Synaptics Guest Code User Guide

Synaptics Confidential

Contents

1. Getting Started 3
1.1. Introduction of Synaptics TouchComm Protocol 3
1.2. Synaptics TouchComm Driver 3
1.3. Requirements 3
2. Synaptics TouchComm Driver 4
2.1. Overview 4
2.2. TouchComm Driver Contents 6
3. Driver Integration 8
3.1. Driver Porting 8
3.2. Build-Time Configurations 9
3.3. Implementations of Runtime Abstraction 10
3.4. Implementations of Platform-Specific Interface 10
3.5. Building of TouchComm Driver 12
4. Driver Implementation 13
4.1. Startup Sequence 13
4.2. Initialization of Touch Reporting 14
4.3. Handling of ATTN Assertion 15
4.4. Touch Reports 16
4.5. Firmware Update or Recovery 16
5. Software Interface 17
5.1. Device Node 17
5.2. IOCTL Operations 17
6. Revision History 18

1. Getting Started

1.1. Introduction of Synaptics TouchComm Protocol

Synaptics TouchComm is a packet-based communication protocol used by Synaptics touch and touch/display controller products to interface with a host system — typically the main system processor. Devices enabled with TouchComm use either I^2C or SPI as the physical transport layer and operate as slave devices on the respective bus.

TouchComm communication consists of messages, categorized as either reads or writes from the host's perspective. Writes from the host to the device are interpreted as commands, while reads from the device to the host are either responses to those commands or device-generated reports.

Reports are produced whenever the device has data to communicate. The type and frequency of these reports depend on the device's features and operational mode. Importantly, TouchComm allows only one command to be processed at a time; the host must wait for a response to each command before issuing the next command.

1.2. Synaptics TouchComm Driver

The Synaptics TouchComm driver is implemented as a platform device driver within the Linux input subsystem framework. Its primary function is to manage communication with the hardware device through the associated communication bus or subsystem.

Like a typical input device driver in a Linux system, the TouchComm driver processes touch reports generated by the device whenever touch input (e.g., finger contact) occurs on the sensor.

To control device behavior, the driver sends commands to the device and waits for a corresponding response. Once a command is sent, the driver retrieves the response message, which indicates whether the operation was successful or not.

Additionally, the driver provides a software interface for interaction with Synaptics user-space applications, enabling configuration, diagnostics, and data access.

1.3. Requirements

To integrate and evaluate the Synaptics TouchComm driver, or to deploy it in a product environment, the following components are required:

- Synaptics TouchComm reference driver release package
- Synaptics touch product or Synaptics Touch Evaluation Board
- A development platform running a Linux-based operating system

2. Synaptics TouchComm Driver

2.1. Overview

The block diagram in Figure 1 provides an overview of the TouchComm driver architecture.

At the core of the design is the **TouchComm core library**, a platform-independent middle layer that encapsulates all supported TouchComm operations. This library abstracts common functionality and serves as the bridge between the hardware and operating system layers.

At the top layer, the device driver implementation integrates with the Linux input subsystem, enabling the system to process and report input events. At the bottom layer, the communication interface consists of hardware-dependent components responsible for communicating with the Touch IC via the appropriate transport bus (e.g., I^2C or SPI).

Upon successful driver installation, an input event node is created and registered by the TouchComm driver, allowing the system to receive and handle touch events through the Linux input event interface.

Additionally, a dedicated character device node (/dev/tcm*) is provided to facilitate communication between the driver and Synaptics user-space applications using IOCTL commands.

