

**UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION**

SOLAS OLED LTD.,

Plaintiff,

vs.

SAMSUNG ELECTRONICS CO., LTD.,
SAMSUNG ELECTRONICS AMERICA,
INC.,

Defendants.

CASE NO. 2:21-cv-00105

Complaint for Patent Infringement

JURY DEMANDED

Complaint for Patent Infringement

Plaintiff Solas OLED Ltd. (“Solas”) files this complaint against Samsung Electronics Co., Ltd. (“SEC”) and Samsung Electronics America, Inc. (“SEA”) (each a “Defendant” and, collectively, “Defendants”), alleging infringement of U.S. Patent Nos. 9,292,144 and 8,526,767 (“Patents-in-Suit”). The Accused Products are the OLED panel displays made, used, offered for sale, sold, imported by Defendants in the United States and supplied by Defendants to customers and integrated into electronic devices sold in the United States.

Plaintiff Solas OLED and the Patents-in-Suit.

1. Plaintiff Solas is a technology licensing company organized under the laws of Ireland, with its headquarters at The Hyde Building, Suite 23, The Park, Carrickmines, Dublin 18, Ireland.

2. Solas is the owner of U.S. Patent No. 9,292,144, entitled “Touch-Sensor-Controller Sensor Hub,” which issued March 22, 2016 (the “’144 patent”). A copy of the ’144 patent is attached to this complaint as Exhibit 1.

3. Solas is the owner of U.S. Patent No. 8,526,767, entitled “Gesture Recognition,” which issued September 3, 2013 (the “’767 patent”). A copy of the ’767 patent is attached to this complaint as Exhibit 2.

Defendants and the Accused Products.

4. On information and belief, Defendant Samsung Electronics Co., Ltd. (“SEC”) is a corporation organized under the laws of South Korea, with its principal place of business at 129, Samsung-Ro, YeongTong-Gu, Suwon-Si, Gyonggi-Do, 443-742, South Korea.

5. On information and belief, Defendant Samsung Electronics America, Inc. (“SEA”) is a United States corporation organized under the laws of the State of New York, with its principal place of business at 85 Challenger Road, Ridgefield Park, New Jersey 07660. SEA is a wholly-owned subsidiary of SEC. SEA distributes certain Samsung consumer electronics products, including the Accused Products, in the United States.

6. The Accused Products are laptop computers, mobile phones and tablets made, used, offered for sale, sold, imported by Defendants in the United States, including without limitation the Samsung laptop computers and Galaxy mobile phones and tablet devices.

Jurisdiction and Venue.

7. This action arises under the patent laws of the United States, Title 35 of the United States Code. This Court has original subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

8. This Court has personal jurisdiction over Defendants in this action because Defendants have established minimum contacts with the United States as a whole such that the exercise of jurisdiction would not offend traditional notions of fair play and substantial justice. Defendants have purposefully directed activities at the United States, in particular, directing

Accused Products for sale to customers and distributors within the United States (including within this District) and engaging in sales and marketing efforts to generate and support such sales. Defendants have committed acts of infringement of Solas's patents giving rise to this action, such as by supplying to distributors and consumer device retailers the Accused Products in this District, including without limitation the Samsung ATIV and Galaxy laptop computers, tablets and phones accused of infringement in this case. Defendants, directly and through subsidiaries, intermediaries, and third parties, have committed and continues to commit acts of infringement in this District by, among other things, making, using, offering to sell, selling, and importing products that infringe the Asserted Patents.

9. Venue is proper in this District under 28 U.S.C. §§ 1391 and 1400(b). Defendant SEC is a foreign corporation. Venue is proper as to a foreign defendant in any district. 28 U.S.C. § 1391(c)(3). Defendant SEA has committed acts of infringement in this District and has regular and established places of business in this District.

Count 1 – Claim for infringement of the '144 patent.

10. Solas incorporates by reference each of the allegations in paragraphs 1–9 above and further alleges as follows:

11. On March 22, 2016, the United States Patent and Trademark Office issued U.S. Patent No. 9,292,144, entitled "Touch-Sensor-Controller Sensor Hub." Ex. 1.

12. Solas is the owner of the '144 patent with full rights to pursue recovery of royalties for damages for infringement, including full rights to recover past and future damages.

13. Each claim of the '144 patent is valid, enforceable, and patent-eligible.

14. Solas and its predecessors in interest have satisfied the requirements of 35 U.S.C. § 287(a) with respect to the '144 patent, and Solas is entitled to damages for Defendants' past infringement.

15. Defendants have directly infringed (literally and equivalently) and induced others to infringe the '144 patent by making, using, selling, offering for sale, or importing products that infringe the claims of the '144 patent and by inducing others to infringe the claims of the '144 patent without a license or permission from Solas, such as for example instructing users of the Accused Products to perform the patented methods of the '144 patent.

16. On information and belief, Defendants make, import, offer for sale, and sell certain infringing products in the United States. The Accused Products are, for example, consumer electronic devices manufactured by SEC and imported, sold, and offered for sale in the United States by SEA, including for example Samsung Galaxy mobile phones. The Accused Products all have touch controller chips for controlling one or more sensors in the Accused Products.



Samsung Galaxy S9

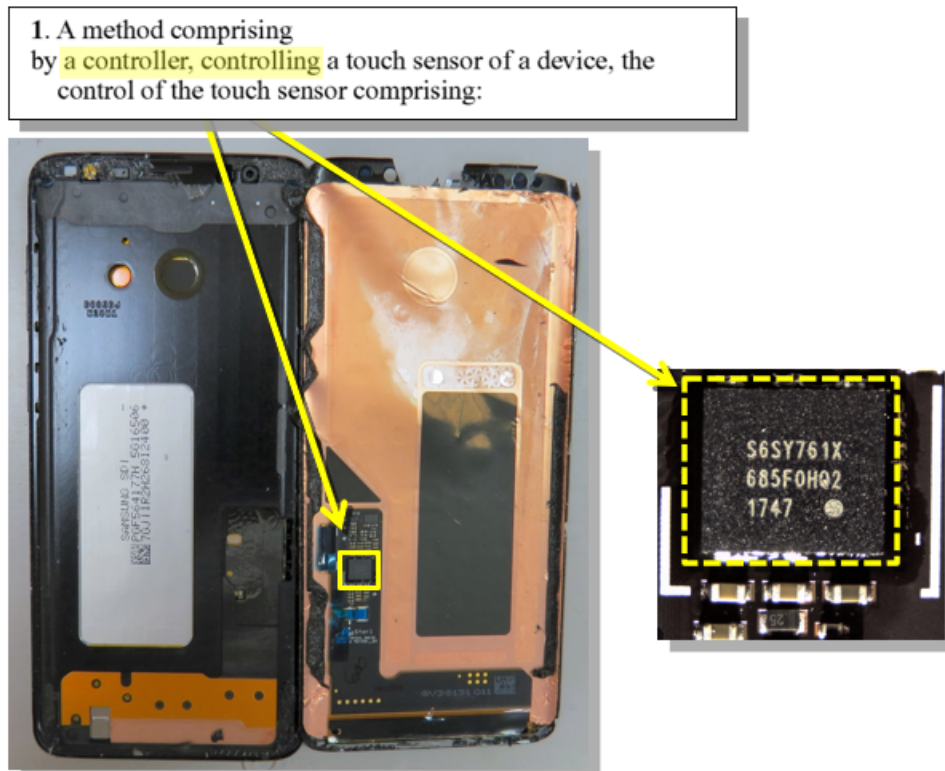


Samsung Galaxy S20

17. For example, claim 1 of the '144 patent claim “a method” as follows:

[1a] “by a controller, controlling a touch sensor of a device, the control of the touch sensor comprising;”

