtains the report structure from the Report Descriptor register and does not use hardcoded values. Typical offset and size values are shown in Table 10-1. Protocol A and Table 10-2. Protocol B.

.

Table 10-1. Protocol A

Item	Device Signal	O/S Signal	Signal Evaluation
1	Touch ID	ABS_MT_TRACKING_ID	T = (int)tch_rec[1] & 0x1F
2	X	ABS_MT_POSITION_X	X = (int)tch_rec[3] << 8 + tch_rec[2]
3	Y	ABS_MT_POSITION_Y	Y = (int)tch_rec[5] << 8 + tch_rec[4]
4	Z	ABS_MT_PRESSURE	P = (int)tch_rec[6]
5	Event ID	N/A	E = (int)tch_rec[1] >> 5 & 0x03
6	Major	ABS_MT_TOUCH_MAJOR	M = (int)tch_rec[7]
7	Minor	ABS_MT_TOUCH_MINOR	m = (int)tch_rec[8]
8	Orientation	ABS_MT_ORIENTATION	O = (int)tch_rec[9]

Table 10-2. Protocol B

Item	Device Signal	O/S Signal	Signal Evaluation
1	Touch ID	ABS_MT_SLOT	T = (int)tch_rec[1] & 0x1F
2	X	ABS_MT_POSITION_X	X = (int)tch_rec[3] << 8 + tch_rec[2]
3	Y	ABS_MT_POSITION_Y	Y = (int)tch_rec[5] << 8 + tch_rec[4]
4	Z	ABS_MT_PRESSURE	P = (int)tch_rec[6]
5	Event ID	N/A	E = (int)tch_rec[1] >> 5 & 0x03
6	Major	ABS_MT_TOUCH_MAJOR	M = (int)tch_rec[7]
7	Minor	ABS_MT_TOUCH_MINOR	m = (int)tch_rec[8]
8	Orientation	ABS_MT_ORIENTATION	O = (int)tch_rec[9]

Touch Error Handling

- If the interrupt service detects an unexpected device initiated reset
    - A driver restart is queued

- On restart, the driver notifies startup notification subscribers.

- If a TTSP module receives a startup notification from the core, then all current touch tracks are terminated and then the module continues normally. A debug print can be posted to the kernel log.
- If a TTSP module detects that a touch report has an invalid number of touches reported:
  - If the number of touches is greater than the maximum touches reported by System Information, then the module uses the maximum touch number to handle touches. In this case, the device may be tracking more touches than can be reported. Only the touches that can be reported are converted to touch signals to the OS. An error can be posted to the kernel log or reported as a debug print.
- A Linux system timer is used to implement a watchdog to periodically ping the device using the PIP defined operational NULL command. If the device fails to respond, then the driver queues a restart.

11. Power Management

Feature Name	n/a
Files	Various
Introduced in Version	Linux Open/Close in TTDA 3.2 Linux Notifier Power Management in TTDA 3.4
Deprecated in Version	

11.1 Linux Open/Close

This feature replaces the early suspend and late resume power management with open/close callback functions. This provides ability to power on/off the device instead of putting device into low power on close. The design is to allow for LCD on/off activity to callback the open/close respectively. To fully enable this capability, the input device name must match the expected name in the product build and the target product *ueventd.rc* file must be modified to have the TTDL input device SysFs node enabled with system as owner. For example:

```
"/sys/bus/i2c/devices/4-0024/input/input* enabled 0660 system system"
```

The "pm_runtime" functions will be used to control low-power, based on running module activity. Early suspend will continue to be supported using the Linux-defined conditional compile constant: CONFIG_HAS_EARLYSUSPEND.

11.2 Linux Notifier Power Management

As an alternative to Early Suspend power management, the Linux Notifier system can be used with some target platforms. In those platforms, the LCD driver can send a notification to subscribing drivers when the LCD is put into blanking and when it is removed from blanking. The TTDL is modified to allow selecting the Notifier method if the Early Suspend method is disabled. To enable Notifier, make sure that the CONFIG_FB is enabled in the default configuration file.

Note: The platform must support CONFIG_FB. The Dragon board supports the Notifier method; however, the Panda board does not.

12. Proximity Sensing Signaling Module

Module Name	BTN Module
Files	cyttsp5_proximity.c
Introduced in Version	TTDL 3.1
Deprecated in Version	TTDL 3.7

12.1 Overview

The Proximity module subscribes to the Core module for touch-event interrupt notification. The Core calls the subscription callback for the subscribing Proximity module. The Proximity module signals the operating system with the ABS_DISTANCE input sensor signal with value either 0 (NEAR) or 1 (FAR).

The Proximity module provides an enable/disable sysfs interface to allow the user space to control turning proximity detection on and off. Proximity detection must also be disabled during suspend. The device proximity detection is enabled by setting the Proximity Bit in the device Scan Type register and disabled by clearing the bit. The sysfs interface is `/sys/bus/ttsp5/devices/cyttsp5_proximity.main_ttsp_core/enable`.

SHOW: current enable/disable state of Proximity detection

STORE: set enable/disable state of Proximity detection

Additionally, for Android compatibility, a special “sensors.c” or “sensors.cpp” file is required and must be built into the Android image. The “sensors” file contains the Hardware Abstraction Layer (HAL) interface for the Android for all supported sensor devices including Proximity detectors. Proximity is disabled by default on startup.

The Proximity module should be a built-in module so that it is available to the OS during startup if the OS needs to enable the proximity immediately. See the **Error! Reference source not found.** section.

12.2 Proximity Details

The proximity module scans the current touch report for any touch record where the touch type is Proximity. If the touch type is Proximity and the touch ID is Face, then the module signals ABS_DISTANCE with value NEAR (0). If no record has the Proximity touch type and the last touch report had a record with the touch type as Proximity, it indicates that Face has moved away from the touch device. Then the proximity module signals the ABS_DISTANCE with the value FAR (1).

To enable proximity sensing, the `Parade TrueTouch Gen5 Proximity` module must be selected as built-in in the Linux Kernel Configuration window before compiling the kernel. TTDA 3.1 must be compiled as built-in to allow the Android Sensor Framework recognize the proximity sensor.

In addition, Android requires a board/platform-specific Sensor HAL shared library (`sensors.<boardname>.so` in `/system/lib/hw/`) to make the Android Sensor Framework determine the number of sensor devices and their types, query them, and acquire data from them. The Sensor HAL library converts the Android Sensor Framework requests into sensor device kernel driver-specific requests.

The three steps to get the Android Sensor Framework functional are as follows:

1. Compile Sensor HAL library.
2. Copy Sensor HAL library into the file system.
3. Modify `init.<boardname>.rc` to change SysFs node permissions to make it accessible for Sensor HAL library

Note: The sysfs path for enabling proximity sensing for TTDL 3.1 and earlier is:
`/sys/bus/ttsp5/devices/cyttsp5_proximity.main_ttsp_core/enable`.

Write ‘1’ to this sysfs path to enable proximity sensing.

```
# echo 1 > /sys/bus/ttsp5/devices/cyttsp5_proximity.main_ttsp_core/enable
```

Write '0' to this sysfs path to disable proximity sensing.

```
# echo 0 > /sys/bus/ttsp5/devices/cyttsp5_proximity.main_ttsp_core/enable
```

Note: The sysfs path for enabling proximity sensing for TTDL 3.2 and later is:

```
/sys/bus/i2c/devices/x-0024/prox_enable
```

Write '1' to this sysfs path to enable proximity sensing.

```
# echo 1 > /sys/bus/i2c/devices/x-0024/prox_enable
```

Write '0' to this sysfs path to disable proximity sensing.

```
# echo 0 > /sys/bus/i2c/devices/x-0024/prox_enable
```

The following instructions describe how to enable proximity sensing for Android Open Source Platform (AOSP) and Linaro with TTDA 3.1 on PandaBoard. Depending on the platform, the Sensor HAL for the target project needs to be implemented accordingly.

- Step 1: Flash TMA488/5XX Base.V2 Firmware with support for proximity sensing into the touch IC

Flash the touch IC with TMA488/5XX Base.V2 firmware that supports proximity sensing. Proximity sensing is disabled by default. This is necessary to enable 10-finger APA-MC (all points accessible – mutual cap) scanning. Proximity sensing will be enabled and disabled at runtime by TTDL on requests from the Android Sensor HAL.

- Step 2: Compiling TTDA 3.1

Compile TTDA 3.1 as built-in with the `Parade TrueTouch Gen5 Proximity` module enabled.

Note: TTDA 3.1 must be compiled as ***built-in*** to make the Android Sensor Framework recognize the proximity sensor.

- Step 3: Setting AOSP for proximity sensing

The Android Sensor HAL shard library should be compiled and copied into the file system. The example here applies to PandaBoard only.

Compiling the Sensor HAL shared library with AOSP:

(1) Obtain the Sensor HAL source code.

(2) Copy `device/ti/panda/sensors` into `<aosp>/device/ti/panda` where `<aosp>` is the root AOSP folder.

(3) Add the following two lines to *device/ti/panda/device.mk* file in AOSP.

```
PRODUCT_PACKAGES := \
        sensors.panda
```

(4) Add the following three lines to the *device/ti/panda/init.omap4pandaboard.rc* file in AOSP under `on fs` section after `mount` operations.

```
# change permissions for Parade Proximity sensor
chmod 0660 /sys/bus/ttsp5/devices/cyttsp5_proximity.main_ttsp_core/enable
chown root system /sys/bus/ttsp5/devices/cyttsp5_proximity.main_ttsp_core/enable
```

(5) Compile the AOSP and load it to PandaBoard using `fastboot`.

13. Restore Parameters on Restart

Feature Name	Restore Parameters on Restart
Files	
Introduced in Version	TTDL 3.1
Deprecated in Version	n/a

13.1 Details

In the cyttsp5_core_platform_flags enumerated values there is the selection: CY_CORE_FLAG_RESTORE_PARAMETERS. This flag tells the driver to save each RAM setting during operation. If a restart is signaled and the driver needs to reset the touch chip and this flag is set, then the driver will restore each of the RAM settings that were previously set using the values that were stored during operation. Set the core flags field to include the `CY_CORE_FLAG_RESTORE_PARAMETERS` bit.

14. Stylus, Hover and Glove

Feature Name	Stylus, Hover, Glove
Files	cyttsp5_mta.c cyttsp5_mtb.c
Introduced in Version	TTDL 3.1
Deprecated in Version	Hover in version TTDL 3.8

14.1 Stylus Details

The multi-touch Protocol A module is updated to report any touch with the touch type of Stylus as BTN_TOOL_PEN. If the touch type is finger, then the BTN_TOOL_FINGER signal is sent. The multi-touch Protocol B module is updated to report any touch with the touch type of Stylus as MT_TOOL_PEN. If the touch type is Finger, then the MT_TOOL_FINGER signal is sent.

Note: For proper stylus performance, ensure that the board configuration has the following line added to the cyttsp5_abs[] table:
`"ABS_MT_TOOL_TYPE, 0, MT_TOOL_MAX, 0, 0,"`

For Device Tree booting method, ensure that the "cy,abs" property has the following string added to the field data: `"0x37 0 1 0 0"`

14.2 Hover Details

The Multi-touch Protocol A and B modules are updated to report any touch when the Touch Type is Hover as MT_TOOL_FINGER and sends ABS_MT_PRESSURE signal of 0, regardless of the touch record Z value. The Z value will be reported as ABS_DISTANCE.

14.3 Glove Details

The multi-touch Protocol A and B modules are updated to report any touch when the touch type is Glove as MT_TOOL_FINGER and sends all other touch signals same as for normal finger.

15. User Interfaces (sysfs, debugfs)

Feature Name	User Interfaces
Files	cyttsp5_core.c, cyttsp5_loader.c, cyttsp5_device_access.c
Introduced in Version	Various, starting with TTDL 3.0
Deprecated in Version	n/a

15.1 Details

When the TTDL startup probe is complete, TTDL creates several sysfs nodes to allow user interaction with the driver itself and if a Parade TrueTouch (TT) device was successfully detected, additional nodes are created to interact directly with the TT device through the driver. The following sections describe the sysfs nodes that TTDL makes available in user space.

The node tables below contain the following information:

Table Name: Describes the Linux path to where the sysfs/debugfs nodes in that table reside where the "x" in the directory name is the bus number that the Parade TT device is connected to. (Note: the bus number is determined by the physical connection in a given setup)

Node Name: The name of the virtual file

R/W: Defines if the node has read (R) capability, performed with a Linux "cat" command, or if the node has write (W) capability which is performed with a Linux "echo" command followed by specific command data.

"echo" Data for W: The data that can be written to the node using the "echo" command. (Note: this column is only applicable for nodes that support write (W)).

Functionality / Extra Data Description: A brief description of the functionality of the node and any extra notes.

TTDL Version: What version of TTDA/TTDL this node first became available

15.1.1 Sysfs Node Path: /sys/kernel/debug

Table 15-1. Available sysfs nodes in /sys/kernel/debug

Node Name	R/W	"echo" Data for W	Functionality / Extra Data Description	TTDL Version
get_panel_data (cyttsp5_device_access.c)	R/W		Retrieve panel or button heat map. (Note: Scanning will be suspended/resumed automatically)	3.1
tthe_tuner (cyttsp5_core.c)	R	n/a	Retrieve a live stream of touch data from the DUT (Note: Must enable `#define TTHE_TUNER_SUPPORT` in cyttsp5_regs.h to enable this feature)	3.1

15.1.2 Sysfs Node Path: /sys/kernel/debug/x-0024/mfg_test

Table 15-3. Available sysfs nodes in /sys/kernel/debug/x-0024/mfg_test

Node Name	R/W	"echo" Data for W	Functionality / Extra Data Description	TTDL Version
auto_shorts (cyttsp5_device_access.c)	R	n/a	Performs the firmware automatic shorts test and retrieves the shorts data	3.3
baseline (cyttsp5_device_access.c)	R/W	1	Perform a baseline on the panel and or buttons for self and or mutual	3.3
calibrate (cyttsp5_device_access.c)	R/W	1	Perform a calibration on the panel/buttons (mutual or self) with the option to perform a baseline.	3.3
cm_button (cyttsp5_device_access.c)	R	n/a	Perform the firmware Cm button self-test and retrieve the data for all the buttons.	3.7
cm_panel (cyttsp5_device_access.c)	R	n/a	Perform the firmware Cm panel self-test and retrieve the data for the full panel.	3.7
cmcp_results (cyttsp5_device_access.c)	R	n/a	Results from a successful Cm/Cp threshold test run	3.7
cp_button (cyttsp5_device_access.c)	R	n/a	Perform the firmware Cp button self-test and retrieve the data for all the buttons.	3.7
cp_panel (cyttsp5_device_access.c)	R	n/a	Perform the firmware Cp panel self-test and retrieve the data for the full panel.	3.7
get_idac 001-90126.docx(cyttsp5_device_access.c)	R/W	1	Retrieves the IDAC data from the DUT	3.3
opens (cyttsp5_device_access.c)	R	n/a	Performs the firmware opens test and retrieves the opens data.	3.3
panel_scan (cyttsp5_device_access.c)	R/W	[0 = Raw | 1 = Base | 2 = Diff]	Retrieves a single panel scan from the DUT, (Note: scanning must be suspended first)	3.3

15.1.3 Sysfs Node Path: /sys/class/firmware/x-0024

This node path, and both nodes, are created by the Linux firmware class and is only visible after they have been enabled by the following command: `echo 1 > /sys/bus/i2c/devices/x-0024/manual_upgrade`