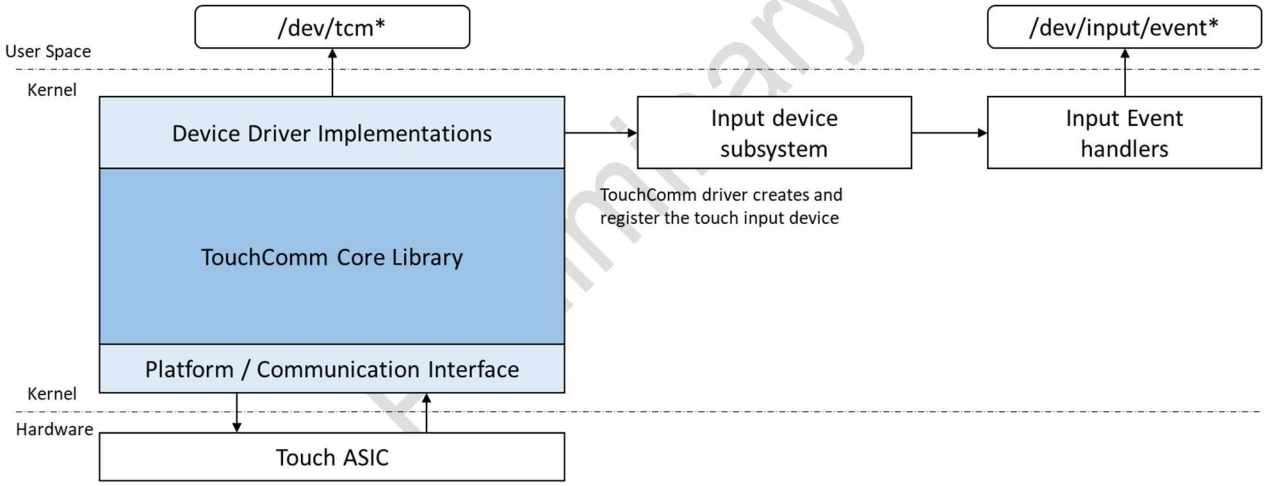

Figure 1. TouchComm Driver Block Diagram

The overall architecture of the TouchComm core library is illustrated in Figure 2.

The core library is implemented as a platform-independent middleware layer, composed of a **command handling module** and a set of **operation API**s. To ensure portability across platforms, all platform-dependent functions—including those from the standard C runtime—are abstracted through a dedicated **Platform Abstraction Layer** (**PAL**).

The **command handling module** serves as the protocol engine, implementing the TouchComm command-response mechanism. It is responsible for encoding and dispatching commands to the device, as well as receiving and parsing corresponding responses. This module is invoked internally by the various APIs exposed by the library.

The **operation APIs** provide a functional interface for higher-level components (such as device drivers) to perform tasks including device initialization, report retrieval, diagnostics, and firmware management. API calls are translated into one or more TouchComm commands, processed by the command handling module, and transmitted to the device via the communication interface.

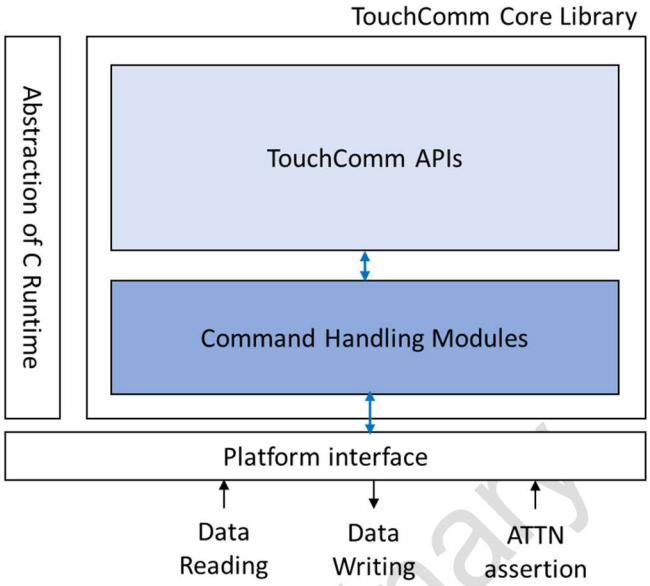

Figure 2. Overview of TouchComm Core Library

The **communication interface**, positioned at the lowest layer of the architecture, abstracts the physical transport mechanism (e.g., I^2C or SPI). It provides essential services to the core library, including:

- Data transmission (read/write operations)
- Attention (ATTN) signal detection and notification
- Transport-layer initialization and termination

This layered architecture ensures a clean separation of concerns between platform-independent logic and hardware-specific operations.

The core library does not make any direct calls to platform-specific or standard C runtime functions (e.g., strcpy, malloc, or memset). Instead, such functionality is accessed exclusively through the PAL, which must be implemented by the integrator to suit the target environment. PAL provides the minimal runtime services required by the core library and serves as its only dependency on platform-level facilities. This abstraction provides modularity, facilitates reuse across diverse hardware and operating systems, and simplifies integration in embedded environments with varying runtime constraints.