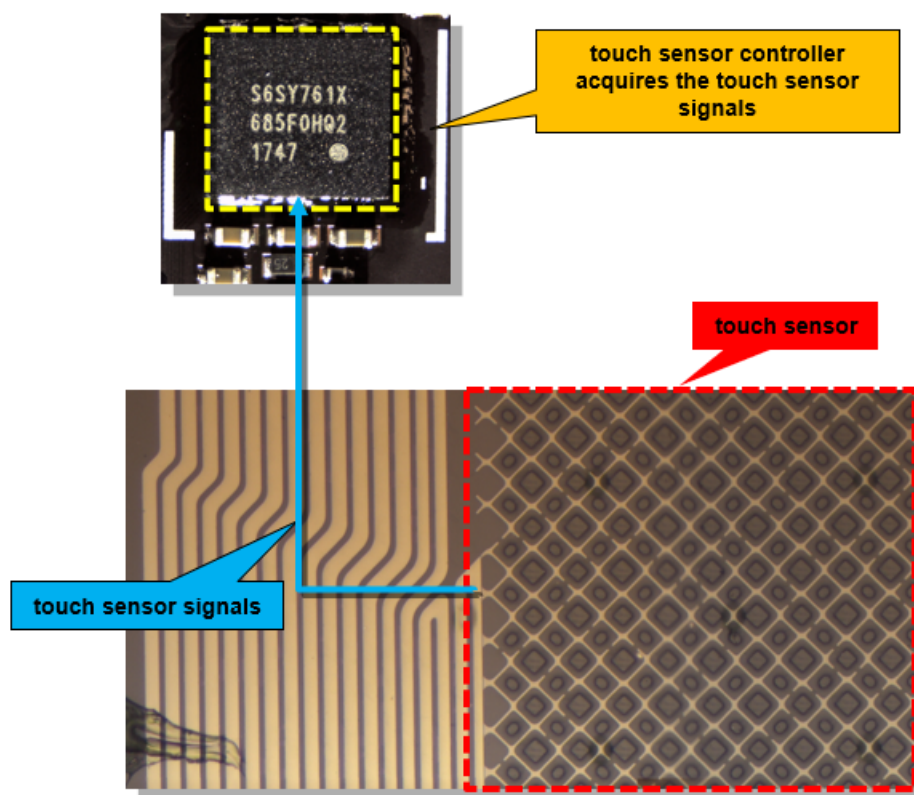
18. The Accused Products (such as the Galaxy S9, pictured below) comprise a touch controller which controls a touch sensor of a device.





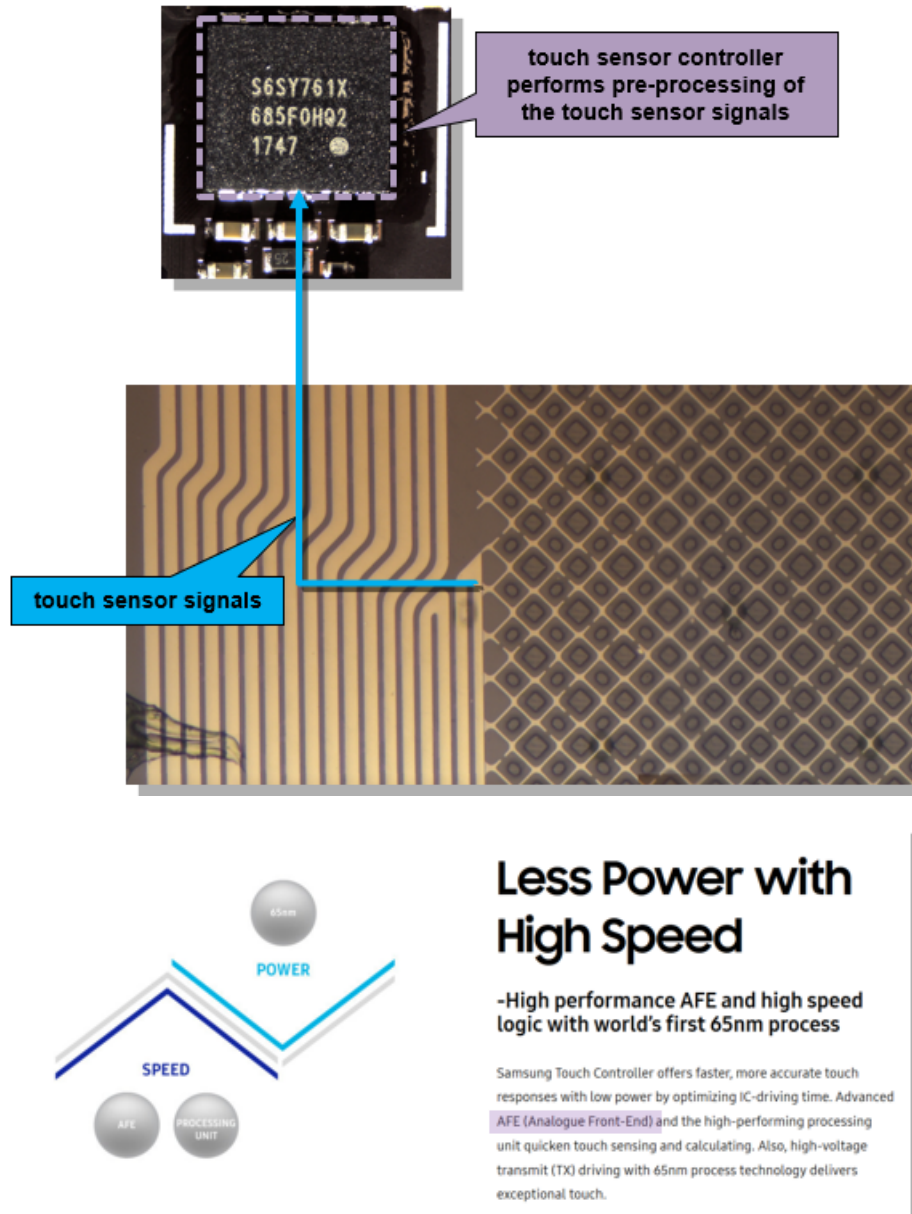
[1b] “acquisition of touch-sensor signals from the touch sensor;”

19. The Accused Products comprise a touch controller which controls a touch sensor by “acquisition of touch-sensor signals from the touch sensor”:



[1c] “pre-processing of the touch-sensor signals;”