Table 15-4. Available sysfs nodes in /sys/class/firmware/x-0024

Node Name	R/W	"echo" Data for W	Functionality / Extra Data Description	TTDL Version
data (Linux FW class)	W	Binary file	FW to be loaded into the device (Note: cat the FW binary file and redirect to this node [`cat file.bin > /sys/class/firmware/x-0024/data`])	3.1
loading (Linux FW class)	W	[0|1]	Start/Stop loading of binary FW data 1 = Prepare driver to accept binary FW image 0 = End loading and perform upgrade	3.1

15.1.4 Sysfs Node Path: /sys/bus/i2c/devices/x-0024

Table 15-5. Available sysfs nodes in /sys/bus/i2c/devices/x-0024

Node Name (Exists in Driver File)	R/W	"echo" Data for W	Functionality / Extra Data Description	TTDL Version
cmcp_test (cyttsp5_device_access.c)	R/W	1	Perform a Cm/Cp test for the panel or buttons, with or without range checking and calibration.	3.5
cmcp_threshold_data (cyttsp5_device_access.c)	W	cmcp threshold file	Load Cm/Cp threshold data into the driver. (Note: cat the threshold file and redirect to this node [`cat threshold.csv > /sys/bus/i2c/devices/x-0024/cmcp_threshold_data`])	3.5
cmcp_threshold_loading (cyttsp5_device_access.c)	W	1	Direct the driver to parse the loaded Cm/Cp threshold data.	3.5
command (cyttsp5_device_access.c)	W	Raw PIP cmd	DUT pass-through generic PIP command interface. NOTE: This node's functionality is grouped with the "status" and "response" nodes.	3.1
config_data (cyttsp5_loader.c)	W	Binary config data	Load config data into the driver (Note: the enable loading flag must first be set via the config_loading node) (Note: cat the config file and redirect to this node [`cat config.bin > /sys/bus/i2c/devices/x-0024/config_data`])	3.1
config_loading (cyttsp5_loader.c)	R/W	[-1|0|1]	W - Load TT Config data to TT device: 1 = Set the enable loading flag 0 = Ends loading of data and performs upgrade -1 = Abort update process R – Read the enable loading flag	3.1
drv_debug (cyttsp5_core.c)	W	4	Suspend TTDL from responding to the INT pin	3.1
		5	Resume, and wake TTDL to respond to the INT pin	3.1
		105	Stop the TTDL watchdog. The watchdog interval can be set with cmd 201 below.	3.1
		106	Re-start the TTDL watchdog	3.1
		107	Gracefully exit out of TTHE tuner logging	3.7
		108	Force a clear of the TTHE Tuner logging buffer	3.7
		110	Clear TTDL auto restore parameter list	3.9
		200 [0|1|2|3|4]	Set TTDL debug message level into /proc/kmsg: 0 = [QUIET] No debug messages 1 = [ERROR] Errors that will effect TTDL behavior 2 = [WARN] Warning messages, may impact TTDL behavior. (default) 3 = [INFO] Information messages 4 = [DEBUG] Detailed debug information	3.8
		201 [WD interval]	Set the TTDL watchdog timeout value in ms. a value of 0 also disables the watchdog. (1000ms is the default)	3.8

		202 [0|1]	Enable/disable the addition of timestamps to other TTDL sysfs nodes. (tthe_tuner, panel_scan) 0 = Disable timestamps (default) 1 = Enable timestamps	3.8
		203 [0|1]	Force PIP2.0 mode flag (TC33xx only) 0 = TTDL operates in standard PIP mode 1 = TTDL operates in PIP2.0 mode	3.9
		204 [0|1]	Force flushing of the I2C bus (Used when IRQ line is stuck asserted) 0 = Read first 2 bytes to determine size of read 1 = Force a full 255 byte read	3.9
		205 [0|1]	Force the state of the TC3300 PIP2.0 length field 0 = PIP1.x and PIP2.1+ length field interpretation 1 = PIP2.0 length field interpretation	3.9
dut_debug (cyttsp5_core.c)	W	49	Send a BL PIP (0x31) “Verify Application Integrity” bootloader cmd to the DUT. Valid in BL mode only.	3.1
		50	Send a HID “RESET” cmd to the DUT (RAM parameters restored as per section: Restore Parameters on Restart)	3.1
		53	Send a HID “SET_POWER” - ON cmd to the DUT	3.1
		54	Send a HID “SET_POWER” - SLEEP cmd to the DUT	3.1
		56	Send a BL PIP (0x38) “Get Bootloader Information” bootloader cmd to the DUT. Valid in BL mode only.	3.1
		57	Send a BL PIP (0x39) “Program & Verify Row” bootloader cmd to the DUT. Valid in BL mode only.	3.1
		59	Send a BL PIP (0x3B) “Launch Application” bootloader cmd to the DUT. Valid in BL mode only.	3.1
		72	Send a BL PIP (0x48) “Initiate Bootload” bootloader cmd to the DUT. Valid in BL mode only.	3.1
		98	Perform a DUT hard reset by toggling the XRES line (RAM parameters restored as per section: Restore Parameters on Restart)	3.1
		100	Send a PIP (0x00) “Ping” cmd to the DUT and validate the response.	3.1
		101	Send a PIP (0x01) “Start Bootloader” cmd to the DUT and validate the response	3.1
		102	Send a PIP (0x02) “Get System Information” cmd to the DUT, verify response and reload TTDL system information variables.	3.1
		103	Send a PIP (0x03) “Suspend Scanning” cmd to the DUT and verify response.	3.1
		104	Send a PIP (0x04) “Resume Scanning” cmd to the DUT and verify response.	3.1
		109	Send a HID “Get HID descriptor” cmd to the DUT	3.8
drv_irq (cyttsp5_core.c)	R/W	[0|1]	Manually disable/re-enable the COMM_INT pin interrupts to the driver	3.1
drv_ver (cyttsp5_core.c)	R	n/a	Read the TTDL version information	3.1

easy_wakeup_gesture (cyttsp5_core.c)	R/W	[0|1]	W - Enable/disable the easy wake gesture feature 0 = Disable the easy wake feature 1 = Enable the easy wake feature. R – Read the gesture that woke the system	3.2
easy_wakeup_gesture_data (cyttsp5_core.c)	R	n/a	Read the gesture data for the enabled gesture ID's in the following format: x1(LSB), x1(MSB),y1(LSB), y1(MSB),x2(LSB), x2(MSB),y2(LSB), y2(MSB),... NOTE: The EASYWAKE_TSG6 feature must enabled during compile for this node to be available	3.1
easy_wakeup_gesture_id (cyttsp5_core.c)	R	n/a	Read the gesture ID (Note: The EASYWAKE_TSG6 feature must enabled during compile for this node to be available)	3.1
forced_upgrade (cyttsp5_loader.c)	W	1	Force the driver to upgrade the FW with the stored .bin or .h FW even if the version is less than what is currently in the device. NOTE: The CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_PLATFORM_FW_UPGRADE feature build flag must be enabled for this node to be available.	3.3
hw_irq_stat (cyttsp5_core.c)	R	n/a	Read the current state of the COMM_INT pin as a digital value	3.1
hw_reset (cyttsp5_core.c)	W	1	Performs a DUT hard reset by toggling the XRES line and then forces a TTDL restart to re-enumerate with the DUT. NOTE: RAM parameters will be restored as per section: Restore Parameters on Restart)	3.1
ic_ver (cyttsp5_core.c)	R	n/a	Retrieve the following TT version information: 1- FW version major, minor, and control number 2- FW Configuration version 3- Boot loader version major and minor 4- Protocol version major and minor	3.1
manual_upgrade (cyttsp5_loader.c)	W	1	Force a manual firmware upgrade	3.1
panel_id (cyttsp5_core.c)	R	n/a	Retrieve the panel ID from the DUT	3.1
pip2_bl_status	R	n/a	Show the present complete of a PIP2 boot-load or an associated boot-load error message.	3.9
pip2_boot_mode_pin	W	[0|1]	Set the state of the TC33xx boot mode (host mode) pin: 0 = Set pin low 1 = Set pin high	3.9
pip2_flash_erase	W	[1-4]	Erase the passed in file number from the FLASH device.	3.9
pip2_get_last_error	R	n/a	Show the last boot loader error code. NOTE: The DUT will be forced to enter the BL in order to retrieve the value. If no FW image present in FLASH, a new BL will be required.	3.9
pip2_get_version	R	n/a	Show the PIP2.x version information including: PIP Version, BL Version, FW Version, Silicon ID NOTE: The FW must support this command if being used in application mode.	3.9

pip2_manual_ram_upgrade	W	1	Start the PIP2.x boot load process that will write the contents of the bin file directly to SRAM	3.9
pip2_manual_upgrade	W	1	Start the PIP2.x boot load process that will write the contents of the bin file directly to FLASH	3.9
platform_data (cyttsp5_core.c)	R	n/a	Show the internal driver platform data structure data elements	3.1
prox_enable (cyttsp5_proximity.c)	R/W	1	W - Enable proximity (Note: This node is only available if the proximity module is built in or running) R – Retrieve the prox enable count NOTE: This node is only available if the prox module is active.	3.2
response (cyttsp5_device_access.c)	R	n/a	DUT pass-through generic PIP command response bytes. Response from last cmd send in 'command' node. NOTE: This node's functionality is grouped with the "status" and "command" nodes.	3.1
sleep_status (cyttsp5_core.c)	R	n/a	Show the current sleep status	3.1
status (cyttsp5_device_access.c)	R	n/a	DUT pass-through PIP response status for the last PIP cmd sent through the 'command' node. 0 = Response not ready 1 = Response ready (Note: Does not get cleared by reading response) NOTE: This node's functionality is grouped with the "command" and "response" nodes.	3.1
t_refresh	R/W	[1-1000]	Measures the average time between x interrupts (Note: max value limited to 1000 interrupts) W – Set number of interrupts to count and start counting. R – Retrieve the following: - Number of interrupts counted - Total elapsed time in ms - Average "t-refresh" time in ms NOTE: If node is read before the requested number of interrupts have occured, the current count will be displayed and the counting will continue. Values are reset on each write to the node.	3.9
ttdl_restart (cyttsp5_core.c)	W	1	Full restart of TTDL driver probe	3.8
ttdl_bist	R	n/a	Runs the TTDL BIST that verifies the connectivity of the following nets: IRQ, TP_XRES, and HOST_MODE. Each net test can have the following results: `[  OK  ]` – Test passed, net connection good `[FAILED]` – Test failed with likely reason following `[UNTEST]` – Untested as failure occurred above	3.9

ttdl_status	R	n/a	Show internal driver flags, debug reporting level, variable states, and GPIO states. Startup Status bit mask includes the following flags: 0x0001 – BL Reset Sentinel detected 0x0002 – FW Reset Sentinel detected 0x0004 – HID Descriptor retrieved 0x0008 – Active Mode determined from HID Desc 0x0010 – Report Descriptor retrieved 0x0020 – System Information retrieved 0x0040 – Configuration CRC verified 0x0080 – Modified parameters restored 0x0100 – DUT enumeration completed (e.g. For PIP1.3 and above a good enum; Startup Status = 0x01FF) NOTE: If driver built with TTDL_DIAGNOSTICS enabled this node will also show IRQ and DUT access error counts.	3.9

15.1.5 Sysfs Examples

15.1.5.1 Using the Command/Status/Response nodes accessing a DUT on I²C bus 2

1. Suspend scanning

   ```
   adb shell
   echo "04 00 05 00 2F 00 03" > /sys/bus/i2c/devices/2-0024/command
   ```

2. Check if the response status is ready (1 = ready).

   ```
   cat /sys/bus/i2c/devices/2-0024/status
   ```

3. Get device passthrough command response bytes.

   ```
   cat /sys/bus/i2c/devices/2-0024/response
   ```

4. Touch screen and verify no touches are reported
5. Resume scanning

   ```
   echo "04 00 05 00 2F 00 04" > /sys/bus/i2c/devices/2-0024/command
   ```

6. Check if the response status is ready. (1 = ready)

   ```
   cat /sys/bus/i2c/devices/2-0024/status
   ```

7. Get device passthrough command response bytes.

   ```
   cat /sys/bus/i2c/devices/2-0024/response
   ```

16. Virtual Key Signaling

Feature Name	Virtual Keys
Files	
Introduced in Version	TTDL 3.0
Deprecated in Version	n/a

16.1 Virtual Key Overview

In mobile touchscreen applications, the active area describes the part of the touch device that actually covers the display. The system automatically reports touches within the active area. You can use parts of the touchscreen that are outside the active area as virtual keys. When you map virtual keys, you configure Android to process touches beyond the active area and parse them as key events. Hardware buttons and virtual keys look the same to applications. This process is described as “performing the mapping from touch coordinates to key codes in software” on the website. The virtual key mapping for the TTDL is hard coded with the following parameters:

- Screen resolution: X=1280, Y=720
- Touchscreen resolution: X=720, Y=1440
- Screen orientation: Flags in the board file are: CY_FLAG_FLIP(0x08), CY_FLAG_INV_X(0x10), CY_FLAG_INV_Y(0x20). Set one or more of these flags to swap X, Y axis, invert X axis, and/or invert Y axis.
- Virtual Key enable: The flag in the board file is CY_MT_FLAG_VKEYS(0x40). Add this flag to the orientation flag settings above to enable Virtual Keys (For example, 0x78 has all orientation flags and the virtual key flag set).

16.2 Virtual Key Map

The virtual key map is presented as a board property and embedded in the board configuration source file. It is visible on the board sysfs and has the following syntax:

0x01 <Linux key code> <centerX> <centerY> <width> <height>

where:

0x01: The version code (must always be 0x01)

<Linux key code> : The Linux key code of the virtual key

<centerX>: The X pixel coordinate of the center of the virtual key

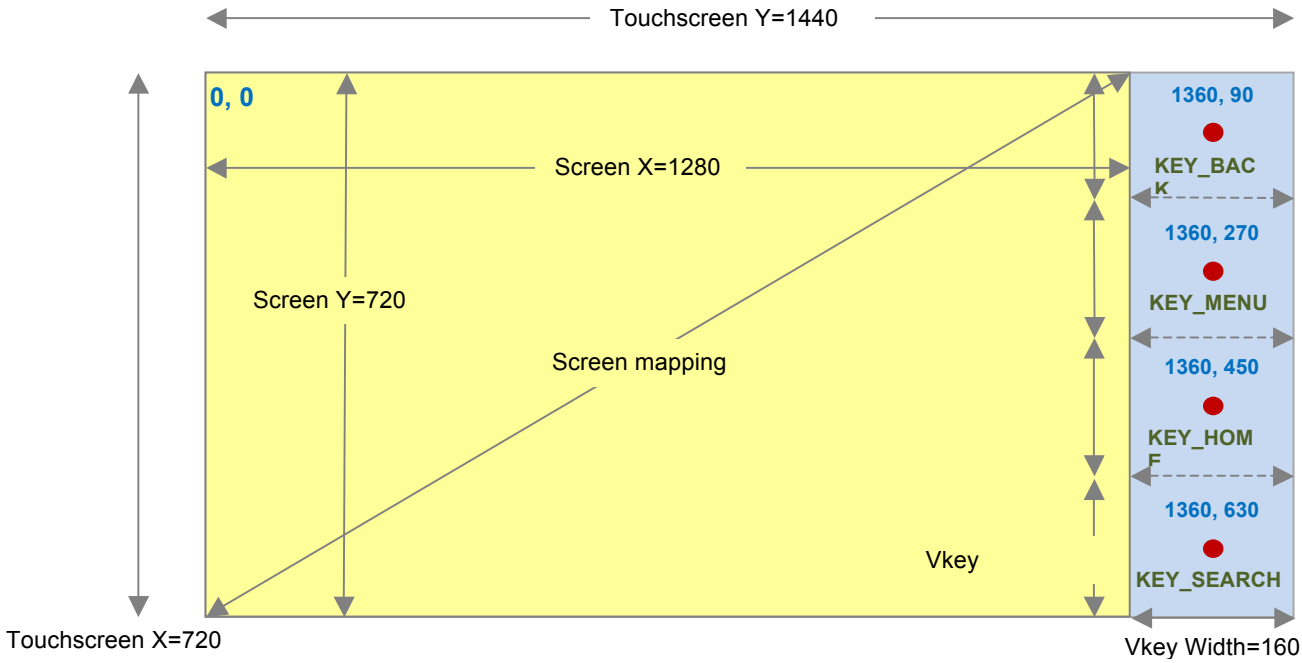
<centerY>: The Y pixel coordinate of the center of the virtual key

<width>: The width of the virtual key in pixels

<height>: The height of the virtual key in pixels

Note The virtual keys mapping file */sys/board_properties/virtualkeys.cyttsp5_mt* (TTDL 3.X) is a runtime-generated object. You can modify the name of this object by modifying the board configuration source file *kernel/arch/arm/mach-omap2/board-omap4panda.c.* The name of this file should be set the same as the Input device name *(cyttsp5_mt* is the input device name in the driver files (reference the macro #define CYTTSP5_MT_NAME “cyttsp5_mt”). The system will look for the key map and key layout files using this name *(cyttsp_mt.kcm, cyttsp5_mt.kl).* You can modify the content of these files according to the mapping of the product build.