2.2. TouchComm Driver Contents

The TouchComm reference driver release package includes all necessary components for integrating Synaptics touch solutions into a Linux-based environment. Specifically, it consists of the following:

- Source code:

source/synaptics_tcm2/
|
|__**tcm/**
| | TouchComm Core Library
| |
| |__**synaptics_touchcom_core_dev.h**
| | The topmost header file declares and defines the main structure for the device context.
| |__**synaptics_touchcom_platform.h**
| | The header file declares the essential controls and links to the platform-specific interface.
| |__**synaptics_touchcom_core_v1.c**
| | Source code implements TouchComm version 1 command-response protocol.
| |__**synaptics_touchcom_core_v2.c**
| | Source code implements TouchComm version 2 command-response protocol.
| |__**synaptics_touchcom_func_base.[c / h]**
| | Source code implements the foundational functions supported.
| |__**synaptics_touchcom_func_touch.[c / h]**
| | Source code implements touch related functions and the handling of touched data.
| |__**synaptics_touchcom_func_reflash.[c / h]**
| | Source code implements firmware update related functions.
| |__**synaptics_touchcom_func_reflash_tddi.[c / h]**
| | Source code implements TDDI firmware update related functions.
| |__**synaptics_touchcom_func_reflash_image.[c / h]**
| | Source code implements relevant helpers for firmware image file.
| |__**synaptics_touchcom_func_reflash_ihex.[c / h]**
| Source code implements relevant helpers for firmware iHex file for touch/display devices.
|
|__**testing/**
| | Cluster of production test items
| |
| |__**synaptics_touchcom_testing.h**
| | The header file declares the interface for the library of production testing.
| |__**synaptics_touchcom_testing_*.c**
| Source code implements associated test item.
|
|__**syna_tcm2.[c / h]**
| Reference code of TouchComm device driver.
|__**syna_tcm2_runtime.h**
| Reference code abstracts platform-specific headers and C runtime APIs.
|__**syna_tcm2_platform.h**
| Reference code declares the platform-specific or hardware relevant data.
|__**syna_tcm2_platform_i2c.c**
| Reference code to communicate with the touch controller over I2C.
|__**syna_tcm2_platform_spi.c**
| Reference code to communicate with the touch controller over SPI.
|__**syna_tcm2_cdev.[c / h]**
| Reference code implements the character device node and IOCTL interface.
|__**syna_tcm2_sysfs.c**
| Reference code implements the kernel attributes.
|__**syna_tcm2_testing.c**
| Reference code implements the testing functions.
|__**Kconfig**
|__**Makefile**

- Examples for kernel configurations for the reference:

reference_configuration/kernel/arch/arm64/configs/
|
|__**defconfig_[i2c | spi]_example**
| Examples of TouchComm I2C / SPI device driver.
|__**defconfig_tddi_[i2c | spi]_example**
Examples of TouchComm I2C / SPI device driver for touch/display (TDDI) products.

reference_configuration/kernel/arch/arm64/boot/dts/overlays/
|
|__***.dts**
Examples of device tree.

3. Driver Integration

This section outlines the process for integrating the Synaptics TouchComm driver into a target system.

The reference driver provided in the release package has been integrated and verified on the Linux kernel using a Synaptics internal development board in an Android environment.

3.1. Driver Porting

The steps below describe the generic process for integrating the TouchComm reference driver into a target platform. Depending on the kernel BSP and platform architecture, some adjustments or additional customization may be necessary.

1. **Copy Source Files**
   Copy the synaptics_tcm2 directory from the source/ folder in the release package into the kernel source tree of the target platform, typically under kernel/drivers/input/touchscreen/.
2. **Review Runtime Abstraction Header**
   Open syna_tcm2_runtime.h to verify whether the required C runtime functions are supported on the target platform. Modify this file as needed to suit the platform's C runtime environment.
3. **Modify Makefile**
   Add the below line to the Makefile in the drivers/input/touchscreen directory of the kernel source tree

   obj-$(CONFIG_TOUCHSCREEN_SYNA_TCM2) += synaptics_tcm2/
4. **Modify Kconfig**
   Add the following line to the Kconfig file in the same directory.

   source "drivers/input/touchscreen/ synaptics_tcm2/Kconfig"
5. **Update defconfig**
   Refer to the sample defconfig in the driver package and apply the relevant configuration options to the target platform's defconfig file:
   a. Enable the Synaptics TouchComm driver.