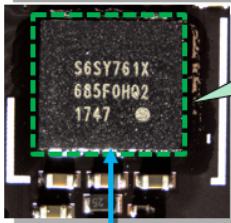
20. The Accused Products pre-process touch-sensor signals:



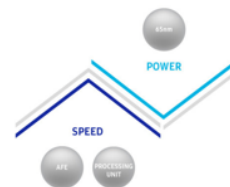
[1d] “and processing of the touch-sensor signals to determine: whether a touch or proximity input has occurred with respect to the touch sensor;”

21. The Accused Products process the touch-sensor signals to determine: whether a touch or proximity input has occurred with respect to the touch sensor:

and processing of the touch-sensor signals to determine: whether a touch or proximity input has occurred with respect to the touch sensor;



touch sensor controller performs processing of the touch sensor signals

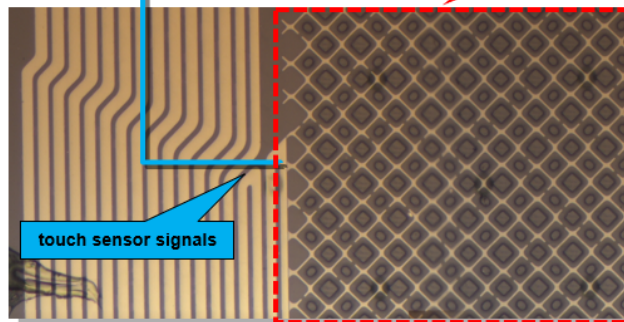


Less Power with High Speed

-High performance AFE and high speed logic with world's first 65nm process

Samsung Touch Controller offers faster, more accurate touch responses with low power by optimizing IC-driving time. Advanced AFE (Analogous Front-End) and the high-performing processing unit (quicker touch sensing and calculating). Also, high-voltage transmit (TX) driving with 65nm process technology delivers exceptional touch.

The touch controller performs processing of the touch-sensors signals to determine if a touch has occurred on the touch sensor.



touch sensor

touch sensor signals

```

325 #define SEC_TS_TOUCHTYPE_NORMAL 0
326 #define SEC_TS_TOUCHTYPE_HOVER 1
327 #define SEC_TS_TOUCHTYPE_FLIPCOVER 2
328 #define SEC_TS_TOUCHTYPE_GLOVE 3
329 #define SEC_TS_TOUCHTYPE_STYLUS 4
330 #define SEC_TS_TOUCHTYPE_PALM 5
331 #define SEC_TS_TOUCHTYPE_WET 6
332 #define SEC_TS_TOUCHTYPE_PROXIMITY 7
333 #define SEC_TS_TOUCHTYPE_JIG 8

```

determine touch/proximity inputs

```

1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093

```

```

if ((ts->coord[t_id].type == SEC_TS_TOUCHTYPE_NORMAL)
    || (ts->coord[t_id].type == SEC_TS_TOUCHTYPE_PALM)
    || (ts->coord[t_id].type == SEC_TS_TOUCHTYPE_WET)
    || (ts->coord[t_id].type == SEC_TS_TOUCHTYPE_GLOVE)) {
    if (ts->coord[t_id].action == SEC_TS_COORDINATE_ACTION_RELEASE) {
        s16 max_force_p = 0;
        u8 rbuf[2] = {0, 0};

        do_gettimeofday(&ts->time_released[t_id]);

        if (ts->time_longest < (ts->time_released[t_id].tv_sec - ts->time_pressed[t_id].tv_sec))
            ts->time_longest = (ts->time_released[t_id].tv_sec - ts->time_pressed[t_id].tv_sec);

        ret = sec_ts_i2c_read(ts, SEC_TS_READ_FORCE_SIG_MAX_VAL, rbuf, 2);
        if (ret < 0)
            input_err(true, &ts->client->dev,
                "ts: fail to read max pressure data\n",
                __func__);
        else
            max_force_p = (rbuf[0] & 0xFF) << 8 | (rbuf[1] & 0xFF);

        input_mt_slot(ts->input_dev, t_id);
        if (ts->plat_data->support_mt_pressure)
            input_report_abs(ts->input_dev, ABS_MT_PRESSURE, 0);
        input_mt_report_slot_state(ts->input_dev, MT_TOOL_FINGER, 0);

        if (ts->touch_count > 0)
            ts->touch_count++;
        if (ts->touch_count == 0) {
            input_report_key(ts->input_dev, BTN_TOUCH, 0);
            input_report_key(ts->input_dev, BTN_TOOL_FINGER, 0);
            ts->check_multi = 0;
        }
    }
}

```

The code excerpt from the touchscreen controller driver in the Samsung Galaxy S9 highlights the different modes of touch which the touchscreen controller can detect. This is achieved through processing of the touch-sensor signals from the touch sensor.

[1e] “if the touch or proximity input has occurred with respect to the touch sensor, a location of the touch or proximity input; and”

22. The Accused Products determine, if the touch or proximity input has occurred with respect to the touch sensor, a location of the touch or proximity input:

```

1130     } else if (ts->coord[t_id].action == SEC_TS_COORDINATE_ACTION_PRESS) {
1131         do_gettimeofday(&ts->time_pressed[t_id]);
1132
1133         ts->touch_count++;
1134         ts->all_finger_count++;
1135
1136         ts->max_z_value = max((unsigned int)ts->coord[t_id].z, ts->max_z_value);
1137         ts->min_z_value = min((unsigned int)ts->coord[t_id].z, ts->min_z_value);
1138         ts->sum_z_value += (unsigned int)ts->coord[t_id].z;
1139
1140         input_mt_slot(ts->input_dev, t_id);
1141         input_mt_report_slot_state(ts->input_dev, MT_TOOL_FINGER, 1);
1142         input_report_key(ts->input_dev, BTN_TOUCH, 1);
1143         input_report_key(ts->input_dev, BTN_TOOL_FINGER, 1);
1144
1145         input_report_abs(ts->input_dev, ABS_MT_POSITION_X, ts->coord[t_id].x);
1146         input_report_abs(ts->input_dev, ABS_MT_POSITION_Y, ts->coord[t_id].y);
1147         input_report_abs(ts->input_dev, ABS_MT_TOUCH_MAJOR, ts->coord[t_id].major);
1148         input_report_abs(ts->input_dev, ABS_MT_TOUCH_MINOR, ts->coord[t_id].minor);

```

touch has occurred

a location of the touch

The code excerpt from the touchscreen controller driver in the Samsung Galaxy S9 highlights that when a touch has occurred with respect to the touch sensor, the location of the touch (x and y coordinates) is output to the operating system running the driver.