Figure 16-1. Example Virtual Key Mapping



For the touchscreen application shown in Figure 16-1, the virtual key map is:

0x01:158:1360:90:160:180:0x01:139:1360:270:160:180:0x01:102:1360:450:160:180:0x01:217:1360:630:160:180

16.3 Implementing Virtual Keys

If you use the Device Tree, code changes that modify flag values only within the Device Tree file require rebuilding the Device Tree. All other changes to driver and/or board configuration code require that the driver and kernel are rebuilt.

1. Update the board configuration source file with the virtual key mapping. The board configuration source file for the PandaBoard is *kernel/arch/arm/mach-omap2/board-omap4panda.c.*

   Reduce the active area of the touchscreen reported to Android to leave room for the virtual keys. For example, change *_cyttsp5_mt_platform_data.flags* from 0x38 to 0x78 (set the CY_MT_FLAG_VKEYS virtual keys enable).

   This will reduce the maximum resolution of the touchscreen by an area equal to CY_VKEYS_X by CY_VKEYS_Y. These parameters are specified in the sample board configuration file (*kernel/arch/arm/mach_omap2/board-omap4panda.c*) and can be set by the defines CY_MAXX and C_MAXY respectively. For the device tree method, the values are coded into the vkeys_x and vkeys_y fields respectively in the device tree file (*kernel/arch/arm/boot/dts/omap4_panda.dts*).

   Note: If virtual keys are not enabled, then the driver will use the maximum X and Y values as defined in the device firmware system information for X, Y resolution.

2. Update the touchscreen resolution in firmware using the TTHE.

   RES_X = 720

   RES_Y = 1440

3. Update the board configuration with the key layout and key character map files.

   Copy *<development_sandbox>/samples/cyttsp5_mt.kl* to board: */system/usr/keylayout/cyttsp5_mt.kl*

   Copy *<development_sandbox>/samples/cyttsp5_mt.kcm* to board: */system/usr/keychars/cyttsp5_mt.kcm*

   Use ADB to push these configuration files to the board:

   ```
   cd <development_sandbox>/samples
   adb remount
   adb push cyttsp5_mt.kl  /system/usr/keylayout/cyttsp5_mt.kl
   ```

```
adb push cyttsp5_mt.kcm  /system/usr/keychars/cyttsp5_mt.kcm
```

Section C: Customization and Debugging

This section documents information about TTDL driver customization, including driver porting instructions, methods to enable virtual keys, driver modification for system without XRES signal. It also provides driver debugging information on various ADB commands to perform during product development and evaluation, firmware upgrade instruction, IDAC calibration and open/short tests scripts. Mobile tuner setup instruction and system debugging tools description are also captured in this section.

Section B contains the following chapters:

- Driver Porting (Linux Board Configuration)
- Driver Porting (Device Tree)
- Firmware Updates
- IDAC Calibration
- Driver Without XRES Signal
- System Debugging
- Mobile Tuner

17. Driver Porting (Linux Board Configuration)

This chapter describes the driver porting instructions for projects that employ the Linux Board Configuration option. For platforms that use the Device Tree option, see Driver Porting (Device Tree).

17.1 Overview

TTDL provides easy ways of driver porting using the Linux board configuration option. Most customization in TTDL can be achieved by modifying the Linux board configuration file. The following settings are available in the Linux board configuration file:

- Setting GPIO for I2C/SPI and XRES line
- Init function setting up XRES and IRQ
- Wakeup/sleep functions
- IRQ status function
- Setting up virtual buttons
- Screen size, major axis, minor axis, orientation settings
- Registering Core, MT, and Button devices
- Initializing I^2C or SPI

17.2 TTDL Porting Example

Take the board configuration file */kernel/arch/arm/mach-omap2/board-omap4panda.c* that comes with the TTDL release as an example. The following code pieces show part of the drive porting steps for PandaBoard.

- Setting GPIO for I^2C/SPI and XRES lines

Note GPIO numbers must be different when both I^2C and SPI are on. I^2C/SPI addresses may vary from design to design. Following is an example of the PandaBoard configuration.

```
#ifdef CYTTSP5_USE_I2C
#define CYTTSP5_I2C_NAME "cyttsp5_i2c_adapter"
#define CYTTSP5_I2C_TCH_ADR 0x24
#define CYTTSP5_LDR_TCH_ADR 0x24
#define CYTTSP5_I2C_IRQ_GPIO 38 /* Customization item */
#define CYTTSP5_I2C_RST_GPIO 37 /* Customization item */
#endif
#ifdef CYTTSP5_USE_SPI
#define CYTTSP5_SPI_IRQ_GPIO 38 /* Customization item */
#define CYTTSP5_SPI_RST_GPIO 37 /* Customization item */
#endif
static int cyttsp5_xres(struct cyttsp5_core_platform_data *pdata,
          struct device *dev)
{
    int rst_gpio = pdata->rst_gpio;
    int rc = 0;
    gpio_set_value(rst_gpio, 1);
    msleep(20);
    gpio_set_value(rst_gpio, 0);
    msleep(40);
```

```
    gpio_set_value(rst_gpio, 1);
    msleep(20);
    dev_info(dev,
            "%s: RESET CYTTSP gpio=%d r=%d\n", __func__,
            pdata->rst_gpio, rc);
    return rc;
}
```

- Init function setting up XRES and IRQ

  In `static int cyttsp5_init(struct cyttsp5_core_platform_data *pdata, int on, struct device *dev)` function,

  Initialize XRES pin: `int rst_gpio = pdata->rst_gpio;`

  Initialize IRQ pin: `int irq_gpio = pdata->irq_gpio;`

  Request XRES pin: `gpio_request(rst_gpio, NULL);`

  Set XRES pin as output GPIO: `gpio_direction_output(rst_gpio, 1);`

  Request IRQ pin: `gpio_request(irq_gpio, NULL);`

  Set IRQ pin as input GPIO: `gpio_direction_input(irq_gpio);`

- Wakeup/Sleep functions

  The actual function for wakeup/sleep is cyttsp5_power(), which calls cyttsp5_wakeup() or cytssp4_sleep() depending on the parameter "on".

```
static int cyttsp5_power(struct cyttsp5_core_platform_data *pdata,
          int on, struct device *dev, atomic_t *ignore_irq)

static int cyttsp5_wakeup(struct cyttsp5_core_platform_data *pdata,
          struct device *dev, atomic_t *ignore_irq)

static int cyttsp5_sleep(struct cyttsp5_core_platform_data *pdata,
          struct device *dev, atomic_t *ignore_irq)
```

- IRQ status function

```
static int cyttsp5_irq_stat(struct cyttsp5_core_platform_data *pdata,
          struct device *dev)
{
    return gpio_get_value(pdata->irq_gpio);
}
```

- Setting up virtual keys

  Set cyttsp5_mt_platform_data properly to enable or disable virtual keys. See **Error! Reference source not found.** for more details.

```
static struct cyttsp5_mt_platform_data _cyttsp5_mt_platform_data = {
    .frmwrk = &cyttsp5_framework,
    .flags = 0x38,
    .inp_dev_name = CYTTSP5_MT_NAME,
};
```

  Here, the .flags = 0x38 indicates that CY_FLAG_FLIP is enabled, CY_FLAG_INV_X is enabled, CY_FLAG_INV_Y is enabled, and virtual keys CY_FLAG_VKEYS is disabled. To enable virtual keys, the flags need to be set to 0x78 for this configuration. The CY_MT_FLAG_NO_TOUCH_ON_LO tells the driver to kill all touch tracks while Large Object is detected.

```
enum cyttsp5_flags {
    CY_FLAG_NONE = 0x00,
    CY_FLAG_HOVER = 0x04,
    CY_FLAG_FLIP = 0x08,
    CY_FLAG_INV_X = 0x10,
    CY_FLAG_INV_Y = 0x20,
    CY_FLAG_VKEYS = 0x40,
    CY_MT_FLAG_NO_TOUCH_ON_LO = 0x80, /* TTDA 3.0.1 and above */
};
```

- Screen size, major axis, minor axis, and orientation settings

```
#define CY_MAXX 880
#define CY_MAXY 1280
#define CY_MINX 0
#define CY_MINY 0
```

- Registering Core, MT, and Button devices

In `static void __init omap4_panda_cyttsp5_init(void)` function,

```
/* Register core and devices (TTDA 3.1 and below) */
cyttsp5_register_core_device(&cyttsp5_core_info);
cyttsp5_register_device(&cyttsp5_mt_info);
cyttsp5_register_device(&cyttsp5_btn_info);
```

- Initializing I^2C or SPI

```
/* Initialize muxes for GPIO pins */
#ifdef CYTTSP5_USE_I2C
    omap_mux_init_gpio(CYTTSP5_I2C_RST_GPIO, OMAP_PIN_OUTPUT);
    omap_mux_init_gpio(CYTTSP5_I2C_IRQ_GPIO, OMAP_PIN_INPUT_PULLUP);
#endif
#ifdef CYTTSP5_USE_SPI
    omap_mux_init_gpio(CYTTSP5_SPI_RST_GPIO, OMAP_PIN_OUTPUT);
    omap_mux_init_gpio(CYTTSP5_SPI_IRQ_GPIO, OMAP_PIN_INPUT_PULLUP);
#endif
```

18. Driver Porting (Device Tree)

This chapter describes the driver porting instructions for projects that employ the Device Tree method. For platforms that use the Linux Board Configuration option, see the section Driver Porting (Linux Board Configuration).

18.1 TTDL Porting

Use the following procedure to set up the device tree build for the end device using TTDA 3.1 or greater:

- Step 1. Copy TTDL 3.x source files into your kernel directory.
- Step 2. Modify Device Tree files according to the end system.
  - Related files include the *omap4-panda.dts* file in the *…\kernel\arch\arm\boot\dts\omap4-panda.dts* directory and the *cyttsp5_platform.c* file in the *…\kernel\drivers\input\touchscreen\cyttsp5_platform.c* directory.
  - The *omap4-panda.dts* file is an example Device Tree file that contains hardware-specific configuration of the Parade driver development platform PandaBoard. To set up a build for a different platform, you should create a similar device tree file that is specific to the hardware configuration of the platform used.
  - The *cyttsp5_platform.c* file contains platform-specific functions such as XRES, INIT, and Power. It also contains loader platform data that includes .h firmware, TT_CONFIG, and loader flags.

    You should modify such platform specific functions according to your design.

    If automatic firmware upgrade or TT_CONFIG update is required, make sure to `#include` the firmware header file *cyttsp5_img.h* and/or TT_CONFIG header file *cyttsp5_params.h* in the *cyttsp5_platform.c* file.
- Step 3. Enable Device Tree support in Linux menuconfig.
  - In Linux menuconfig, in addition to other necessary options, select the **Enable Device Tree support** item under "Parade TrueTouch Gen5 Touchscreen Driver."
- Step 4. Compile the kernel.
  - Compile the kernel and build an image for the end device.

19. Firmware Updates

The TTDL provides an interface to load a new firmware image into a TTSP device. You need the firmware *<project>.bin* file or the *<project>.h* file generated by the TTHE tool for either method.

These methods describe how to update the firmware image. If you flashed the TTSP device and have not made any changes, you do not have to update the firmware.

19.1 Remote Host

Note: For TTDA 3.1 and earlier:

Typical `/path/to/loader` in this chapter is: `/sys/bus/ttsp5/devices/cyttsp5_loader.main_ttsp_core`.

Typical `/path/to/firmware/class` in this chapter is:
`/sys/class/firmware/cyttsp5_loader.main_ttsp_core`

Note: For TTDA 3.2 and later:

Typical `/path/to/loader` in this chapter, for I^2C host bus products, is:
`/sys/bus/i2c/devices/4-0024`.

Typical `/path/to/firmware/class` in this chapter is:
`/sys/class/firmware/4-0024`

To update the firmware manually from a remote host, you need to have the ADB utility installed and serial debugging enabled in your device. Execute the following commands using a command window or script.

echo Start data transfer

```
adb shell "echo 1 > /path/to/loader/manual_upgrade"

adb shell "echo 1 > /path/to/firmware/class/loading"
sleep 1
```

push firmware (for example, the Firmware.bin file) to the device (typically in /data directory)

```
adb push FW.bin /data
```

echo Transfer data

```
adb shell "cat /data/Firmware.bin > /path/to/firmware/class/data"
sleep 1
```

echo Initiate flashing

```
adb shell "echo 0 > /path/to/firmware/class/loading"
```

19.2 Automatic OTA Updates

TTDL can update the firmware automatically based on revision information. The firmware to be used during the upgrade is built into the kernel that will be used during an OTA upgrade. When the Android device receives the new kernel, TTDL upgrades the firmware automatically. After the firmware is upgraded, TTDL will not upgrade the firmware again until the Android device receives another new kernel.

19.2.1 Kernel Upgrade

Both the *<project>.h* and *<project>.bin* files are generated by the Parade TrueTouch Host Emulator (TTHE) tool. If you do not know the two-byte major/minor version, inspect the *<project>.bin* file in a hexadecimal viewer. The major/minor version is the second and third byte of the file.

When new firmware is available for upgrade, build the firmware .h file into the kernel so firmware upgrade will be performed at the next device reboot.

If loading the new firmware fails, the TTDL will try again for a specific number of times. If the firmware is not successfully loaded before this number of retries, then the TTDL will stop trying and quietly wait for a restart or a manual upgrade attempt. To reset the TTDL to retry the automatic updates, repower the product to reload the driver. The maximum number of retries can be set by a constant in the *cyttsp5_loader.c* file (it is set at default of 3):

```
#define CYTTSP5_LOADER_FW_UPGRADE_RETRY_COUNT 3
```

19.2.2 ADB Push

For firmware upgrade using .bin file, a .bin file needs to be built into the kernel once. For later upgrades, you can either rebuild the kernel with new .bin file, or ADB push the new .bin file to */system/etc/firmware* (typical) directory. The firmware upgrade will be executed in the next device reboot.