      CONFIG_TOUCHSCREEN_SYNA_TCM2
   b. Select the communication bus.
      – For I^2C

        CONFIG_TOUCHSCREEN_SYNA_TCM2_I2C
      – For SPI

        CONFIG_TOUCHSCREEN_SYNA_TCM2_SPI
   c. Select the TouchComm protocol version.
      – For Protocol v1

        CONFIG_TOUCHSCREEN_SYNA_TCM2_TOUCHCOMM_VERSION_1
      – For Protocol v2

        CONFIG_TOUCHSCREEN_SYNA_TCM2_TOUCHCOMM_VERSION_2

d. Enable optional modules based on requirements.

– Support FW update

CONFIG_TOUCHSCREEN_SYNA_TCM2_REFLASH

– Support sysfs attribute

CONFIG_TOUCHSCREEN_SYNA_TCM2_SYSFS

– Support production testing

CONFIG_TOUCHSCREEN_SYNA_TCM2_TESTING

– Support Touch/Display (TDDI)

CONFIG_TOUCHSCREEN_SYNA_TCM2_TDDI

6. **Update Device Tree**
Update the .dtsi device tree files in the kernel source tree of the target platform by referring to the example device tree files in the release package.
Reference files can be found at reference_configuration/kernel/arch/arm64/boot/dts/

3.2. Build-Time Configurations

In addition to the kernel module configuration, several build-time options are available for tuning behavior under specific operating scenarios. These are typically defined in the following header files:

- source/synaptics_tcm2/syna_tcm2_platform.h
  - RD_CHUNK_SIZE
    Maximum number of bytes allowed in a single read transfer, '0' indicates no limit.
  - WR_CHUNK_SIZE
    Maximum number of bytes allowed in a single write transfer, '0' indicates no limit.
- source/synaptics_tcm2/syna_tcm2.h
  - RESET_ON_CONNECT
    Perform a chip reset before device initialization.
  - RESET_ON_RESUME
    Issue a software reset upon system resume.
  - LOW_POWER_MODE
    Enable power-saving mode when the device enters suspend state.
  - ENABLE_WAKEUP_GESTURE
    Enable gesture detection in low-power mode.
  - USE_CUSTOM_TOUCH_REPORT_CONFIG
    Apply the custom format of touch report configuration.
  - STARTUP_REFLASH
    Attempt to start the sequence of firmware update after the initialization.
  - ENABLE_EXTERNAL_FRAME_PROCESS
    Enable support for kernel FIFO storing the requested frames.
  - IS_TDDI_MULTICHIP
    Enable for touch/display products that use a multichip TDDI architecture.

3.3. Implementations of Runtime Abstraction

syna_tcm2_runetime.h aims to abstract OS-specific headers and C runtime. The following are few essentials to review, and please make the associated modifications if needed.

- Logs printing

  #define LOGD(log, ...) – C macros to print out debug messages
  #define LOGI(log, ...) – C macros to print out messages of common information
  #define LOGE(log, ...) – C macros to print out error messages
  #define LOGN(log, ...) – C macros to print out noticed messages
  #define LOGW(log, ...) – C macros to print out warning messages

- Atomic operations

  The structure, syna_pal_atomic_t, and its relevant macros such as ATOMIC_SET and ATOMIC_GET

- Sleep functions

- Memory allocation and management

  syna_pal_mem_alloc – memory allocation
  syna_pal_mem_free – memory removal
  syna_pal_mem_cpy – copy the data from the location to the destination

  Mutex operationsDefine the specific structure, syna_pal_mutex_t, and its associated operations.

  syna_pal_mutex_alloc – allocation of a mutex object
  syna_pal_mutex_free – removal of a mutex object
  syna_pal_mutex_lock – acquire a mutex
  syna_pal_mutex_unlock – unlock the mutex lock

- Event completion operations

  The structure, syna_pal_completion_t, and its associated operations.

  syna_pal_completion_alloc – allocation of a completion object
  syna_pal_completion_free – removal of a completion object
  syna_pal_completion_complete – send a completion event
  syna_pal_completion_wait_for – wait for the event in the given period

3.4. Implementations of Platform-Specific Interface

The tcm_hw_platform structure defined in the library file, tcm/synaptics_touchcom_platform.h, declares the essential platform-specific utilities required for abstracting communication with the touch controller. When initializing the TouchComm core library, the following operations must be implemented within this structure:

- Data writing (ops_write_data)

  Implementation to send data to the touch controller, either directly over the communication bus or through a communication subsystem.