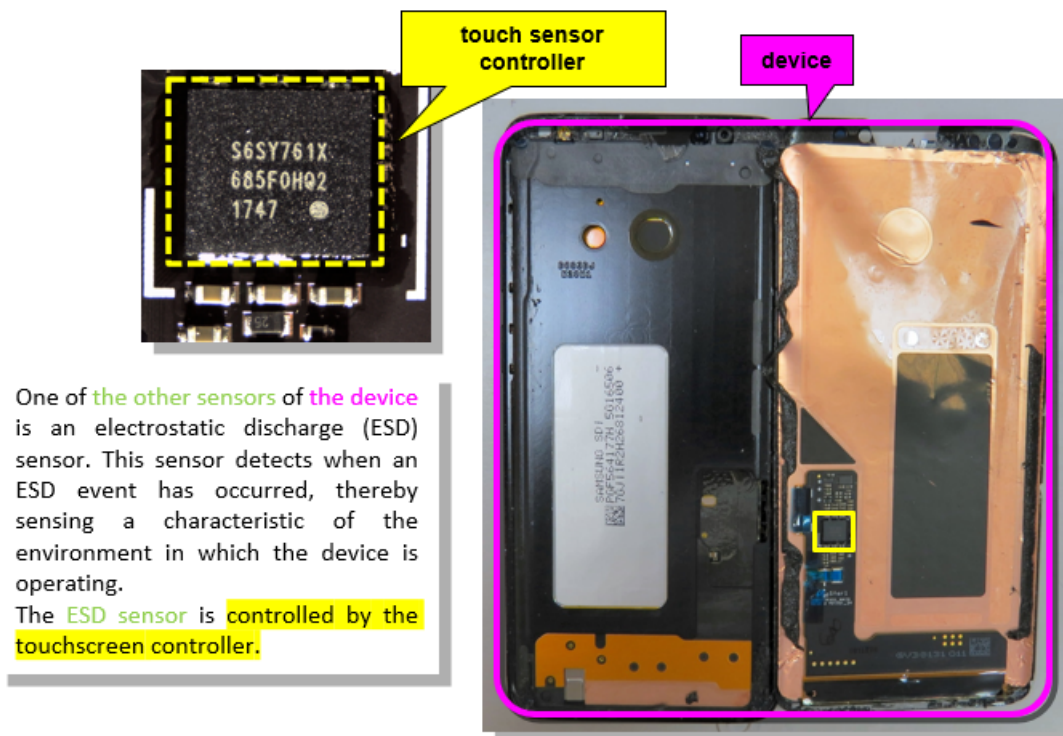
```

286 ABS_MT_POSITION_X
287 The surface X coordinate of the center of the touching ellipse.
288
289 ABS_MT_POSITION_Y
290 The surface Y coordinate of the center of the touching ellipse.
291
292

```

[1f] “by the controller, controlling one or more other sensors of the device, the control of the other sensors occurring at least in part concurrently with the acquisition of the touch-sensor signals from the touch sensor or the pre-processing of the touch-sensor signals.”

23. The Accused Products’ touch controllers control one or more other sensors of the device, the control of the other sensors occurring at least in part concurrently with the acquisition of the touch-sensor signals from the touch sensor or the pre-processing of the touch-sensor signals:

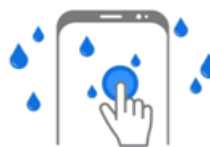


Flawless detection in any environment

-Anti-noise technology for accurate detection

Enhancing touch detection even in tough conditions such as high display noise, electrostatic discharge (ESD) and electromagnetic interference (EMI), Samsung Touch Controller offers diverse anti-noise technologies that prevent interference and deliver accurate and reliable touch experiences.

touch sensor controller



ENHANCED TOUCH DETECTION

one or more other sensors (ESD detection sensor)

control of other sensors (ESD detection sensor)

```

964
965     if ((p_event_status->type == TYPE_STATUS_EVENT_ERR) &&
966         (p_event_status->status_id == SEC_TS_ERR_EVENT_ESD)) {
967         input_err(true, &ts->client->dev, "%s: ESD detected. run reset\n", __func__);
968     }
969     #ifdef USE_RESET_DURING_POWER_ON
970     schedule_work(&ts->reset_work.work);
971     #endif
972 }
  
```

24. Defendants also knowingly and intentionally induce and contribute to infringement of the '144 patent in violation of 35 U.S.C. §§ 271(b) and 271(c). Through the

filing and service of this Complaint, Defendants have had knowledge of the '144 patent and the infringing nature of the Accused Products. Despite this knowledge of the '144 patent, Defendants continue to actively encourage and instruct its customers to use and integrate the accused products in ways that directly infringe the '144 patent. Defendants do so knowing and intending that their customers will commit these infringing acts. Defendants also continue to make, use, offer for sale, sell, and/or import the Accused Products, despite their knowledge of the '144 patent, thereby specifically intending for and inducing its customers to infringe the '144 patent through the customers' normal and customary use of the Accused Products.

25. Defendants have infringed multiple claims of the '144 patent, including independent claim 1. By way of example only, the normal and customary use of the accused Samsung Galaxy S9 phone infringes an exemplary claim of the '144 patent, as in the description set forth above, which Solas provides without the benefit of information about the Accused Products obtained through discovery.

26. Defendants have known how the Accused Products are made and have known, or have been willfully blind to the fact, that making, using, offering to sell, and selling the Accused Products to their customers, would constitute willful infringement of the '144 patent. Those products imported into and sold within the United States include, without limitation, Samsung laptop computers, Galaxy tablets and phones.

27. Defendants have induced, and continue to induce, infringement of the '144 patent by actively encouraging others (including its customers) to use, offer to sell, sell, and import the Accused Products. On information and belief, these acts include providing information and instructions on the use of the Accused Products; providing information, education and

instructions to its customers; providing the Accused Products to customers; and indemnifying patent infringement within the United States.

28. Solas has been damaged by Defendant's infringement of the '144 patent and is entitled to damages as provided for in 35 U.S.C. § 284, including reasonable royalty damages.

Count 2 – Claim for infringement of the '767 patent.

29. Solas incorporates by reference each of the allegations in paragraphs 1–28 above and further alleges as follows:

30. On September 3, 2013, the United States Patent and Trademark Office issued U.S. Patent No. 8,526,767, entitled "Gesture Recognition." Ex. 2.

31. Solas is the owner of the '767 patent with full rights to pursue recovery of royalties for damages for infringement, including full rights to recover past and future damages.

32. Each claim of the '767 patent is valid, enforceable, and patent-eligible.

33. Solas and its predecessors in interest have satisfied the requirements of 35 U.S.C. § 287(a) with respect to the '767 patent, and Solas is entitled to damages for Defendants' past infringement.

34. Defendants have directly infringed (literally and equivalently) and induced others to infringe the '767 patent by making, using, selling, offering for sale, or importing products that infringe the claims of the '767 patent and by inducing others to infringe the claims of the '767 patent without a license or permission from Solas, such as for example instructing users of the Accused Products to perform the patented methods of the '767 patent.

35. On information and belief, Defendants make, import, offer for sale, and sell certain infringing products in the United States. The Accused Products are, for example, consumer electronic devices manufactured by SEC and imported, sold, and offered for sale in the

United States by SEA, including for example Samsung Galaxy mobile phones. The Accused Products all have touch controller chips for controlling one or more sensors in the Accused Products.