19.2.3 OTA Upgrade With a .ihex File

Some customers prefer to use the .ihex file format rather than the .bin file. To support this, the following steps are required:

- Don't add the file to the kernel using "make menuconfig"
- Convert the FW .bin file to a .ihex by using the Linux command:
  `objcopy -I binary -O ihex filename.bin filename.bin.ihex`
- Modify the makefile in `/kernel/firmware` to build the filename.bin.ihex into the kernel:
  `fw-shipped-y += filename.bin.ihex`

19.2.4 Board Configuration File Change

Make the following changes to the Linux Board Configuration file:

Add the following code to the board configuration file before the cyttsp5 platform data is declared

Note: The loader flags must be set in the *cyttsp5_platform.c* file.

```
#ifdef CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_PLATFORM_FW_UPGRADE
#include "cyttsp5_fw.h"
static struct cyttsp5_touch_firmware cyttsp5_firmware = {
        .img = cyttsp5_img,
        .size = ARRAY_SIZE(cyttsp5_img),
        .ver = cyttsp5_ver,
        .vsize = ARRAY_SIZE(cyttsp5_ver),
};
#else
static struct cyttsp5_touch_firmware cyttsp5_firmware = {
        .img = NULL,
        .size = 0,
        .ver = NULL,
        .vsize = 0,
};
#endif
```

Update the platform data structure for the core to include the fw field:

```
static struct cyttsp5_core_platform_data _cyttsp5_core_platform_data = {
        .irq_gpio = CYTTSP5_I2C_IRQ_GPIO,
        .rst_gpio = CYTTSP5_I2C_RST_GPIO,
        .hid_desc_register = CYTTSP5_HID_DESC_REGISTER,
        .xres = cyttsp5_xres,
        .init = cyttsp5_init,
        .power = cyttsp5_power,
        .detect = cyttsp5_detect, /* TTDA 3.3 */
        .irq_stat = cyttsp5_irq_stat,
        .sett = {
            NULL, /* Reserved */
            NULL, /* Command Registers */
            NULL, /* Touch Report */
            NULL, /* Data Record */
            NULL, /* Test Record */
            NULL, /* Panel Configuration Record */
```

```
                NULL, /* &cyttsp5_sett_param_regs, */
                NULL, /* &cyttsp5_sett_param_size, */
                NULL, /* Reserved */
                NULL, /* Reserved */
                NULL, /* Operational Configuration Record */
                NULL, /* &cyttsp5_sett_ddata, *//* Design Data Record */
                NULL, /* &cyttsp5_sett_mdata, *//* Manufacturing Data Record */
                NULL, /* Config and Test Registers */
                &cyttsp5_sett_btn_keys, /* button-to-keycode table */
        },
        .loader_pdata = &_cyttsp5_loader_platform_data, /* TTDA 3.1 */
        .flags = 0x0, /* Core flags */
        .easy_wakeup_gesture = 0x00,  /* TTDA 3.3 */
};
```

Update loader platform data and set the flags properly. This example indicates that automatic calibration must be done following a firmware load.

```
static struct cyttsp5_loader_platform_data _cyttsp5_loader_platform_data = {
    .fw = &cyttsp5_firmware,
    .ttconfig = &cyttsp5_ttconfig,
    /* Sample shows Loader flags set for calibrate after FW load */
    .flags = CY_LOADER_FLAG_CALIBRATE_AFTER_FW_UPGRADE,
};
```

The following flags are used in the loader.

```
enum cyttsp5_loader_platform_flags {
    CY_LOADER_FLAG_NONE = 0x00,
    CY_LOADER_FLAG_CALIBRATE_AFTER_FW_UPGRADE = 0x01,
    /* Use CONFIG_VER field in TT_CFG to decide TT_CFG update */
    CY_LOADER_FLAG_CHECK_TTCONFIG_VERSION, /* TTDA 3.3 */
    CY_LOADER_FLAG_CALIBRATE_AFTER_TTCONFIG_UPGRADE, /* TTDA 3.4 */
};
```

CAUTION: The CY_LOADER_FLAG_CALIBRATE_AFTER_FW_UPGRADE must be used carefully. It allows a product to be configured to automatically perform a calibration after the firmware is upgraded. The product environment cannot be controlled when the updates are performed and it is not recommended to perform calibrations while fingers or other objects are contacting the touch sensor. It is recommended to use this selection in a factory environment only. Also, note that this automatic calibration requires that the firmware is upgraded. TT configuration only updates are not covered by this setting.

RECOMMENDATION: To test calibration in a factory environment, use an ADB connection to either use the general Device Access interface to script a calibration command (see IDAC Calibration) or, for TTDA 3.3, use the calibration command as described in Run Calibration Host Interface. The TTHE Mobile Tuner tool can be used to connect to the product and perform the calibrations.

CLARIFICATION: When the CY_LOADER_FLAG_CHECK_TTCONFIG_VERSION flag is used, the #define CY_TTCONFIG_VERSION_OFFSET must be set to the correct offset in the configuration data (this define is set in the cyttsp5_regs.h driver file). It is set by default to a location common to TSG5 devices but must be confirmed for TSG5 firmware and must be looked up and set correctly for TSG6 device firmware. It is possible to determine the offset for the configuration byte (CONFIG_VER) by locating it in three places in the Driver.h Project File for the firmware. Find the third location in the Driver.h file and use the location information minus the CONFIG_DATA_SIZE location information to determine the offset.

19.2.5 Manual Touch Parameters Update Using Binary File (Driver.bin)

TTDL provides manual touch parameters update capability via a binary Driver.bin file. The Driver.bin file can be generated by TrueTouch Host Emulator 3.0 and above.

To enable manual touch parameter update functionality, make sure to select **TT Configuration upgrade via SysFs** in Kernel menuconfig menu.

To manually update the touch parameters, follow four steps. Steps 2 to 4 are interconnected by common flag and sysfs accesses.

Note: For TTDA 3.1 and earlier:

Typical `/path/to/device` in this chapter is:
`/sys/bus/ttsp5/devices/cyttsp5_device_access.main_ttsp_core`.

Note: For TTDA 3.2 and later:

Typical `/path/to/device` in this chapter, for I²C host bus products, is:
`/sys/bus/i2c/devices/4-0024`.

1. Push Driver.bin file to the device (typically in /data directory).
   ```
   adb push Driver.bin /data
   ```
2. Write '1' to the touch parameters (TT_CFG) "Loading" sysfs interface. This step will set the Enable Loading flag.
   ```
   adb shell
   echo 1 > /path/to/device/config_loading
   ```
3. Write binary touch parameters file to (TT_CFG) "Data" sysfs interface. This step copies touch parameter file across the sysfs interface to the kernel memory space.
   ```
   cat /data/Driver.bin > /path/to/device/config_data
   ```
4. Write '0' to the touch parameters (TT_CFG) "Loading" sysfs interface. This step copies the file data to the driver. The driver writes the binary data to the touch parameter flash block and clears the Enable Loading flag.
   ```
   echo 0 > /path/to/device/config_loading
   ```
5. To abort the touch parameters update process, instead of writing '0' to the touch parameters (TT_CFG) "Loading" sysfs interface in Step 4, write '-1' to the interface. This step aborts the loading and clears the Enable Loading flag.
   ```
   echo -1 > /path/to/device/config_loading
   ```

Note 1 Until you write a '0' to activate the write to the device, the touch parameters written to the product will not match the device.

Note 2: If the product is deactivated before you write a '0' to activate the write to the device, the device will not have the updates.

There is firmware version information in the Driver.h file. Firmware version is checked before touch parameters CRC are compared. Only when firmware versions in the Driver.h file and the device match, will it proceed to the touch parameters CRC check and configuration update. This ensures that the register maps used in new touch configuration matches with the firmware in use. The following is a sample:

```
static u8 ttconfig_fw_ver[] = {0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0xF1, 0x9B,
0x01, 0x2A, 0x24, 0x90, 0x00, 0x00};
```

Table 19-1. Firmware Version Information

Offset	Sample Value	Byte Description
1	0x00	RESERVED (TTPID[1] (MSB))
2	0x00	RESERVED (TTPID[0] (LSB))
3	0x02	FW Revision (Major)
4	0x00	FW Revision (Minor)
5	0x00	FW Revision Control[7] (MSB)
6	0x00	FW Revision Control[6]
7	0x00	FW Revision Control[5]
8	0x00	FW Revision Control[4]
9	0x00	FW Revision Control[3]
10	0x06	FW Revision Control[2]
11	0xF1	FW Revision Control[1]
12	0x9B	FW Revision Control[0] (LSB)
13	0x01	Silicon ID (MSB)
14	0x2A	Silicon ID (LSB)
15	0x24	Silicon Revision ID

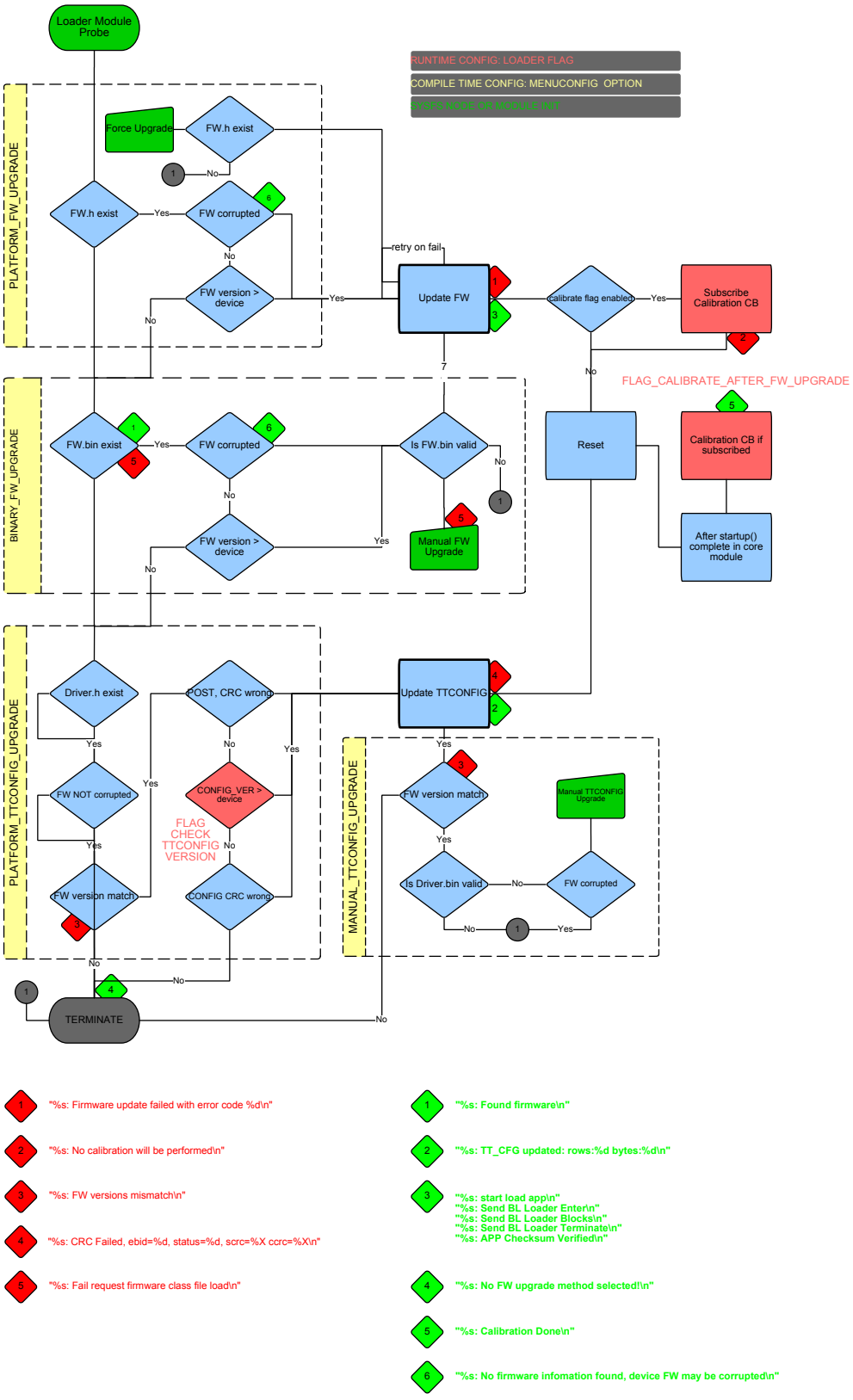
16	0x90	Silicon Family ID
17	0x00	RESERVED (TT Config Version[1] (MSB))
18	0x00	RESERVED (TT Config Version[0] (LSB))

Note The above table shows the firmware version record bytes. This record is included at the beginning of both the firmware ".h" and parameters ".h" include files. The firmware version in the array is in hex. For the sample above the FW converted to decimal) is 2.0.455067, the Silicon ID is 0x012A, the Silicon Revision ID = 0x24, and the Silicon Family ID = 0x90.

Note For the firmware ".bin" files, there is a leading byte 0 which is the firmware version record size. For the firmware ".bin" file the byte 0 equals 0x12 (18 bytes).

Note For the parameters ".bin" files, there is a leading byte 0 which is the firmware version record size. For the parameters ".bin" file the byte 0 equals 0x10 (16-bytes). The parameters ".bin" file excludes the bytes 1 and 2 (RESERVED (TTPID[1:0])).

Figure 19-1 - Rules for Automated Firmware and TT Configuration Updates



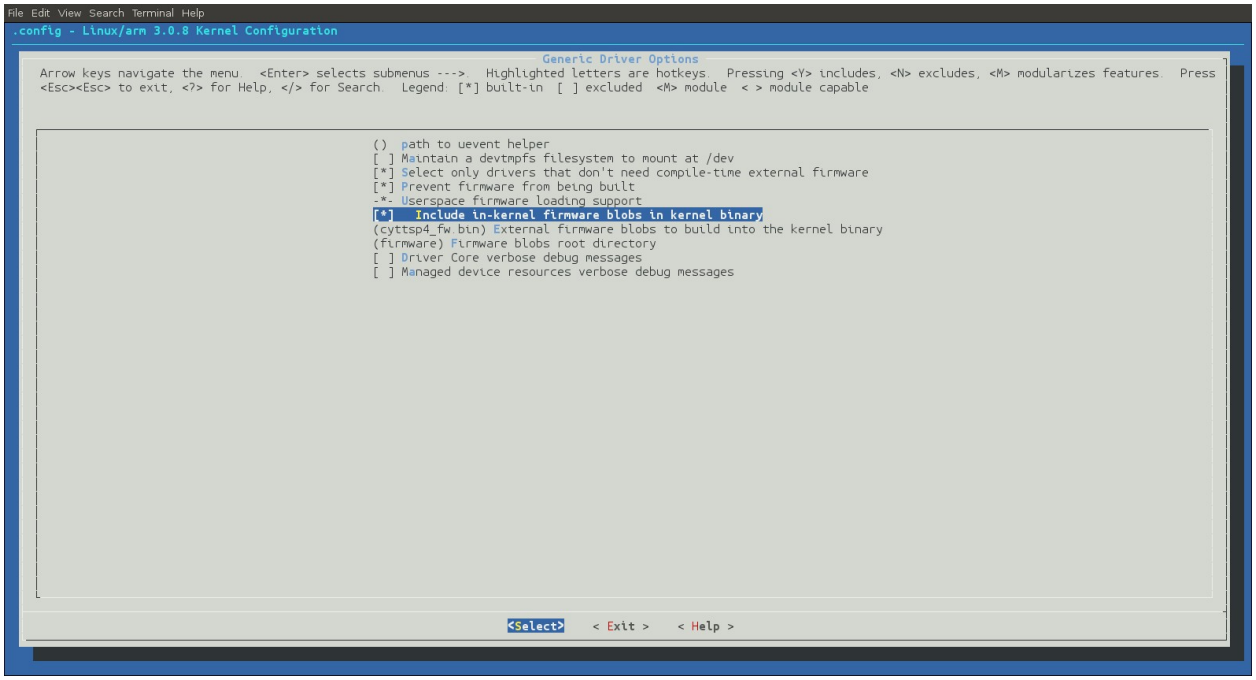
19.2.6 Create a New Kernel with *<project>.bin* File

The following steps describe how to create a new kernel:

1. Enable CONFIG_FIRMWARE_IN_KERNEL kernel configuration.

   Call “make menuconfig” and browse to **Device Drivers > General Driver Options**. Press <Y> to include **[*] Parade TrueTouch Gen5 Multitouch Loader Press** <Y> to include **[*] Include in-kernel firmware blobs in kernel binary,** as shown in Figure 19-.