- Data reading (ops_read_data)

  Implementation to read data from the touch controller, either directly over the communication bus or through a communication subsystem.

- Wait for ATTN assertion (ops_wait_for_attn)

  Implementation to block and wait until the ATTN (attention) signal from the device is asserted.

- Enable/disable ATTN line (ops_enable_attn)

  Implementation to temporarily enable or disable notifications from the ATTN signal.

In addition, several variables must also be set to configure TouchComm command processing:

- ATTN Support Capability (support_attn)

  A flag indicating whether the platform supports ATTN-driven notifications. Set this to True to enable command processing based on ATTN assertion. If set to False, command processing will rely on polling.

- Data Chunk Size (rd_chunk_size , wr_chunk_size)

  These variables specify the maximum number of bytes transferred in a single read or write operation, respectively.

3.5. Building of TouchComm Driver

After completing source porting and configuration, the TouchComm driver can be built as either a built-in kernel component or a loadable module, depending on the settings in the platform's defconfig file.

To ensure proper compilation, include the TouchComm core library path in the driver's Makefile.

ccflags-y += -I<path-to-touchcomm-core-library>

Replace <path-to-touchcomm-core-library> with the actual path to the core library.

Figure 3 is the example of Makefile for the reference.

```
DIR := $(dir $(realpath $(lastword $(MAKEFILE_LIST))))

TCM_CORE=tcm/

ccflags-y += -I$(DIR)
ccflags-y += -I$(DIR)$(TCM_CORE)


synaptics_tcm2$(NAME)-objs := syna_tcm2.o
synaptics_tcm2$(NAME)-objs += syna_tcm2_cdev.o
synaptics_tcm2$(NAME)-objs += \
                                        $(TCM_CORE)synaptics_touchcom_func_base.o \
                                        $(TCM_CORE)synaptics_touchcom_func_touch.o

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_TOUCHCOMM_VERSION_1),y)
        synaptics_tcm2$(NAME)-objs += $(TCM_CORE)synaptics_touchcom_core_v1.o
endif

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_TOUCHCOMM_VERSION_2),y)
        synaptics_tcm2$(NAME)-objs += $(TCM_CORE)synaptics_touchcom_core_v2.o
endif

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_REFLASH),y)
ifeq ($${CONFIG_TOUCHSCREEN_SYNA_TCM2_TDDI},y)
        synaptics_tcm2$(NAME)-objs += $(TCM_CORE)synaptics_touchcom_func_reflash_tddi.o
else
        synaptics_tcm2$(NAME)-objs += $(TCM_CORE)synaptics_touchcom_func_reflash.o
endif
endif

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_I2C),y)
        synaptics_tcm2$(NAME)-objs += syna_tcm2_platform_i2c.o
endif

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_SPI),y)
        synaptics_tcm2$(NAME)-objs += syna_tcm2_platform_spi.o
endif

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_SYSFS),y)
        synaptics_tcm2$(NAME)-objs += syna_tcm2_sysfs.o
endif

ifeq ($(CONFIG_TOUCHSCREEN_SYNA_TCM2_TESTING),y)
        synaptics_tcm2$(NAME)-objs += syna_tcm2_testing.o
endif

obj-$(CONFIG_TOUCHSCREEN_SYNA_TCM2) += synaptics_tcm2$(NAME).o
```

Figure 3. Example of Makefile

4. Driver Implementation

This section describes the typical sequence implemented in the TouchComm reference driver.

4.1. Startup Sequence

The figure below, Figure 4, shows the typical sequence for the startup initialization including the allocation of the object of TouchComm core library.