Samsung Galaxy S9



Samsung Galaxy S20

36. For example, claim 1 of the '767 patent claim a “touch sensor device” as follows:

[1a] “a sensor having a sensitive area extending in at least one-dimension and arranged to output sense signals responsive to proximity of an object to the sensitive area;”

37. The Accused Products (such as the Galaxy S9, pictured below) comprise a sensor having a sensitive area extending in at least one-dimension and arranged to output sense signals responsive to proximity of an object to the sensitive area:

a sensor having a sensitive area extending in at least one-dimension and arranged to output sense signals responsive to proximity of an object to the sensitive area;

Controlling the touchscreen



- Do not allow the touchscreen to come into contact with other electrical devices. Electrostatic discharges can cause the touchscreen to malfunction.
- To avoid damaging the touchscreen, do not tap it with anything sharp or apply excessive pressure to it with your fingertips.
- Leaving the touchscreen idle for extended periods may result in afterimages (screen burn-in) or ghosting. Turn off the touchscreen when you do not use the device.



- The device may not recognise touch inputs close to the edges of the screen, which are outside of the touch input area.
- It is recommended to use fingers when you use the touchscreen.

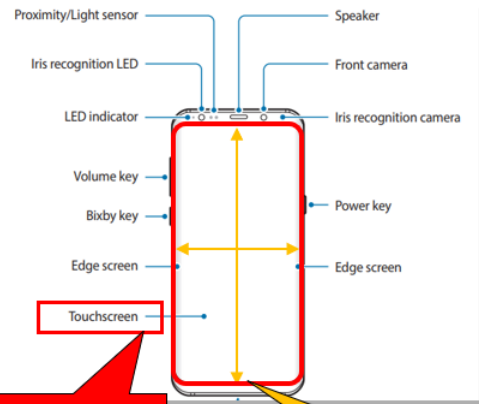
Tapping

Tap the screen.



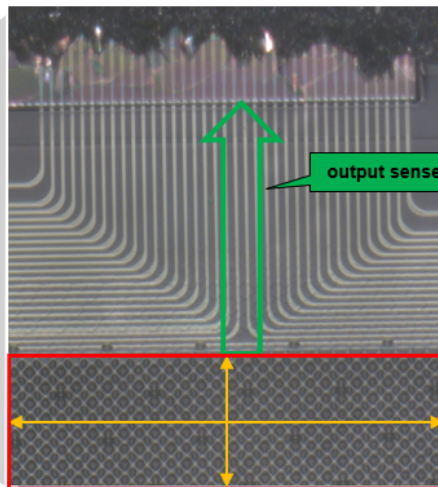
sensitive area

output sense signals
responsive to proximity
of an object



a sensor having a
sensitive area

extending in at least
one-dimension



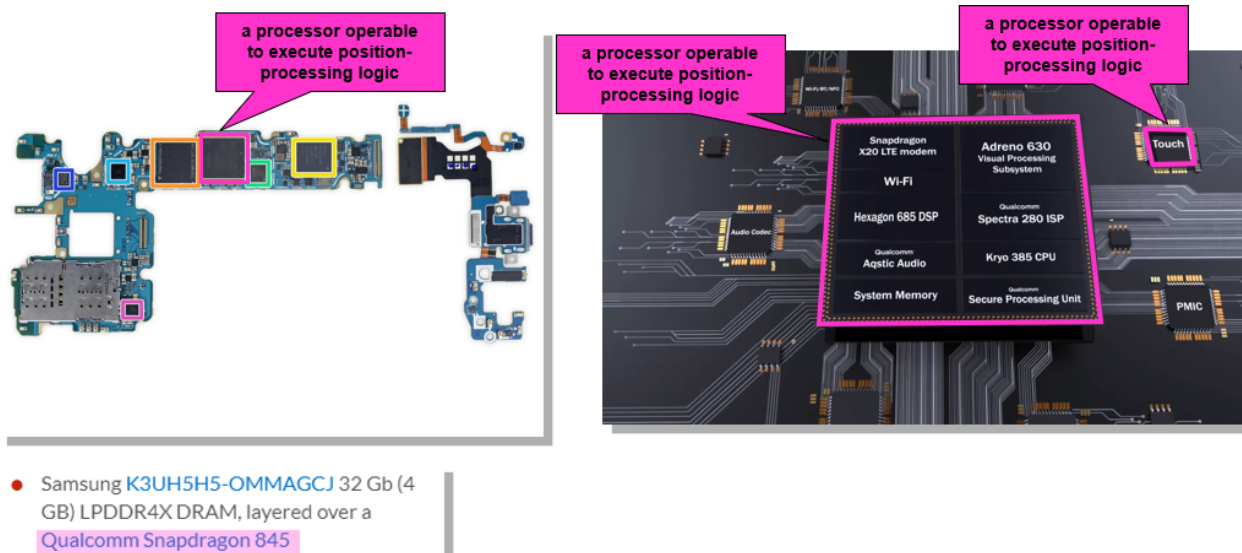
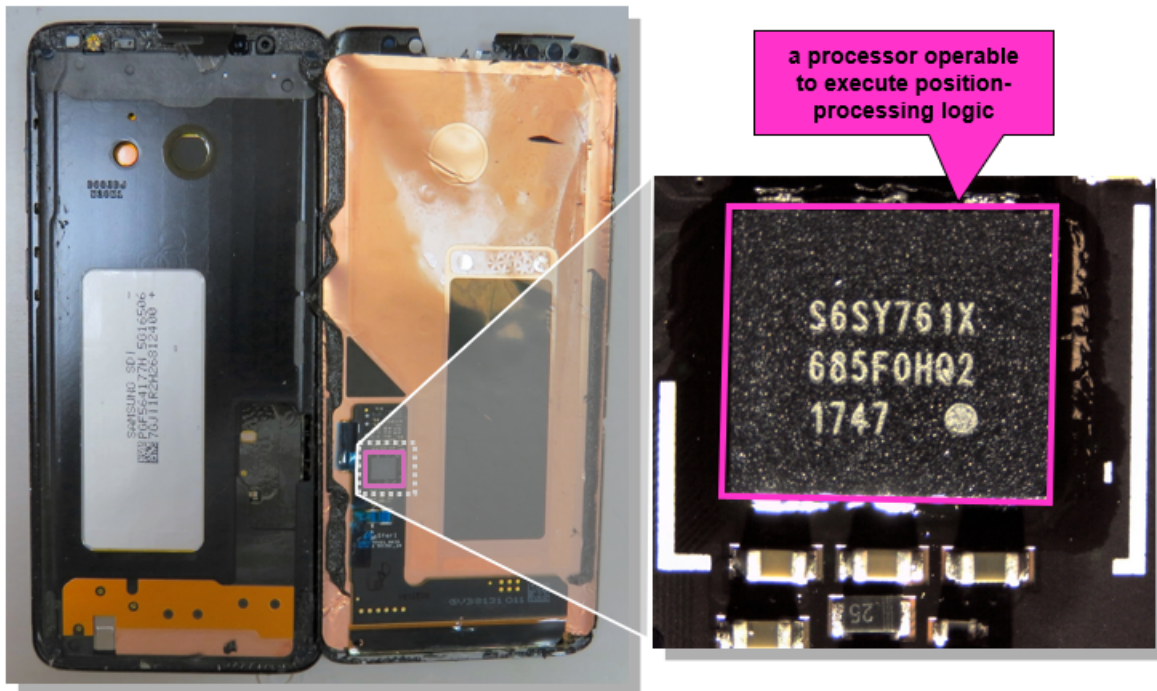
output sense signals

extending in at least
one-dimension

a sensor having a
sensitive area

[1b] “a processor operable to execute position-processing logic stored in one or more tangible media, the position-processing logic, when executed by the processor, configured to:”