   Figure 19-2. General Driver Options in menuconfig



2. Enter the firmware name *cyttsp5_fw.bin* in the CONFIG_EXTRA_FIRMWARE kernel configuration option.

   From the same General Driver Options window, enter **(cyttsp5_fw.bin) External firmware blobs to build into the kernel binary,** as shown in Figure 19-.

   Note: The file name is hard-coded to “*cyttsp5_fw.bin*.”

3. Confirm that the CONFIG_EXTRA_FIRMWARE_DIR option is set to firmware.

   From the same General Driver Options window, **(firmware) Firmware blobs root directory** should be the default value, as shown in Figure 19-.

4. Copy the firmware file into the kernel source directory.

   Copy the firmware *.bin* file into to the *<kernel>/firmware/* subdirectory, and give it the name *cyttsp5_fw.bin* .

   The file and subdirectory names must match the ones in the previous steps. You can use a different name for the kernel subdirectory; however, the firmware file name must be *cyttsp5_fw.bin*.

5. Compile the new kernel.

   Use the “make” command in the kernel source directory to compile the new kernel. Your new firmware will be included in the resulting kernel.

   If the kernel compiled correctly, the following lines should be added to kernel compile log:

```
CHK     include/linux/version.h
CHK     include/generated/utsrelease.h
make[1]: `include/generated/mach-types.h' is up to date.
```

```
CALL    scripts/checksyscalls.sh
CHK     include/generated/compile.h
AS      firmware/cyttsp5_fw.bin.gen.o
LD      firmware/built-in.o
LD      vmlinux.o
MODPOST vmlinux.o
GEN     .version
CHK     include/generated/compile.h
UPD     include/generated/compile.h
CC      init/version.o
LD      init/built-in.o
LD      .tmp_vmlinux1
KSYM    .tmp_kallsyms1.S
AS      .tmp_kallsyms1.o
LD      .tmp_vmlinux2
KSYM    .tmp_kallsyms2.S
AS      .tmp_kallsyms2.o
LD      vmlinux
```

If the firmware file was not included when the kernel compiled, it will generate a compile error similar to this:

```
CALL    scripts/checksyscalls.sh
CHK     include/generated/compile.h
make[1]: *** No rule to make target `firmware/cyttsp5_fw.bin', needed by
`firmware/cyttsp5_fw.bin.gen.o'. Stop.
make: *** [firmware] Error 2
```

19.2.7 Testing the OTA Feature Using the New Kernel

1. Boot with the new kernel.

   Put the new kernel into the SD card boot partition using fastboot, adb, or linaro tools.

2. Load the *cyttsp5_loader.ko* module.

   Loading the *cyttsp5_loader.ko* module will trigger a firmware upgrade if the firmware revision in the SD card kernel is newer than the TTSP device's current firmware. If the *cyttsp5_loader.ko* module is compiled as "built-in", an upgrade will be triggered automatically during boot up.

   To observe loader messages, *cyttsp5_loader.ko* should be compiled with debug messages enabled.

   Using the host system command line, give following commands:

   For TMA5xx devices using TTDL 3.X,

   ```
   insmod cyttsp5_i2c.ko
   ```

   ```
   insmod cyttsp5_core.ko
   ```

   ```
   insmod cyttsp5_loader.ko
   ```

   If the firmware revision in the TTSP device and the firmware revision in the SD card kernel are not the same, and the version of the firmware revision in the SD card kernel is newer than the revision in the TTSP device then loader will try to update the firmware in the TTSP device. If the load is successful, the following information should be printed into the kernel log:

   ```
   cyttsp5_load_app: Send BL Loader Enter
   ```

   ```
   cyttsp5_load_app: Send BL Loader Blocks
   ```

   ```
   cyttsp5_load_app: Send BL Loader Terminate
   ```

   If all of these prints appear, then there should be a successful load of the new firmware, otherwise, it is possible that a failure occurred.

19.2.8 Summary Menuconfig System Generated Defines

Table 19-2. System and User Defines

Automatic Defines Created by Menuconfig Selections	Description	TTDA/TTDL					
		3.1	3.2	3.3	3.4	3.5	3.6-3.8
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_DEVICETREE_SUPPORT	Use Device Tree instead of standard Board Configuration for Platform Data	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_PLATFORM_FW_UPGRADE	Enable Automatic FW upgrades using Platform Data included FW image array	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_PLATFORM_TTCONFIG_UPGRADE	Enable Automatic TT Configuration upgrade using Platform Data included TT Configuration array	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_BINARY_FW_UPGRADE	Enable Automatic FW upgrade using binary file (This is also required if TTHE_TUNER_SUPPORT is defined)	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_MANUAL_TTCONFIG_UPGRADE	Enable manual upgrade of TT Cconfiguration using binary file	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5	Includes TTDL driver in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_I2C	Includes the TTDL I2C Adapter module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_SPI	Includes the TTDL SPI Adapter module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_MT_A	Includes the Multi-touch Protocol A module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_MT_B	Includes the Multi-touch Protocol B module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_BUTTON	Includes the CapSense Button module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_PROXIMITY	Includes the Proximity module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_DEVICE_ACCESS	Includes the Device Access module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_LOADER	Includes the Loader module in the kernel build	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_DEBUG	Defines the system “DEBUG” to expand all dev_dbg() macros	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_VDEBUG	Defines the system “VERBOSE_DEBUG” to expand all dev_vdbg() macros	✓	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_DEBUG_MODULE	Includes the Debug module in the kernel build	✓	–	–	–	–	–
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_DEBUG_MDL	Includes the Debug module in the kernel build	–	✓	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_DEVICE_ACCESS_API	Includes the Device Access API for other driver interface to the TT device via the TTDL Device Access	–	–	✓	✓	✓	✓
CONFIG_TOUCHSCREEN_CYPRESS_CYTTSP5_TEST_DEVICE_ACCESS_API	Includes a test module for the Device Access API in the kernel build (used for TTDL validation and coding example).	–	–	✓	✓	✓	✓
CONFIG_FB	Build enabled for FB notifications for power management	–	–	–	✓	✓	✓

Automatic Defines Created by Menuconfig Selections	Description	TTDA/TTDL					
		3.1	3.2	3.3	3.4	3.5	3.6-3.8
UPGRADE_FW_AND_CONFIG_IN_PROBE	Perform FW and config upgrade during probe instead of scheduling a worker for it. This is disabled by default.	–	–	–	✓	✓	✓
TTHE_TUNER_SUPPORT	Build in support for the TrueTouch Host Emulator. This must be defined to support the Mobile Tuner for the build. This is disabled by default. Add define for this item in the cyttsp5_regs.h to enable.	✓	✓	✓	✓	✓	✓
CY_WATCHDOG_TIMEOUT	Enable/disable driver watchdog. Define as '0' to disable. Define as milliseconds to enable (for example, set to 1000 for a 1-second watchdog period (this is the default)).	✓	✓	✓	✓	✓	✓
EASYWAKE_TSG6	Use the advanced Easywake features of PIP 1.6 which allows custom wake gestures and also provides sliding finger wake gesturing.	–	–	–	–	–	✓
CY_TTCONFIG_VERSION_OFFSET	Set to the offset into the TT Configuration flash to where the configuration version is located. This value is used in order to lookup the configuration version in the Driver.bin and Driver.h Project Files (See CLARIFICATION note at end of OTA Upgrade With a .ihex File Some customers prefer to use the .ihex file format rather than the .bin file. To support this, the following steps are required: • Don't add the file to the kernel using "make menuconfig" • Convert the FW .bin file to a .ihex by using the Linux command: `objcopy -I binary -O ihex filename.bin filename.bin.ihex` • Modify the makefile in `/kernel/firmware` to build the filename.bin.ihex into the kernel: `fw-shipped-y += filename.bin.ihex` Board Configuration File Change)	–	–	✓	✓	✓	✓

20. IDAC Calibration

TTDL allows you to perform firmware self-test to execute IDAC calibration via ADB commands. This section describes the necessary steps required to run IDAC calibration. ADB utility is required to be installed and serial debugging be enabled.

20.1 Testing With TTDL

Note: For TTDA 3.1 and earlier: Typical `/path/to` in this chapter is `/sys/bus/ttsp5/devices/cyttsp5_device_access.main_ttsp_core`.

Note: For TTDA 3.2 and later: Typical `/path/to` in this chapter, for I^2C host bus products, is `/sys/bus/i2c/devices/4-0024`.

20.1.1 Calibrate IDAC for Mutual Capacitance Sensors

To calibrate IDAC, execute the following commands in order using a command window or script:

8. Suspend scanning is required for IDAC calibration test.

   ```
   adb shell
   echo "04 00 05 00 2F 00 03" > /path/to/command
   ```

9. Check if the response status is ready.

   ```
   cat /path/to/status
   ```

10. Get device passthrough command response bytes.

    ```
    cat /path/to/response
    ```

11. Send mutual-cap Sensors IDAC calibration command to command sysfs node.

    ```
    echo "04 00 06 00 2F 00 28 00" > /path/to/command
    ```

12. Check if the response status is ready.

    ```
    cat /path/to/status
    ```

13. Get device passthrough command response bytes.

    ```
    cat /path/to/response
    ```

14. Resume scanning after IDAC calibration test is done.

    ```
    echo "04 00 05 00 2F 00 04" > /path/to/command
    ```

15. Check if the response status is ready.

    ```
    cat /path/to/status
    ```

16. Get device passthrough command response bytes.

    ```
    cat /path/to/response
    ```

20.1.2 Calibrate IDAC for Mutual Capacitance Buttons

To calibrate IDAC, execute the following commands in sequence using a command window or script:

1. Suspend scanning is required for IDAC calibration test.

   ```
   adb shell
   ```

```
echo "04 00 05 00 2F 00 03" > /path/to/command
```

2. Check if the response status is ready.

```
cat /path/to/status
```

3. Get device passthrough command response bytes.

```
cat /path/to/response
```

4. Send mutual-cap Button IDAC calibration command to command sysfs node.

```
echo "04 00 06 00 2F 00 28 01" > /path/to/command
```

5. Check if the response status is ready.

```
cat /path/to/status
```

6. Get device passthrough command response bytes.

```
cat /path/to/response
```

7. Resume scanning after IDAC calibration test is done.

```
echo "04 00 05 00 2F 00 04" > /path/to/command
```

8. Check if the response status is ready.

```
cat /path/to/status
```

9. Get device passthrough command response bytes.

```
cat /path/to/response
```

20.1.3 Calibrate IDAC for Self Capacitance

To calibrate IDAC, execute the following commands in sequence using a command window or script:

1. Suspend scanning is required for IDAC calibration test.

```
adb shell
echo "04 00 05 00 2F 00 03" > /path/to/command
```

2. Check if the response status is ready.

```
cat /path/to/status
```

3. Get device passthrough command response bytes.

```
cat /path/to/response
```

4. Send self-cap Sensors IDAC calibration command to command sysfs node.

```
echo "04 00 06 00 2F 00 28 02" > /path/to/command
```

5. Check if the response status is ready.

```
cat /path/to/status
```

6. Get device passthrough command response bytes.

```
cat /path/to/response
```

7. Resume scanning after IDAC calibration test is done.

```
echo "04 00 05 00 2F 00 04" > /path/to/command
```

8. Check if the response status is ready.

```
cat /path/to/status
```

9. Get device passthrough command response bytes.

```
cat /path/to/response
```

21. Driver Without XRES Signal

Some hardware platforms for Linux/Android may not connect the TrueTouch XRES to a GPIO. Other cases may have creative solutions such as power cycling the TrueTouch part. This is acceptable, but if not done correctly, the reset operation in the driver will not work. Even though the driver supports using soft reset, Parade strongly recommends using hardware reset or power cycle because this is more reliable than the soft reset command because if the part is not addressable, then the soft reset will fail.

21.1 Solution to Designs Without XRES Signal

The driver will try the hardware reset and if that fails, it will perform a soft reset command. See Technical Notes for function names. The hardware reset driver functions look for the board configuration file definition of the atomic hardware reset function. If you use a sample board configuration file with an XRES reset defined, it generally returns zero (success) after pulsing the default GPIO assignment, even if the reset does not actually happen. This may need to be customized. If you are not using any hardware reset, it should not be defined so that the function returns a null. If implementing a hardware reset, it should return zero only if some action is performed. Otherwise, the failure return (-ENOSYS) will tell the driver to try the soft reset.

The soft reset command will take place as a backup only if either the reset function is not defined or returns -ENOSYS for a failure.

Debugging suggestion: If you ask for a copy of the kernel log file, it will show which reset failed. If the hardware reset is not causing an error message, the driver assumes it happened. If the XRES is not actually in use, this is an indication that there may be an incorrect function in the board configuration file.

21.2 Technical Notes

The board configuration file is a product configuration file that adapts the product to use various drivers and software based on the specific hardware. The ones sent out with our drivers are suggested scenarios. This is an area of development that needs the involvement of the customer's product integration engineer who can determine what platform data selections should be made for the driver.

The program flow for the reset is:

```
cyttsp5_reset_and_wait()          // reset and wait for bootloader mode
cyttsp5_hw_reset()                // try XRES then firmware reset command if fails
cyttsp5_hw_hard_reset()           //checks existence of reset function then calls it and if
that fails
cyttsp5_hw_soft_reset()
```

Board configuration file XRES pulse example

```
static int cyttsp5_xres(struct cyttsp5_core_platform_data *pdata,
             struct device *dev)
{
       int rst_gpio = pdata->rst_gpio;
       int rc = 0;
       gpio_set_value(rst_gpio, 1);
       msleep(20);
       gpio_set_value(rst_gpio, 0);
       msleep(40);
       gpio_set_value(rst_gpio, 1);
       msleep(20);
       dev_info(dev,
              "%s: RESET CYTTSP gpio=%d r=%d\n", __func__,
```

```
                pdata->rst_gpio, rc);
        return rc;
}
```

The power cycling source may look similar if the GPIO is connected to a transistor switch instead of XRES.

If you choose to code a power cycle into the cyttsp5_xres, be aware that the interrupt from the TrueTouch part is active LOW, so a power cycle will appear as an interrupt to the driver. Parade recommends that you nest the power cycle inside a disable/enable of the interrupts.

22. System Debugging

This section discusses tools and methods for successfully debugging your system.

22.1 Parade's TrueTouch Host Emulator Tool

The first tool you should use when debugging your system is Parade's TrueTouch Host Emulator (TTHE) v2.1 or later. Because the TTDL does not modify data or touch events, it is a good idea to verify the touchscreen configuration and operation. If the touch performance is poor, you should connect the touchscreen directly to TTHE to verify behavior.

22.2 Logic Analyzer

A logic analyzer is required to debug the communication between your Android system and the PIP device. Both the I^2C and SPI electrical interfaces use a limited number of signals. You must capture, (after the grounding connection) reset, interrupt, and the serial interface signals with decoding capability. This will allow you to trigger on the initial host reset pulse and capture the initial communication up to and including the first touch events.