1. Assume that the porting of TouchComm core library is done at Section **Error! Reference source not found.**.
2. Allocate the object of TouchComm core library through *syna_tcm_allocate_device()*
3. Start to communicate with touch controller.
    a. Power on.
    b. Set the hardware reset pin to the non-reset state.
    c. Detect and connect to TouchComm firmware.
    d. Get the current TouchComm firmware mode.
    e. Perform the initialization of touch reporting.
    f. Request an IRQ and register the interrupt handling routine.

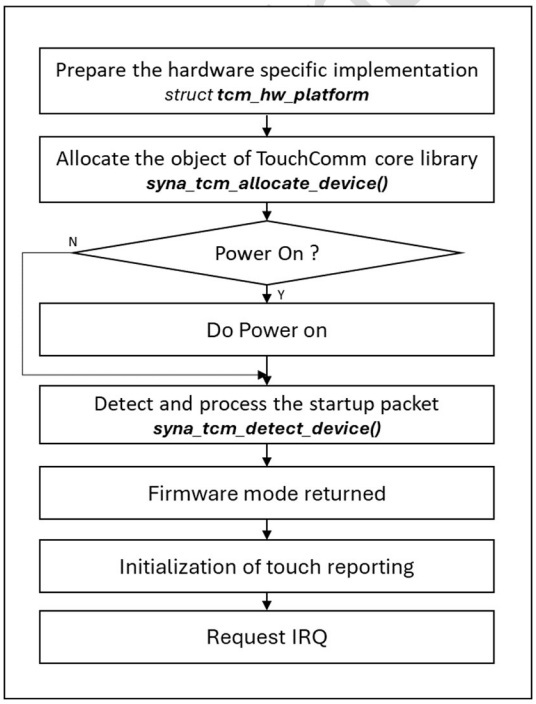

Figure 4. Typical Start-up Sequence

4.2. Initialization of Touch Reporting

This section describes the sequence initializing the touch report reporting.

1. Confirm that the application firmware is running.
2. Get APP_INFO packet through the API, *syna_tcm_get_app_info()*.
3. Set up the format of touch report in case of needs, use the built-in format otherwise.
4. Preserve the touch report config through the API, *syna_tcm_preserve_touch_report_config()*.
5. Register the target function handling the touch data through the API, *syna_tcm_set_report_dispatcher()*. So that, once a touch report is read in, the registered function will be invoked, and then the touch data will be processed properly.

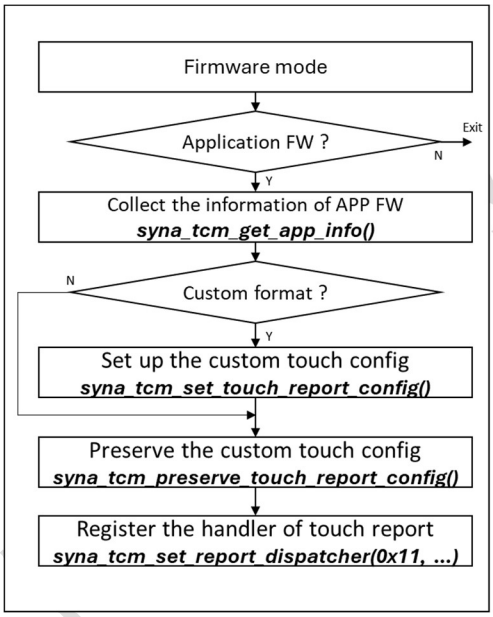

Figure 5. Flow of Touch Reporting Initialization

4.3. Handling of ATTN Assertion

ATTN (Attention) is the signal between the host and the device. The assertion of ATTN is used to notify the host messages whenever the firmware generates reports or responses to commands from the host. Typically, ATTN signal will be back to the normal state after the message is read out.

As shown in Figure 6, there are only a few steps to handle the situation of ATTN assertion.