38. The Accused Products comprise a processor operable to execute position-processing logic stored in one or more tangible media, the position-processing logic:



```

1 /* drivers/input/touchscreen/sec_ts.h
2  *
3  * Copyright (C) 2015 Samsung Electronics Co., Ltd.
4  * http://www.samsungsemi.com/
5  *
6  * Core file for Samsung TSC driver
7  *
8  * This program is free software; you can redistribute it and/or modify
9  * it under the terms of the GNU General Public License version 2 as
10  * published by the Free Software Foundation;
11  */
12
13 #ifndef SEC_TS_H
14 #define SEC_TS_H
15
16 #include <asm/unaligned.h>
17 #include <linux/completion.h>
18 #include <linux/crypt.h>
19 #include <linux/delay.h>
20 #include <linux/firmware.h>
21 #include <linux/gpio.h>
22 #include <linux/hrtimer.h>
23 #include <linux/i2c.h>
24 #include <linux/input.h>
25 #include <linux/input/sec.h>
26 #include <linux/input/sec_cmd.h>
27 #include <linux/interrupt.h>
28 #include <linux/io.h>
29 #include <linux/irq.h>
30 #include <linux/kernel.h>
31 #include <linux/module.h>
32 #include <linux/of_gpio.h>
33 #include <linux/platform_device.h>
34 #include <linux/regulator/consumer.h>
35 #include <linux/slab.h>
36 #include <linux/time.h>
37 #include <linux/uaccess.h>
38 #include <linux/vmalloc.h>
39 #include <linux/wakelock.h>
40 #include <linux/workqueue.h>

```

a processor operable
to execute position-
processing logic

position-processing logic
based on multi-touch (MT)
protocol

The Samsung touchscreen controller (TSC) and the Snapdragon 845 execute position processing logic based on the Multi-touch (MT) Protocol. This is evidenced by the use of the Multi-Touch "MT.h" library in the Samsung touchscreen controller driver.

```

1 Multi-touch (MT) Protocol
2 -----
3 Copyright (C) 2009-2010 Henrik Rydberg <rydberg@euromail.se>
4
5
6 Introduction
7 -----
8
9 In order to utilize the full power of the new multi-touch and multi-user
10 devices, a way to report detailed data from multiple contacts, i.e.,
11 objects in direct contact with the device surface, is needed. This
12 document describes the multi-touch (MT) protocol which allows kernel
13 drivers to report details for an arbitrary number of contacts.
14
15 The protocol is divided into two types, depending on the capabilities of the
16 hardware. For devices handling anonymous contacts (type A), the protocol
17 describes how to send the raw data for all contacts to the receiver. For
18 devices capable of tracking identifiable contacts (type B), the protocol
19 describes how to send updates for individual contacts via event slots.
20

```

[1c] “calculate positions of interactions with the sensitive area from an analysis of the sense signals; and output a times series of data indicative of the interaction positions on the sensor, the interactive positions corresponding to the touches; and”

39. The Accused Products calculate positions of interactions with the sensitive area from an analysis of the sense signals, and output a times series of data indicative of the interaction positions on the sensor, the interactive positions corresponding to the touches:

calculate positions of interactions with the sensitive area from an analysis of the sense signals; and output a times series of data indicative of the interaction positions on the sensor, the interaction positions corresponding to touches; and

calculate positions of interactions

touch

sensitive area

```

112  /**
113   * struct input_mt_pos - contact position
114   * @x: horizontal coordinate
115   * @y: vertical coordinate
116   */
117  struct input_mt_pos {
118      s16 x, y;
119  };

```

sensitive area

```

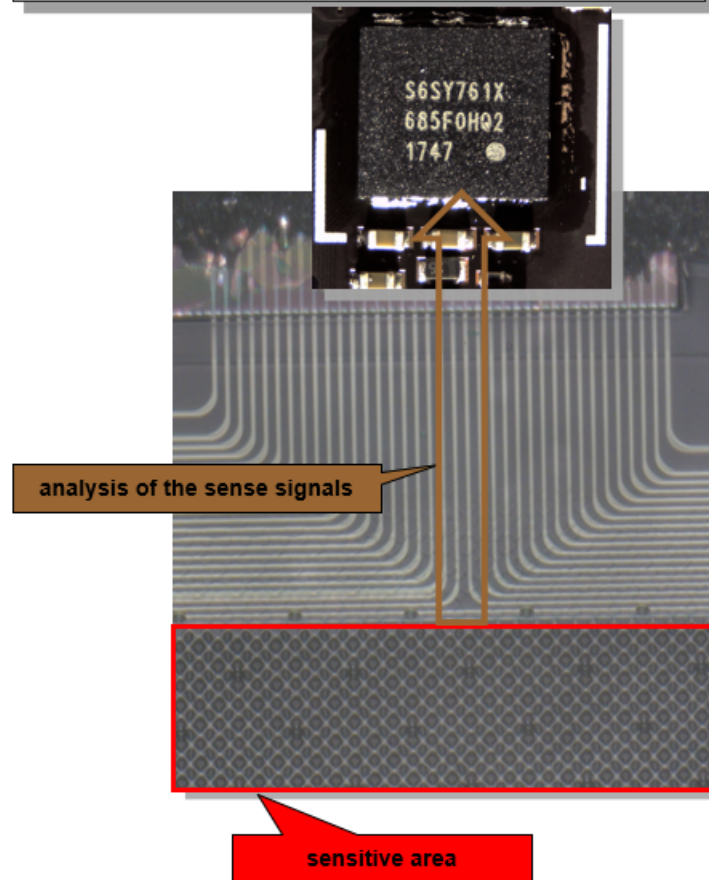
516  * Axis constant: X axis of a motion event.
517  * @x:
518  * <li>For a touch screen, reports the absolute X screen position of the center of
519  * the touch contact area. The units are display pixels.
520  * <li>For a touch pad, reports the absolute X surface position of the center of the
521  * contact area. The units are device-dependent; use @link InputDevice::getMotionRange()
522  * to query the effective range of values.
523  * <li>For a mouse, reports the absolute X screen position of the mouse pointer.
524  * The units are display pixels.
525  * <li>For a trackball, reports the relative horizontal displacement of the trackball.
526  * The value is normalized to a range from -1.0 (left) to 1.0 (right).
527  * <li>For a joystick, reports the absolute X position of the joystick.
528  * The value is normalized to a range from -1.0 (left) to 1.0 (right).
529  * </li>
530  * </li>
531  * </li>
532  *
533  * @see #getx(int)
534  * @see #getHistoricalX(int, int)
535  * @see MotionEvent.PointerCoords
536  * @see InputDevice::getMotionRange
537  */
538  public static final int AXIS_X = 0;
539
540  /**
541  * Axis constant: Y axis of a motion event.
542  * @y:
543  * <li>For a touch screen, reports the absolute Y screen position of the center of
544  * the touch contact area. The units are display pixels.
545  * <li>For a touch pad, reports the absolute Y surface position of the center of the
546  * contact area. The units are device-dependent; use @link InputDevice::getMotionRange()
547  * to query the effective range of values.
548  * <li>For a mouse, reports the absolute Y screen position of the mouse pointer.
549  * The units are display pixels.
550  * <li>For a trackball, reports the relative vertical displacement of the trackball.
551  * The value is normalized to a range from -1.0 (up) to 1.0 (down).
552  * <li>For a joystick, reports the absolute Y position of the joystick.
553  * The value is normalized to a range from -1.0 (up or far) to 1.0 (down or near).
554  * </li>
555  * </li>
556  * </li>
557  *
558  * @see #gety(int)
559  * @see #getHistoricalY(int, int)
560  * @see MotionEvent.PointerCoords
561  * @see InputDevice::getMotionRange
562  */
563  public static final int AXIS_Y = 1;