Typically, you will see "heartbeat" pulses on the interrupt line indicating that the part is in bootloader mode. This may go on for a long time (seconds) while Linux and Android start up. The TTDL driver will then send a reset pulse, and one or two more interrupts will occur as the system re-enters bootloader mode. The TTDL will then communicate to switch from bootloader mode to operating mode. After this, interrupts are triggered by touch events and will stay asserted until the TTDL clears the event because it must always be in synchronous, hand-shaking mode.

If you need to report bugs or request support for this type of communication problem, a logic analyzer trace is useful.

Note When you measure the clock rate of a PIP device that is set to operate at 1 MHz, you will observe an initial legacy negotiating phase of 400-kHz operation during initialization. This is expected.

22.3 Enable Linux Debug Macros

Parade uses the standard Linux define device debug macros such as dev_dbg(). To enable these debug features, define DEBUG, and optionally VERBOSE_DEBUG, at the appropriate level. An example of this is in *cyttsp5_bus.h*. When the debug features are enabled, the TTDL will log error messages as they occur and significant normal milestones such as bootloader exit.

22.4 Helpful Linux Host Windows

The following three Linux host windows are helpful for system debug. You must have Android USB debugging enabled and be connected to a host Linux system with ADB installed. These separate command windows can run at the same time. You may need to locate your ADB command path.

22.4.1 logcat

```
$ sudo ./adb shell logcat
```

This will run the continuously updated Android log file.

22.4.2 getevent

```
$ sudo ./adb shell getevent
```

This will list all of the input event devices that are currently running. Look for the device associated with the TTDL such as */dev/input/event4*.

Use ^C to exit getevent and then restart getevent with details enabled for the device associated with the TTDL.

```
$ sudo ./adb shell getevent  -lt /dev/input/event4
```

This will run continuously updated event logging for the TTDL. These are the event signals sent by the Linux event handler to the Android platform event hub.

```
$ sudo ./adb shell getevent -r /dev/input/eventX
```

This will get the event rate, where “X” is the number of the associated device being monitored.

22.4.3 kmsg

```
$ sudo ./adb shell cat /proc/kmsg
```

This will run the continuously-updated kernel log. This log is where debug prints will appear from the debug enabled TTDL for each touch.

22.4.3.1 Debug Monitor

It is possible to get formatted prints into the kernel log dynamically. By loading the TTDL Debug Monitor module, for each touch event, a formatted print of the touch record information is printed to the log. This must be used cautiously, because it will degrade the touch update performance to handle the printing.

```
$ sudo ./adb shell push cyttsp5_debug.ko /data
$ sudo ./adb shell insmod /data/cyttsp5_debug.ko
```

When the debug monitoring is no longer desired, the Debug Monitor can be disabled:

```
$ sudo ./adb shell rmmod cyttsp5_debug.ko
```

To see what modules are currently inserted and running, use the listing command:

```
$ sudo ./adb shell lsmod
```

22.5 Android Display Settings

The Android system can be configured to help you debug touch issues using the normal user GUI. Here are some suggestions:

- In **Settings > Display**, set **Sleep** to **30 minutes**. This will keep the display on for as long as possible.
- In **Dev Tools**, use the **Pointer Location** tool to show touch ID, coordinates, cross hairs, and trail ending colors.
- In **Developer Options**, turn the following options on:
  - **Set USB Debugging**: This enables ADB remote debugging, which is used throughout this guide.
  - **Pointer Location**: This provides touch ID, coordinates, cross hairs, and trail ending colors everywhere in the GUI. It is important to know that if you have this turned on and go to the **Pointer Location** tool in **Dev Tools** you will see two traces for every touch.
  - **Show Touches**: This displays a circle or ellipse for each touch event proportional to the size of touch.

23. Mobile Tuner

This section discusses describes how to use ADB with TTHE 3.1 or later. ADB allows tuning and testing with a closed device environment using USB or Wi-Fi. TTHE is able to keep full functionality while using ADB.

23.1 Overview

Figure 23-1. Mobile Tuner Overview

The Mobile Tuner feature allows touchscreen solutions to be tested in an end system using TTHE. For example, testing touchscreens integrated on mobile phones.

23.2 Prerequisites

- Android SDK. Download the SDK at developer.android.com.
- Start SDK Manager after Android SDK installation is done. Use the SDK Manager to install the SDK tools and the API for your build of Android and the Google USB driver.
- TrueTouch Host Emulator v3.1 or later. See the TrueTouch device product release notes for TTHE version requirements. When the target Android device is connected to a Windows PC for the first time, chances are that USB driver cannot be installed properly. Thus ADB connection does not work as expected. To resolve the issue, see section Install USB Driver for Target Android Device for instructions on how to install USB driver for the target Android device on Windows PC.
- Ensure that the image flashed to the phone/tablet has the CONFIG_DEBUG_FS symbol defined. DEBUGFS must be selected in Linux menuconfig to run TTHE mobile tuner. Device Access module and Loader module must also be enabled in TTDL driver to use Mobile Tuner. In the *cyttsp5_bus.h* file for TTDA 3.0, TTDA 3.0.1, TTDA 3.1, or the *cyttsp5_core.h* file for TTDA 3.2 and TTDA 3.3, make sure that the constant TTHE_TUNER_SUPPORT is defined and enabled. Note that TTHE_TUNER_SUPPORT only needs to be enabled once. It is not necessary to enable it in the *cyttsp5_core.c*, *cyttsp5_device_access.c*, or *cyttsp5_loader.c* files. Also the “FW upgrade from binary file” option must be selected in menuconfig to enable the Mobile Tuner to perform firmware updates on the device (see Figure 23-2. Binary File Selection).

  Note: The Mobile Tuner uses the manual_upgrade system object to load firmware.

  - Load `cyttsp5_loader.ko` files to the Android device using the following commands:

    ```
    adb push cyttsp5_loader.ko /data
    ```

- Use the following commands to check whether DEBUGFS is already mounted:

  ```
  adb shell
  # mount
  ```

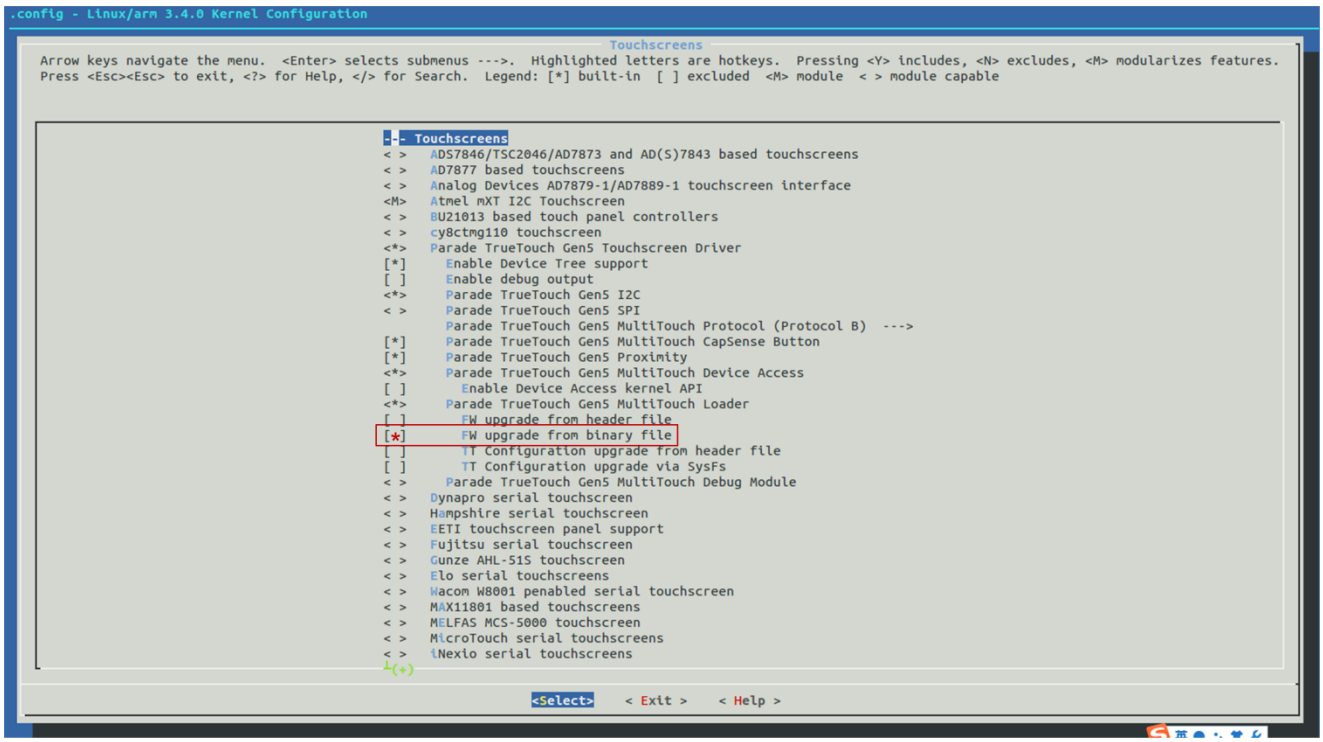
  If DEBUGFS is not already mounted, then perform the following commands to mount the debug file system on top of the `/sys/kernel/debug` directory.

  ```
  # mount -t debugfs none /sys/kernel/debug
  ```

- Insert `cyttsp5_loader.ko` is also needed.

  ```
  adb shell
  # cd /data
  # insmod cyttsp5_loader.ko
  ```

Figure 23-2. Binary File Selection



23.3 Install USB Driver for Target Android Device

Note 1 For the same make and model of Android devices, this process of installing USB drivers only need to be done once on each Windows PC running the Mobile Tuner.

Note 2 Make sure the Android SDK is properly installed before proceeding with the instructions in this section.

Note 3 The images in this section are based a Windows 7 machine. Other Windows PCs should have similar options.

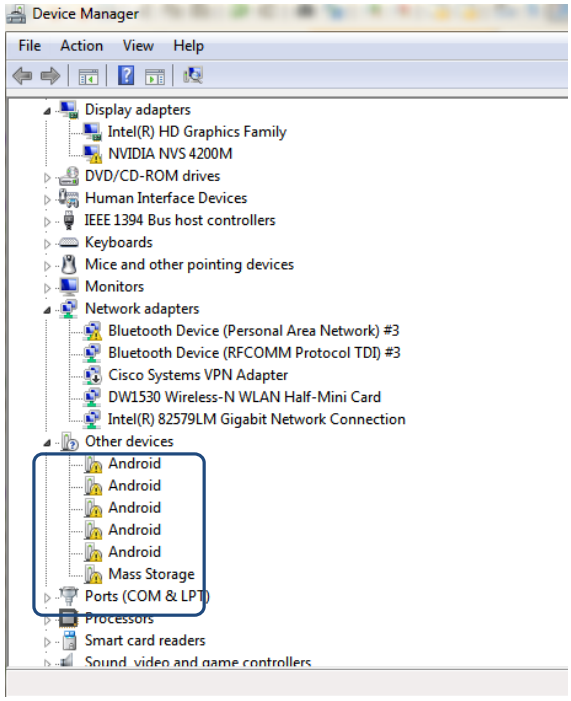
When a target Android device is connected to a Windows PC for the first time, it may display a warning message, “Unable to install USB driver for the device”. Automatic driver update online may also fail. As a result, the ADB cannot work appropriately, which blocks further device testing and debugging.

To install the USB driver for the target Android device properly, follow these instructions:

1. Open the Device Manager window; you will see a warning sign (yellow triangle) on the lower right corner of the target Android device icon. It indicates that the driver for the device is not properly installed.

Note: If there is no warning sign on the Android device icon, then the USB driver is already installed successfully. So there is no need to follow the steps in this section. Proceed to the Settings for Mobile Tuning via USB or Settings for Mobile Tuning Via WiFi section.

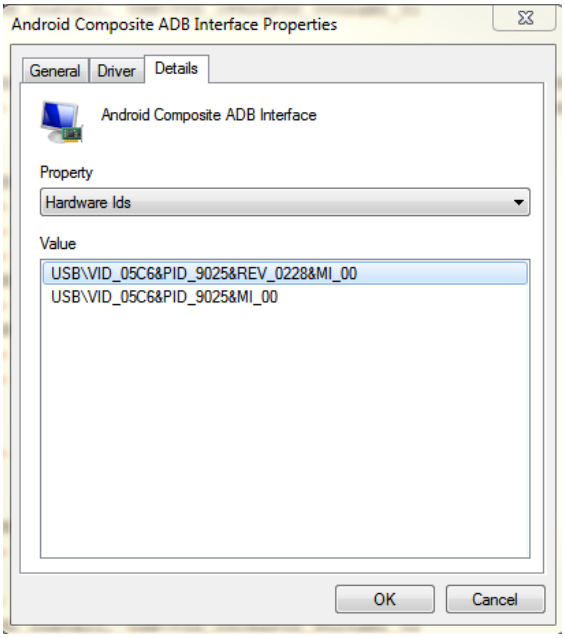
Figure 23-3. Device Manager Window



2. Right-click on the warning sign and select **Properties**.
3. Click the **Details** tab, select **Hardware Ids**. Record the Hardware IDs of the device.

   Note: Figure 23-4 is only an example. Other Android devices have different Hardware IDs.

Figure 23-4. Example Hardware IDs Window



4. Go to the Android installation folder and open **android_winusb.inf** file. Typical path is `C:\Program Files\Android\android-sdk\extras\google\usb_driver`. You can see a list of devices in `[Google.NTx86]` and `[Google.Ntamd64]` similar to Figure 23-5.

Figure 23-5. Example [Google.NTx86] and [Google.Ntamd64] in android_winusb.inf

```
[Google.NTx86]
;Google NexusOne
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_0D02
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_0D02&MI_01
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_4E11
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E12&MI_01
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E22&MI_01

[Google.NTamd64]
;Google NexusOne
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_0D02
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_0D02&MI_01
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_4E11
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E12&MI_01
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E22&MI_01
```

5. In both `[Google.NTx86]` and `[Google.Ntamd64]`, add the similar "`%SingleAdbInterface%`" and "`%CompositeAdbInterface%`" lines for the target Android device according to the Hardware Ids obtained in Step 3. Figure 23-6 shows an example.

Figure 23-6. Example Addition of Android Device Information in android_winusb.inf

```
[Google.NTx86]
;Google NexusOne
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_0D02
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_0D02&MI_01
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_4E11
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E12&MI_01
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E22&MI_01

;Name of the target Android device
%SingleAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&MI_00
%CompositeAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&REV_0228&MI_00

[Google.NTamd64]
;Google NexusOne
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_0D02
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_0D02&MI_01
%SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_4E11
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E12&MI_01
%CompositeAdbInterface%     = USB_Install, USB\VID_18D1&PID_4E22&MI_01

;Name of the target Android device
%SingleAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&MI_00
%CompositeAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&REV_0228&MI_00
```

6. If there are multiple devices, as shown in Figure 23-3, repeat Step 2 to 5 for each of them. Save the **android_winusb.inf** file.
7. Go back to Device Manager. Right-click on the Android device icon (shown in Figure 23-3), select **Update Driver Software…** > **Browse my computer for driver software**. In the "Search for driver software in this location" box, click **Browse** and point to the directory where the **android_winusb.inf** file is located. Typical path is `C:\Program Files\Android\android-sdk\extras\google\usb_driver`.
8. Driver update should start and be successful.