1. Receive the notification of ATTN assertion.
2. Read out the message generated by firmware, for example, call *syna_tcm_get_event_data()*

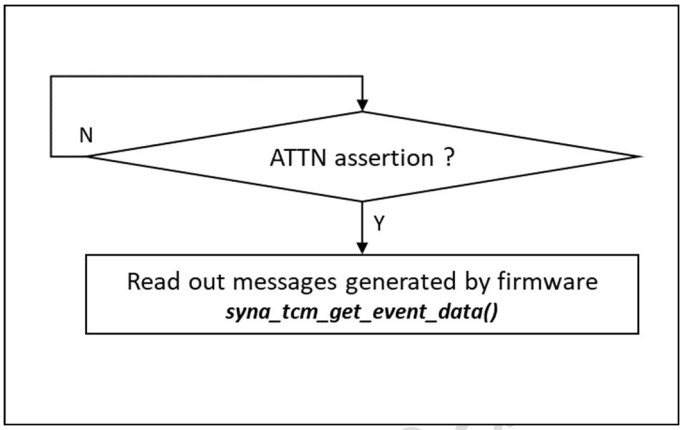

Figure 6. Typical Sequence for Handling of ATTN Assertion

The device may generate multiple different types of reports depending on its features or modes. if pending data in firmware is still waiting to read, ATTN signal may be still asserted even one message has been read out. In this case, host shall keep retrieving the messages as Figure 6 until the ATTN de-assertion.

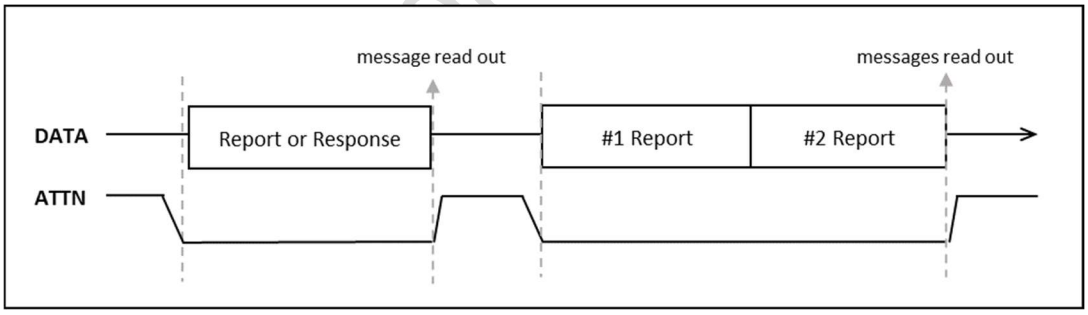

Figure 7. Example Waveform of ATTN Assertion

4.4. Touch Reports

TouchComm touch (0x11) report is designed to pass touch data to the host.

Figure 8 briefly illustrates the timeline of touch reporting. When a finger touches the sensor, TouchComm firmware will generate the first touch report marked with “finger-down” and then assert the ATTN signal so the host will read out the report. After that, touch report along with ATTN asserted will be generated at regular intervals until the finger lifting. Once lifting, ATTN line will be back to the normal state after the last report of “finger-up” is read out.

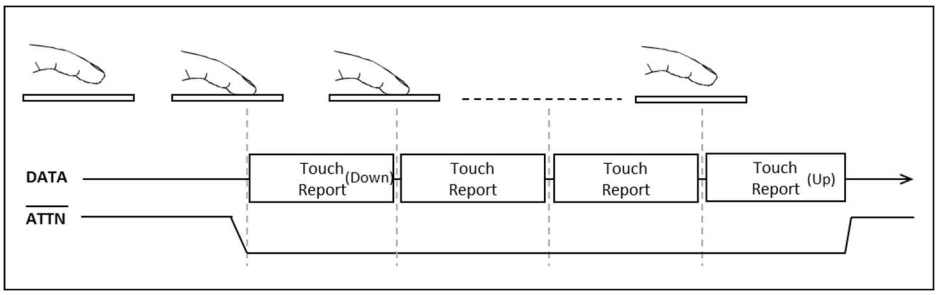

Figure 8. Example of Touch Reports Generating

Based on the implementations of TouchComm driver, if the dispatcher of touch report has been properly registered through *syna_tcm_set_report_dispatcher(), syna_tcm_get_event_data()* could read out the one touch report and then dispatch the data to the proper function. After that, *syna_tcm_parse_touch_report()* can obtain the touched data such as XY position from the touch report.

The following is the summary of the process of touch report.

1. Retrieve the touch report by calling *syna_tcm_get_event_data()* when ATTN is asserted.
2. The registered handler shall be invoked if a touch report has been read in successfully. Inside the handler, *syna_tcm_parse_touch_report()* is the helper to parse the touched data.

Please note that the format of touch report is configurable. The host can set and specify exactly what items get reported each time a touch report is generated.