```

calculate positions of interactions

calculate positions of interactions

calculate positions of interactions with the sensitive area
from an analysis of the sense signals; and
output a times series of data indicative of the interaction
positions on the sensor, the interaction positions cor-
responding to touches; and



calculate positions of interactions with the sensitive area from an analysis of the sense signals; and
 output a times series of data indicative of the interaction positions on the sensor, the interaction positions corresponding to touches; and

Multi-touch (MT) Protocol

Copyright (C) 2009-2010 Henrik Rydberg <rydberg@euromail.se>

Introduction

touches

Interaction positions

In order to utilize the full power of the new multi-touch and multi-user devices, a way to report detailed data from multiple contacts, i.e., objects in direct contact with the device surface, is needed. This document describes the multi-touch (MT) protocol which allows kernel drivers to report details for an arbitrary number of contacts.

sensor

The protocol is divided into two types, on the capabilities of the hardware. For devices handling anonymous contacts (type A), the protocol describes how to send the raw data for all contacts to the receiver. For devices capable of tracking identifiable contacts (type B), the protocol describes how to send updates for individual contacts via event slots.

a time series of data

Drivers for type B devices separate contact packets by calling `input_mt_slot()`, with a slot as argument, at the beginning of each packet. This generates an `ABS_MT_SLOT` event, which instructs the receiver to prepare for updates of the given slot.

All drivers mark the end of multi-touch transfer by calling the usual `input_sync()` function. This instructs the receiver to act upon events accumulated since last `EV_SYN` REPORT and prepare to receive a new set of events/packets.

a time series of data

(reported via `mt_slot` event)

The main difference between the stateful protocol and the stateful type B slot protocol lies in the usage of identifiable contacts to reduce the amount of data sent to userspace. The slot protocol requires the use of the `ABS_MT_TRACKING_ID`, either provided by the hardware or computed from the raw data [5].

For type A devices, the kernel driver should generate an arbitrary enumeration of the full set of anonymous contacts currently on the surface. The order in which the packets appear in the event stream is not important. Event filtering and finger tracking is left to user space [3].

For type B devices, the kernel driver should associate a slot with each identified contact, and use that slot to propagate changes for the contact. Creation, replacement and destruction of contacts is achieved by modifying the `ABS_MT_TRACKING_ID` of the associated slot. A non-negative tracking id is interpreted as a contact, and the value -1 denotes an unused slot. A tracking id not previously present is considered new, and a tracking id no longer present is considered removed. Since only changes are propagated, the full state of each initiated contact has to reside in the receiving end. Upon receiving an MT event, one simply updates the appropriate attribute of the current slot.

calculate positions of interactions with the sensitive area from an analysis of the sense signals; and
 output a times series of data indicative of the interaction positions on the sensor, the interaction positions corresponding to touches; and

```

1  /* drivers/input/touchscreen/sec_ts.c
2  *
3  * Copyright (C) 2011 Samsung Electronics Co., Ltd.
4  * http://www.samsungsemi.com/
5  *
6  * Core file for Samsung TSC driver
7  *
8  * This program is free software; you can redistribute it and/or modify
9  * it under the terms of the GNU General Public License version 2 as
10 * published by the Free Software Foundation.
11 */
12
13 struct sec_ts_data *tsp_info;
14
15 #include "sec_ts.h"
16
17 struct sec_ts_data *ts_dup;
18
19
1130 } else if (ts->coord[t_id].action == SEC_TS_COORDINATE_ACTION_PRESS) {
1131     do_gettimeofday(&ts->time_pressed[t_id]);
1132     ts->touch_count++;
1133     ts->all_finger_count++;
1134
1135     ts->max_z_value = max((unsigned int)ts->coord[t_id].z, ts->max_z_value);
1136     ts->min_z_value = min((unsigned int)ts->coord[t_id].z, ts->min_z_value);
1137     ts->sum_z_value += (unsigned int)ts->coord[t_id].z;
1138
1139     input_mt_slot(ts->input_dev, t_id);
1140     input_mt_report_slot_state(ts->input_dev, MT_TOOL_FINGER, 1);
1141     input_report_key(ts->input_dev, BTN_TOUCH, 1);
1142     input_report_key(ts->input_dev, BTN_TOOL_FINGER, 1);
1143
1144     input_report_abs(ts->input_dev, ABS_MT_POSITION_X, ts->coord[t_id].x);
1145     input_report_abs(ts->input_dev, ABS_MT_POSITION_Y, ts->coord[t_id].y);
1146     input_report_abs(ts->input_dev, ABS_MT_TOUCH_MAJOR, ts->coord[t_id].major);
1147     input_report_abs(ts->input_dev, ABS_MT_TOUCH_MINOR, ts->coord[t_id].minor);
1148
  
```

The Samsung touchscreen controller (TSC) is capable of tracking multiple touches on the mutual-capacitive touchscreen of the Galaxy S9. The Samsung TSC outputs a time series of data for each identified contact or touch detected by the touchscreen which can be accessed by the TSC driver running on the main processor i.e. Snapdragon 845.