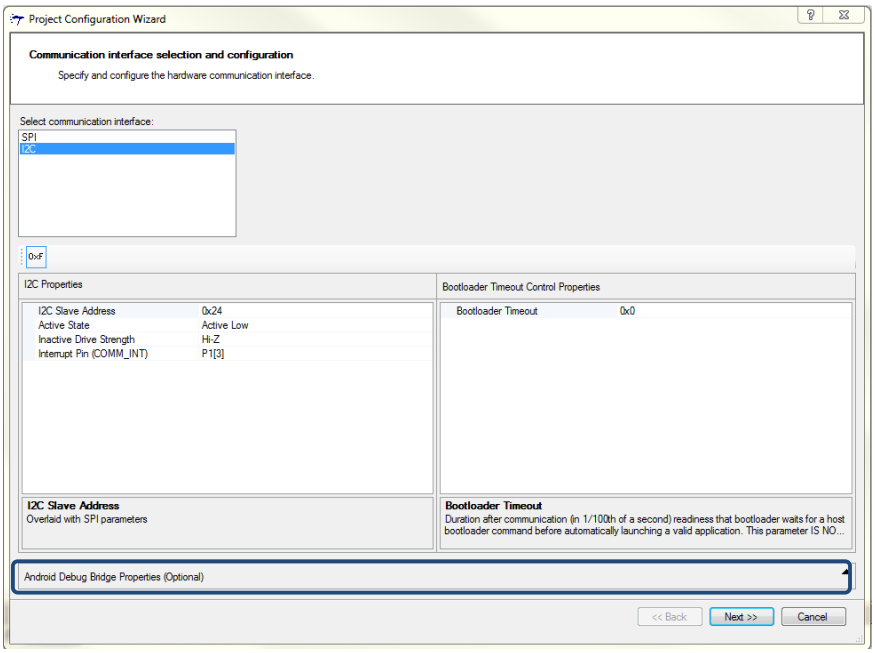
23.4 Settings for Mobile Tuning via USB

23.4.1 TTHE 3.1 and Later

Follow these steps to set up the mobile tuner:

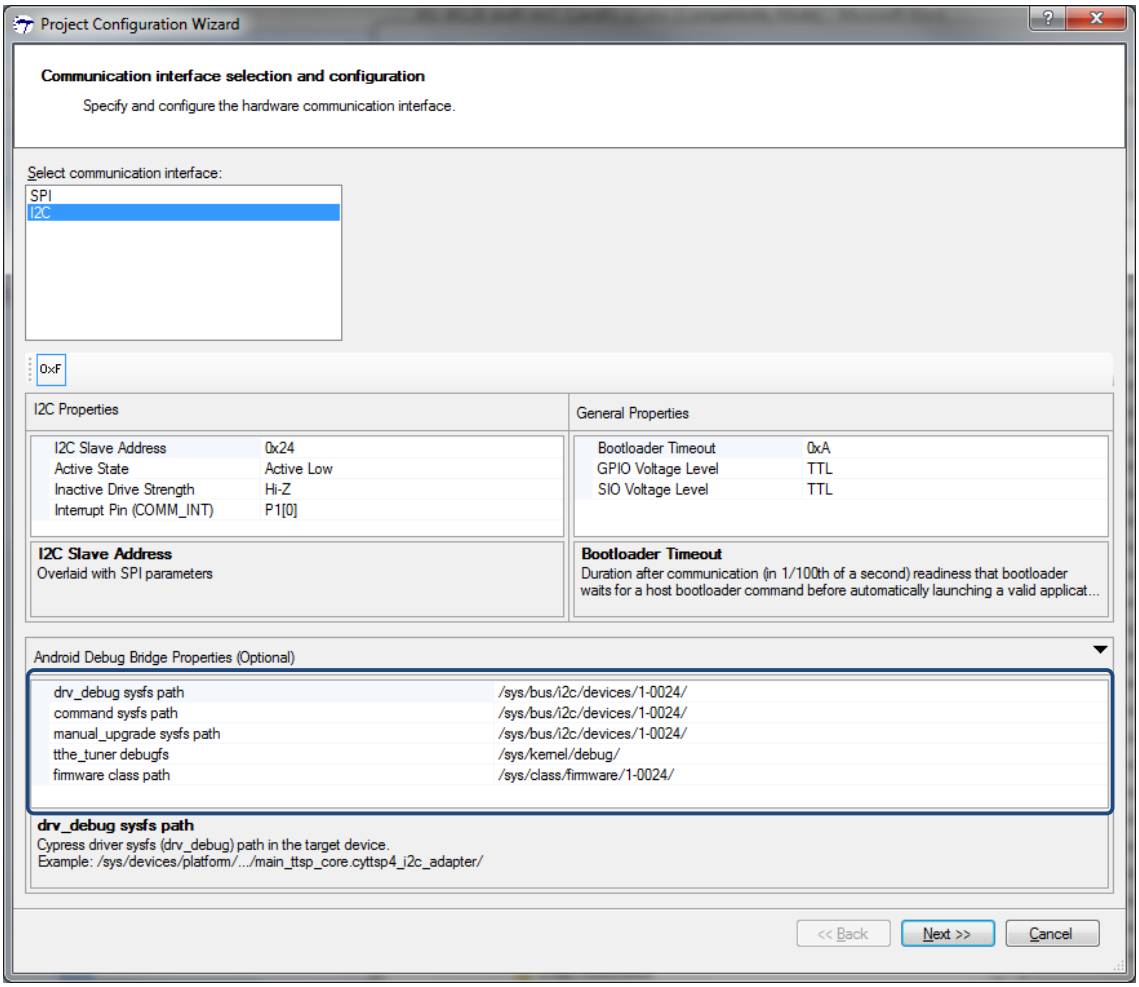
1. In TTHE menu, select **View** > **Project Configuration Wizard**.

Figure 23-7. Project Configuration Wizard in TTHE 3.1 and Later



2. Click **Android Debug Bridge Properties (Optional)** to expand the ADB settings menu, as shown in Figure 23-8.

Figure 23-8. Example Android Debug Bridge Settings in TTHE 3.2 and Later



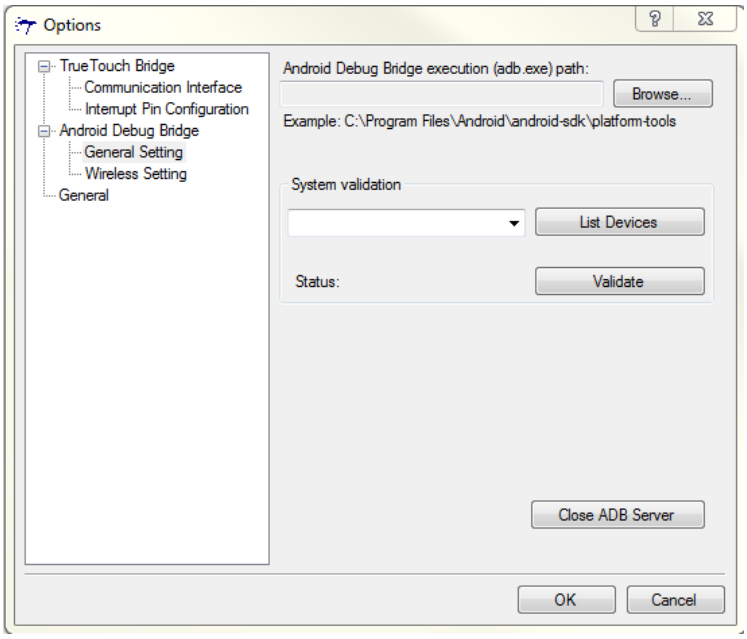
3. Type in the ADB paths for each of the sysfs and debugfs required according to Table 23-1. These are sample paths; the actual paths may vary depending on where the driver objects are located in the product file directory structure. The “tthe_tuner” path in Table 23-1 is a fixed path location. The “firmware class” path in the table is fixed for the “/sys/class/firmware/” portion of the path. Refer to Section 23.4.2 for information about finding the path.

Table 23-1. Sample Android Debug Bridge Path Settings for TSG5 Devices

drv_debug sysfs path	/sys/bus/i2c/devices/1-0024/
command sysfs path	/sys/bus/i2c/devices/1-0024/
manual_upgrade sysfs path	/sys/bus/i2c/devices/1-0024/
tthe_tuner debugfs	/sys/kernel/debug/
firmware class path	/sys/class/firmware/1-0024

4. Open TTHE menu **Tools** > **Options** > **Android Debug Bridge** > **General Setting** tab. Enter the *adb.exe* path in the “Android Debug Bridge execution (adb.exe) path”. Typical path is `C:\Program Files\Android\android-sdk\platform-tools`.

Figure 23-9. Android Debug Bridge Execution General Setting in TTHE 3.1



5. Click **List Devices** and then **Validate**. If ADB is set up correctly, “Status:” shows “Validate success”. Then you are ready to use mobile tuner features.

23.4.2 Finding the Driver Directory Paths

For TTDA 3.1 and below, the path for the “drv_debug”, “command”, and “manual_upgrade” sysfs objects are found at the path `/sys/bus/ttsp5/devices/cyttsp5_loader.main_ttsp_core/`.

For TTDA 3.2 and above, the example paths shown in Section 23.4.1 are for devices that are integrated onto a host I2C bus and at the default Parade device address 0x24 (“1-0024” where the 1 is for the bus number and the 0024 is for the device address on that bus). For SPI bus, the directory name typically shows the SPI bus and the device slave select number (for example, “spi1.1”).

It is not always known where the device is located in the host buses so knowing some of the expected sysfs object names associated with the driver helps to find the path. One way to determine this is to perform a recursive directory listing to a text file and search for the driver sysfs objects. The following objects should always exist if the driver comes up normally:

- drv_debug
- drv_irq
- drv_ver

When the path for these objects is found, that path is the one for use in the “drv_debug”, “command”, and “manual_upgrade”; you should also see those objects in the list for that directory.

If you do not see the “manual_upgrade” in the list, then either the Loader module (cyttsp5_loader.ko) was not built or was built as a module but not inserted at run time. If this object is missing, then the TTHE will fail to connect through the driver.

If you see the objects: “formated_output” and “int_count”, then the driver debug module is running (cyttsp5_debug.ko). This module should only be loaded and running during specific touch debug time. It reduces touch performance significantly and is recommended not be compiled as built-in but rather as a module and only inserted when needed.

Note that it is not possible to find the firmware class path until the driver activates the firmware class library. This only occurs when a ‘1’ is written to the “manual_upgrade” sysfs object such as:

```
> adb  push  cyttsp5_loader.ko  /data
> adb  shell
# insmod  /data/cyttsp5_loader.ko
# echo 1 > /sys/bus/i2c/devices/1-0024/manual_upgrade
```

Then you should be able to see the following path (using the sample path) and the “data” and “loading” objects:

```
# ls /sys/class/firmware/1-0024
data
```

```
loading
power
subsystem
uevent
```

Note that the “data” and “loading” objects are used for downloading firmware to the device as described in Section 8.1 .

23.5 Settings for Mobile Tuning Via WiFi

1. First, an ADB Wireless app needs to be installed and run in the Android device for the Mobile Tuner to be working with WiFi. The ADB Wireless app can be obtained and installed from Google Play. Allow root access to the wireless application.

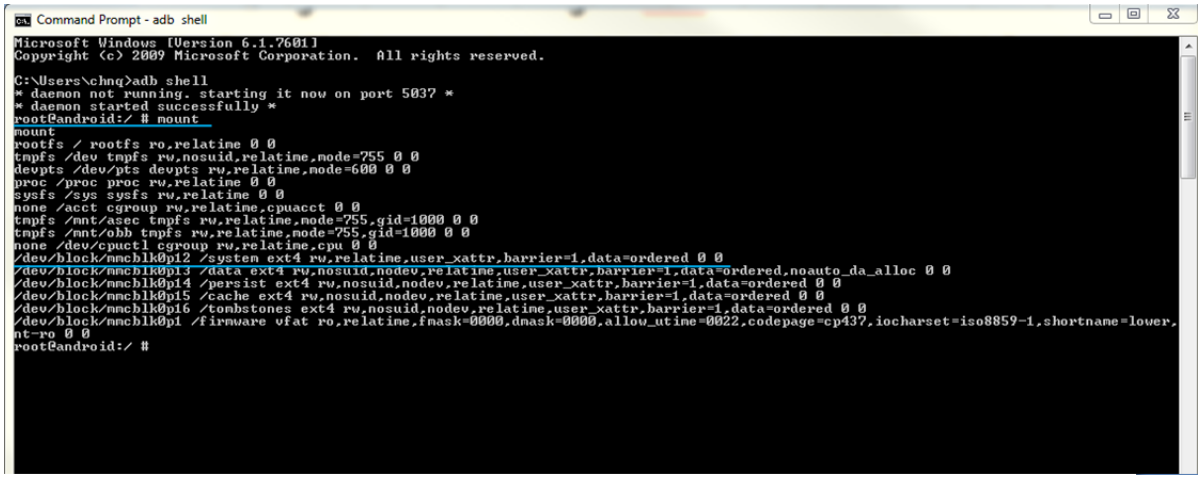
   Alternatively, ADB Wireless app apk and *Superuser.apk* can be acquired and installed in the Android device manually to achieve the above requirement. The following steps show the example procedure to install *adbWireless.apk* and allow root access. Depending on the end Android device, some paths in the example need to be changed. Consult a Parade FAE to get the ADB Wireless app apk and *Superuser.apk* if you do not have them already.

   (1) Find system settings using command:

   ```
   adb shell
   # mount
   ```

   Record the line that contains the flash device information under the */system* directory. Example from a Parade development kit is shown in Figure 23-10. Customer platform would result in a different path.

   Figure 23-10. Example Output of ADB `mount` Command



   (2) Using system information, construct the following command to remount the system:

   ```
   # mount -o remount,rw -t ext4 /dev/block/mmcblk0p12 /system
   ```

   Note: The above line is example only. According to Step (1) example, the */dev/block/mmcblk0p12* is the flash device of fs type “ext4” and the */system* directory is mounted as “rw”. Depending on the information recorded in Step (1), modify the mount command accordingly. Make sure that */system* is remounted as “rw” (read and write). Then proceed with the commands below.

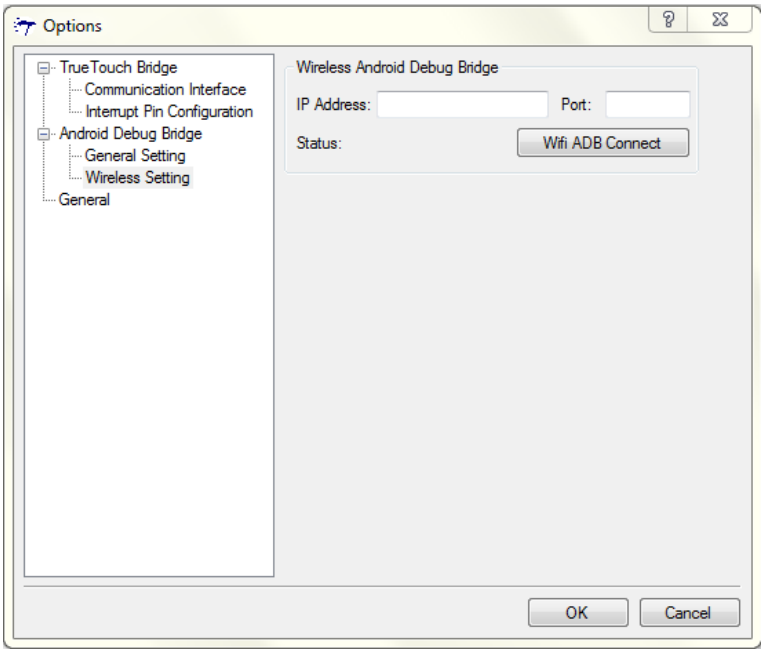
   ```
   adb push /bin/su system
   adb push /app/Superuser.apk system
   adb push adbWireless.apk system
   adb shell
   # cd system/xbin
   ```

```
# mv su su_old
# mv ../su .
# chmod 04755 su
# cd ../
# mv Superuser.apk app
# mv adbWireless.apk app
```

(3) Restart the system.