In TouchComm driver, USE_CUSTOM_TOUCH_REPORT_CONFIG is set to enable the custom format of touch report configuration.

4.5. Firmware Update or Recovery

TouchComm driver supports the feature of firmware update, and the details of process had been implemented and wrapped up inside the TouchComm core library. Few steps are required to trigger the firmware update.

1. Load the firmware image file from the file system and prepare the binary data to update, for example using *request_firmware()* to load firmware in Linux kernel.
2. Do firmware update through *syna_tcm_do_fw_update()*.
   Be noted that, for touch/display products, please use *syna_tcm_tddi_do_fw_update()* instead.
3. At the end of a successful firmware update, application firmware shall be loaded and running functionally.

5. Software Interface

The Synaptics TouchComm driver exposes a user-space interface via IOCTL (Input/Output Control), which is the standard mechanism for issuing device-specific commands in Unix and Unix-like operating systems. This allows applications to interact with the driver for control, configuration, and diagnostic purposes beyond standard input event reporting.

5.1. Device Node

Upon successful driver installation, a character device node **/dev/tcm*** is created. This node serves as the communication channel between user-space applications and the TouchComm driver in the kernel. The node may depend on the number of devices or driver instances present in the system.

5.2. IOCTL Operations

The TouchComm driver defines a set of custom IOCTL commands to support various device operations, that is available at the syna_tcm2_sysfs.c.

6. Revision History

Revision	Description
1.8	Initial release
1.9	Update the section of driver implementation
1.10	Simplify and refine the description

Copyright

Trademarks

Synaptics, the Synaptics logo, Design Studio, are trademarks or registered trademarks of Synaptics Incorporated or its affiliates in the United States and/or other countries.

Android and Google are trademarks or registered trademarks of Google LLC. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries. All other trademarks are the properties of their respective owners.

Notice

Synaptics Confidential

This document contains information that is proprietary to Synaptics Incorporated ("Synaptics"). The holder of this document shall treat all information contained herein as confidential, shall use the information only for its intended purpose, and shall not duplicate, disclose, or disseminate any of this information in any manner unless Synaptics has otherwise provided express, written permission.

Use of the materials may require a license of intellectual property from a third party or from Synaptics. This document conveys no express or implied licenses to any intellectual property rights belonging to Synaptics or any other party. Synaptics may, from time to time and at its sole option, update the information contained in this document without notice.

INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS-IS," AND SYNAPTICS HEREBY DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES OF NON-INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS. IN NO EVENT SHALL SYNAPTICS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF THE INFORMATION CONTAINED IN THIS DOCUMENT, HOWEVER CAUSED AND BASED ON ANY THEORY OF LIABILITY, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, AND EVEN IF SYNAPTICS WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. IF A TRIBUNAL OF COMPETENT JURISDICTION DOES NOT PERMIT THE DISCLAIMER OF DIRECT DAMAGES OR ANY OTHER DAMAGES, SYNAPTICS' TOTAL CUMULATIVE LIABILITY TO ANY PARTY SHALL NOT EXCEED ONE HUNDRED U.S. DOLLARS.

Public Notice

Use of the materials may require a license of intellectual property from a third party or from Synaptics. This document conveys no express or implied licenses to any intellectual property rights belonging to Synaptics or any other party. Synaptics may, from time to time and at its sole option, update the information contained in this document without notice.

INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS-IS," WITH NO EXPRESS OR IMPLIED WARRANTIES, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND ANY WARRANTIES OF NON-INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHTS. IN NO EVENT SHALL SYNAPTICS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR IN CONNECTION WITH THE USE OF THE INFORMATION CONTAINED IN THIS DOCUMENT, HOWEVER CAUSED AND BASED ON ANY THEORY OF LIABILITY, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, AND EVEN IF SYNAPTICS WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. IF A TRIBUNAL OF COMPETENT JURISDICTION DOES NOT PERMIT THE DISCLAIMER OF DIRECT DAMAGES OR ANY OTHER DAMAGES, SYNAPTICS' TOTAL CUMULATIVE LIABILITY TO ANY PARTY SHALL NOT EXCEED ONE HUNDRED U.S. DOLLARS.

Contact Us

Visit our website at www.synaptics.com to locate the Synaptics office nearest you.