2. In TTHE menu, select **Tools** > **Options**. In **Android Debug Bridge** > **Wireless Setting** tab, enter the IP address and port for the Wireless Android Debug Bridge.
3. Click **Wifi ADB Connect**.
4. Check “Status”; **connected** indicates successful connection to the target device. Now, you can tune your mobile.

Figure 23-11. Tools Options Wireless Settings Example



24. Troubleshooting and FAQ

***Q:** Touch device does not go into Sleep mode.*

A: Make sure the CONFIG_PM_RUNTIME function is enabled in TTDL. It must be enabled for the early suspend function in TTDL to work. The CONFIG_PM_RUNTIME function enables the runtime power management features of Linux so that TTDL can put the controlled touch device into the low-power state on early suspend/late resume events.

***Q:** What are the benefits of the Bus Model in TTDL over monolithic design?*

A: Compared to traditional monolithic driver design, modular design is more flexible, customizable, scalable, and extensible. Modules can be added or removed at runtime. Each module subscribes to the required events from core modules. Debug modules can be inserted without a kernel rebuild, so debugging the driver during the product development and evaluation phase becomes easier. When you want to add new features to the driver, monolithic designs require driver redesign, while a modular design allows new features to be isolated to existing or new modules. It allows easier and faster development and implementation.

You can customize most of the features by modifying the Linux board configuration file. You do not need to modify the module files. Thus, the bus model lowers the barrier of entry and expedites the process for porting drivers.

***Q:** What are the benefits of the Simplified Core Driver model TTDL over TTDL Bus Module model?*

A: The Simplified Core only needs a single Linux driver device. It also combines the most common worker modules into the Core module, thus reducing the complexity. The combined and integrated worker modules can be conditionally included in the kernel build and can include multi-touch (MTA or MTB), CapSense Button, and Proximity sensing. The full product support modules (device access, loader, and debug monitor) remain separate loadable kernel modules that can be built-in or built as modules. The loadable and integrated worker modules continue to subscribe to the Core for device interrupt event notification.

***Q:** Where can I find register map information for TrueTouch parts?*

A: The register map information is documented in Technical Reference Manual (TRM). The document is distributed by a Parade Field Applications Engineer (FAE) to customers under NDA. See TrueTouch® Documentation for details.

***Q:** Where can I find information about handshaking methods between the host and the TrueTouch products? How do I configure the handshaking method?*

A: The TTDL 3.x firmware is not configurable for handshaking method. It always uses the Synchronous Level Handshake: the firmware signals the event interrupt and holds it active until the host reads the last byte in the device read buffer. Normally, the driver will configure the interrupt event as low-edge trigger; however, if required by the host, the driver will configure the interrupt event for low-level trigger. This is done by setting the platform data field “level_irq_udelay” to a non-zero value.

***Q:** I want to check and debug touch performance in an enclosed Android device with TTDL built in. Can I change the tuning configurable parameters without disassembling the unit?*

A: Yes. There are two ways of achieving this.

- Option 1: Connect the Android device to the TrueTouch Host Emulator (TTHE) via USB or Wi-Fi. You can see the touch performance, line drawing, and heat map data of the touch device in TTHE. It also allows you to make configuration changes in TTHE and apply directly to the Android device. See Mobile Tuner for details.
- Option 2: TTDL provides a Group 6 sysfs interface that you can use to read and write touch configurations. You can access the interface via Android Debug Bridge (ADB) connection. See **Error! Reference source not found.** for details.

***Q:** What is the relationship between TTDA 2.x and TTDL 3.X?*

A: TTDL 3.X supports TSG5/6 products that use the Packet Interface Protocol (PIP) device interface, whereas TTDA 2.x supports TMA4xx/TMA1036/TMA44X products that employ TrueTouch Standard Product (TTSP) register map interface. The TTDL 3.x design is based on the TTDA 2.2 architecture, but is re-designed specifically for the PIP interface.

***Q:** In TTDA 3.0 and TDA 3.1, there is a #define CY_LDR_SWITCH_TO_APP_MODE_TIMEOUT in cyttsp5_loader.c file. What is its use and how do you set this value properly?*

A: The `CY_LDR_SWITCH_TO_APP_MODE_TIMEOUT` value is the timeout in the bootloader mode before the driver issues a launch application command to switch to the application mode. When the timer expires, the driver will get the HID descriptor for the bootloader and will execute the launch application command if the touch device is still in the bootloader mode. However, if the firmware has already launched the application before the `CY_LDR_SWITCH_TO_APP_MODE_TIMEOUT` timer expires, then the driver will not send the launch application command. The default value for this timeout is 300 ms.

Depending on your firmware implementation, this value may need to be modified. For example, if the firmware automatically launches the application in about 300 ms as well, chances are that the driver may get the HID descriptor indicating that the device is in the bootloader mode, which triggers a launch application command. At the same time, firmware may have switched to the application mode. Therefore, the launch application command will be executed in the application mode, which is not desirable. It is a good practice to set the two timeouts in the driver and firmware apart.

(**Note:** This definition is no longer used in TTDA 3.2 and later.)

***Q**: I have sent a command to the device using one of the PIP defined commands. I observe that the interrupt line is asserted low after the command is sent and I read the response, but the interrupt line does not de-assert after the read. How do you get the device to properly de-assert the interrupt line at the end of the read?*

A: The PIP specification (refer to the application note (001-85948) or the device TRM) specifies that the interrupt for a command response will automatically de-assert after the host has read all the response bytes in the device read buffer. The first two bytes in the read buffer tell how many total bytes are in the read buffer including the first two length bytes. The host must read this number of bytes. Typically, the host will read 2-bytes = length. Then read (length) bytes. See the PIP specification for proper command response formats.

***Q**: I notice that the interrupt line is stuck low when the driver starts up. How do you get the device to de-assert the interrupt line?*

A: The PIP specification (refer to the application note (001-85948) or the device TRM)) specifies that the device will assert the interrupt line when it completes a device reset including after power up and after an XRES assertion. This assertion is called the reset sentinel. The host must read the first two bytes from the device read buffer. These two bytes will be 00 00 if this is a reset sentinel response. After reading the two bytes, the device will de-assert the interrupt and normal operations can be performed. It is also possible to receive the byte pair 02 00 which means the device is ready for new commands (note: this is a length of 2-bytes which equals the length bytes alone).

***Q**: What is the driver Watchdog Timer and how it is used?*

A: The driver runs a Watchdog timer event. On the timer event the Watchdog tries to execute the PIP Ping Command to the deivce to see if it properly responds. If this command fails, then the driver will attempt to restart which includes resetting the device. The Watchdog timer period is set by the #define CY_WATCHDOG_TIMEOUT to some number of milli-seconds (default is 1000 ms (1 sec)). This timeout period may be too often when in driver integration development and can be set to some longer timeout period such as 5000 (5 sec) or 10000 (10 sec). The default value was selected as the least intrusive for a well integrated driver and short enough to restart the device in a reasonably short human time frame. It is also possible to completely disable the Watchdog by setting the constant to 0.

Revision History

Document Revision History

Document Title: TrueTouch® Driver for Android (TTDL) PIP User Guide

Document Number: 001-90126

Revision	Issue Date	Origin of Change	Description of Change
**	11/21/2013	SWU	New User Guide.
*A	02/11/2014	SWU	Added **Error! Reference source not found.** Fixed several grammatical edits Added missing hover feature to **Error! Reference source not found.**
*B	06/18/2014	KEV	Added TTDA 3.3 Features Updated TTDA 3.2 Features section to include the new Path Name Convention (Section **Error! Reference source not found.**) and Updated Structures (Section **Error! Reference source not found.**). Added Manufacturing (In-system) Tests - Section 9.1 Added fixes for User Guide defects: 1. CDT112303 - Added text to Section Virtual Keys to describe CY_MT_FLAG_VKEYS. Updated note in 16.2 Virtual Key Map to clarify the naming of the "virtualkeys" object and the key map and key layout file names. Added information to 16.3 item 1) to explain that CY_MAXX and CY_MAXY need to be set manually when Virtual Keys are used, otherwise the driver uses the X, Y resolution information from the System Information. 2. CDT124223 - Added explanation in section **Error! Reference source not found.** that PIP always effectively uses Synchronous Level handshake method. Added comment to FAQ Answer to question about handshake that the level_irq_udelay is used as Boolean to determine whether to set the host interrupt service to low edge trigger (default) or low level trigger (any non-zero value. 3. CDT172586 - Added Table 1-1. Driver Supported Features. Table include Driver PIP versions and is matrixed by supported features. Includes applicable kernel versions, FW versions and devices. 4. CDT172589 - Cleaned up the bullet 4 in Section 23.2 to include all needed driver settings to enable Assisted Tuning in the driver. 5. CDT174215 - Section 23.2 shows prerequisites for Mobile Tuner. Bullet 4 explains what modules and definitions are required to enable the Mobile Tuner including the CONFIG_DEBUG_FS and TTHE_TUNER_SUPPORT definitions. 6. CDT174272 - Section 23.2 shows prerequisites for Mobile Tuner. Added note to Bullet 4 that explains that Mobile Tuner uses the Manual_upgrade system object to load firmware. 7. CDT175694 - Added comment to section 16.3 that for Device Tree, changes may be made to the Device Tree which only requires that the Device Tree is rebuilt. Otherwise changes to driver code require driver and kernel rebuild. 8. CDT179464 - Updated Section **Error! Reference source not found.**to

Revision	Issue Date	Origin of Change	Description of Change
			include enable and disable Watchdog instructions. 9. CDT181245 - CDT Items 1-8 above. 10. CDT181475 - Added instructions at the end of section 19.2.1 for updating the constant: CYTTSP5_LOADER_FW_UPGRADE_RETRY_COUNT in the loader source to control the number of times to attempt to automatically load the firmware at startup. 11. CDT184253 - Added Caution and Recommendation paragraphs at end of section 19.2.3
*C	10/08/2014	KEV	Added NDA requirement to cover page. Added NDA requirement to footers. Section A - Added New Features in TTDA 3.4 to the list. Table 1-1 Added Row for TMA445A at bottom of table to exclude the features that are not supported in TSG6_M.BASE.V1 (EZ-wake, Hover, Stylus). Section 1.3.1 Added TMA445A Datasheet and TRM to the documentation table. Section 1.3.1 Added CY3295-MTK User guide to documentation table. Table 3-2 Added Dragon board files to the table of TTDA 3.2 and above file list with notes that the files are for TTDA 3.4. Table 3-2 Added missing files: cyttsp5_test_device_access_api.c and cyttsp5_device_access-api.h to the list. Section 9.0 - Added new section for TTDA 3.4 new features including: support for Panel ID reporting by the bootloader for use by the Driver to select proper configuration to load and LCD blanking and unblanking notification for power management functions (sleep and wake). Added CM/CP Test Program description. Added Figure 15-1 for automatic firmware and TT Configuration update rules. Section 15.2.6 Added TTDA 3.4 column to the definitions table. Added CONFIG_FB and UPGRADE_FW_AND_CONFIG_IN_PROBE for TTDA 3.4. Figure 19-7 Updated Wizard picture to show the typical paths for TTDA 3.2 and above devices for ADB connection. Table 19-1 Updated to show typical paths for TTDA 3.2 and above. Added firmware class path.
*D	10/13/2014	KEV	Section 9.1.3: Removed CM/CP test program description. This test program is not part of the TTDA distribution. Section 19.4.1 Item 3: Added comment that that the paths shown are sample paths and the actual paths may vary depending on where the driver objects are located in the product file directory structure. Section 19.4.2: Added the section to describe steps for locating the directory path the driver sysfs objects.
*E	02/25/2015	KEV	Changed title to “TrueTouch® Driver for Android (TTDA) PIP User Guide.” Added Troubleshooting and FAQs in section 1.4 for stuck interrupt assertions after command and startup conditions. Updated Table 3-2 to include Kconfig.patch and Makefile.patch in the kit file list (TTDA 3.5 and above). Updated Section 4.1.3 to describe calls to HID Set Power (SLEEP, ON) command to put the device on low power and to wake the device. Added Section 6.5 Restore Parameters on Restart. Updated Section 8.1.1 to describe debugfs path for Manufacturing Tests. New path is for TTDA 3.5 and above. Added new Section 10 New Features in TTDA 3.5 to include path for debugfs.

Revision	Issue Date	Origin of Change	Description of Change
			Added Section 16.2.4 Manual Update TT Configuration using a binary file. Updated table in Section 16.2.7 to add TTDA 3.5.
*F	06/16/2015	KEV	Updated Table 1-1. Driver-Supported Features to add 8-Button Support and Advanced EZ-Wake Feature columns. Added TTDL 3.6 row. Added Watchdog Timer Question and Answer in Section **Error! Reference source not found.Error! Reference source not found.**. Removed obsolete sysfs "Output Register ID" in Section **Error! Reference source not found. Error! Reference source not found.**. Updated Section **Error! Reference source not found.Error! Reference source not found.** to describe that only the easy_wakeup_gesture flag needs to be set with a gesture id to enable the easy wakeup for TTDA 3.6. Even though the advanced easy wakeup does not need a gesture selection, the driver structure still needs one of the gesture flags assigned to easy_wakeup_gesture to enable easy wakeup feature in the driver. Added Section **Error! Reference source not found.Error! Reference source not found.** Added Watchdog suggested use information at end of Section **Error! Reference source not found.Error! Reference source not found.**. Added Table 19-1. Firmware Version Information for firmware and configuration parameter files. Updated Section 19.2.8 Summary Menuconfig System Generated Defines to include the new EASYWAKE_TSG6 define and to add the TTDL 3.6 column.
*G	09/16/2015	KEV	Convert to Parade Technologies, Ltd. document. Add Section 9.3 Range Checking CM/CP Manufacturing Test to describe the new range checking CM/CP manufacturing test. Add CLARIFICATION note to end of Section 19.2.3 OTA Upgrade With a .ihex File Some customers prefer to use the .ihex file format rather than the .bin file. To support this, the following steps are required: • Don't add the file to the kernel using "make menuconfig" • Convert the FW .bin file to a .ihex by using the Linux command: `objcopy -I binary -O ihex filename.bin filename.bin.ihex` • Modify the makefile in `/kernel/firmware` to build the filename.bin.ihex into the kernel: `fw-shipped-y += filename.bin.ihex` Board Configuration File Change to describe setting the #define CY_TTCONFIG_VERSION_OFFSET. Add #define CY_TTCONFIG_VERSION_OFFSET to Table 19-2. System and User Defines.
*H	07/27/2016	PWAR	Updated Table 1-1. Driver-Supported Features to add TTDL3.8 Updated section **Error! Reference source not found.**"Proximity" to describe the sysfs node for enabling proximity for TTDL 3.2 and above. Added Section **Error! Reference source not found.**"New Features in TTDL 3.8" Added Section 19.2.3 "OTA Upgrade With a .ihex File" Updated Table 19-2. System and User Defines to include TTDL3.8 Added getevent example in section 22.4.2

Revision	Issue Date	Origin of Change	Description of Change
			Added Section **Error! Reference source not found.** “TTDL sysfs and debugfs nodes” which describes all available nodes.
*I	01/18/2017	Phil W	Re-organized the entire document, moved section B to section C and created a new section B “Features and Modules”, moving all sections showing new features in a specific version into a more organized section B. Added new sysfs node information coming in TTDL3.9 into section 15