a time series of data

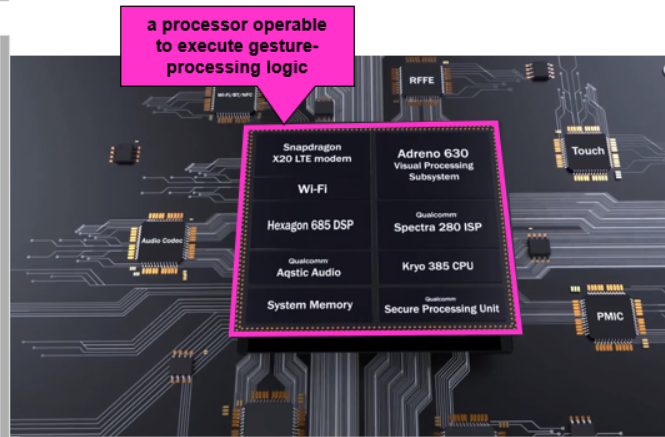
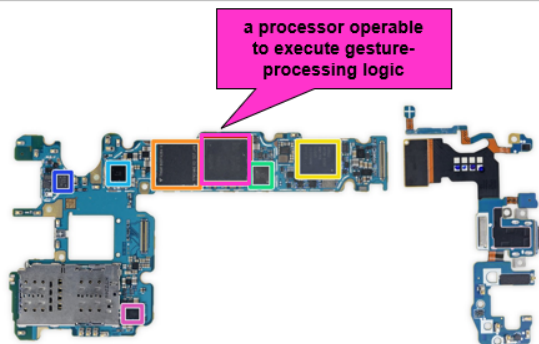
corresponding to touches

interaction positions

[1d] “a processor operable to execute gesture-processing logic stored in one or more tangible media, the gesture-processing logic”

40. The Accused Products have a processor operable to execute gesture-processing logic stored in one or more tangible media, the gesture-processing logic, when executed by the processor, configured to analyze the time series of data to distinguish one or more gesture inputs from the time series of data, the gesture-processing logic being coded with gesture-recognition code comprising a plurality of state-machine modules:

a processor operable to execute gesture-processing logic stored in one or more tangible media, the gesture-processing logic, when executed by the processor,



● Samsung K3UH5H5-OMMAGCJ 32 Gb (4 GB) LPDDR4X DRAM, layered over a Qualcomm Snapdragon 845

stored in one or more tangible media

a multi-touch gesture

Swipe down from the top of the screen with two fingers: Open your Quick Settings even faster. There's Airplane Mode! You turned it on just in the nick of time.

<https://r2.community.samsung.com/t5/Tech-Talk/Guide-to-Touchscreen-Gestures/td-p/4095444>

[1e] “when executed by the processor, configured to analyze the time series of data to distinguish one or more gesture inputs from the time series of data, the gesture-processing logic being coded with gesture-recognition code comprising a plurality of state-machine modules, the plurality of state-machine modules comprising:”

41. The Accused Products have a logic “when executed by the processor, configured to analyze the time series of data to distinguish one or more gesture inputs from the time series of data, the gesture-processing logic being coded with gesture-recognition code comprising a plurality of state-machine modules:

configured to **analyze the time series of data** to distinguish one or more gesture inputs from the time series of data, the gesture-processing logic being coded with gesture-recognition code comprising a plurality of state-machine modules, the plurality of state-machine modules comprising:

The Samsung touchscreen controller (TSC) driver runs on the Snapdragon845 processor which is the receiver of **the time series of data** for each individual touch detected by the TSC.

a time series of data (reported via mt_slot event)

analyze the time series of data (mt_slot events provided)

```

36 Drivers for type B devices separate contact packets by calling
37 input_mt_slot(), with a slot as argument, at the beginning of each packet.
38 This generates an ABS_MT_SLOT event, which instructs the receiver to
39 prepare for updates of the given slot.
40
41 All drivers mark the end of a multi-touch transfer by calling the usual
42 input_sync() function. This instructs the receiver to act upon events
43 accumulated since last EV_SYN/SYN_REPORT and prepare to receive a new set
44 of events/packets.
45
46 The main difference between the stateless type A protocol and the stateful
47 type B slot protocol lies in the usage of identifiable contacts to reduce
48 the amount of data sent to userspace. The slot protocol requires the use of
49 the ABS_MT_TRACKING_ID, either provided by the hardware or computed from
50 the raw data [5].
51
52 For type A devices, the kernel driver should generate an arbitrary
53 enumeration of the full set of anonymous contacts currently on the
54 surface. The order in which the packets appear in the event stream is not
55 important. Event filtering and finger tracking is left to user space [3].
56
57 For type B devices, the kernel driver should associate a slot with each
58 identified contact, and use that slot to propagate changes for the contact.
59 Creation, replacement and destruction of contacts is achieved by modifying
60 the ABS_MT_TRACKING_ID of the associated slot. A non-negative tracking id
61 is interpreted as a contact, and the value -1 denotes an unused slot. A
62 tracking id not previously present is considered new, and a tracking id no
63 longer present is considered removed. Since only changes are propagated,
64 the full state of each initiated contact has to reside in the receiving
65 end. Upon receiving an MT event, one simply updates the appropriate
66 attribute of the current slot.

```

Detect common gestures

distinguish gesture inputs ☆☆☆☆

A "touch gesture" occurs when a user places one or more fingers on the touch screen, and your application interprets that pattern of touches as a particular gesture. There are correspondingly two phases to gesture detection:

1. Gather data about touch events.
2. Interpret the data to see if it meets the criteria for any of the gestures your app supports.

analyze

Android runs on the Snapdragon845 processor which provides a convention for **distinguishing** single and multi-touch **gestures**. The Snapdragon845 processor **analyses the time series of data** received for each individual touch detected to **distinguish gestures** (i.e. spreading and pinching) made on the touchscreen.

configured to analyze the time series of data to distinguish one or more gesture inputs from the time series of data, the gesture-processing logic being coded with gesture-recognition code comprising a plurality of state-machine modules, the plurality of state-machine modules comprising:

```

1  /* drivers/input/touchscreen/sec_ts.h
2
3  * Copyright (C) 2015 Samsung Electronics Co., Ltd.
4  * http://www.samsungsemi.com/
5
6  * Core file for Samsung TSC driver
7
8  * This program is free software; you can redistribute it and/or modify
9  * it under the terms of the GNU General Public License version 2 as
10 * published by the Free Software Foundation.
11
12 */
13
14 #define SEC_TS_STATUS_EVENT 0
15 #define SEC_TS_COORDINATE_ACTION_NONE 1
16 #define SEC_TS_COORDINATE_ACTION_PRESS 2
17 #define SEC_TS_COORDINATE_ACTION_MOVE 3
18 #define SEC_TS_COORDINATE_ACTION_RELEASE 4
19 #define SEC_TS_EVENT_BUFF_SIZE 8
20
21 #define SEC_TS_COORDINATE_ACTION_NONE 0
22 #define SEC_TS_COORDINATE_ACTION_PRESS 1
23 #define SEC_TS_COORDINATE_ACTION_MOVE 2
24 #define SEC_TS_COORDINATE_ACTION_RELEASE 3
25
26 #define SEC_TS_TOUCHTYPE_NORMAL 0
27 #define SEC_TS_TOUCHTYPE_HOVER 1
28 #define SEC_TS_TOUCHTYPE_FLIPCOVER 2
29 #define SEC_TS_TOUCHTYPE_GLOVE 3
30 #define SEC_TS_TOUCHTYPE_STYLUS 4
31 #define SEC_TS_TOUCHTYPE_PALM 5
32 #define SEC_TS_TOUCHTYPE_WET 6
33 #define SEC_TS_TOUCHTYPE_PROXIMITY 7
34 #define SEC_TS_TOUCHTYPE_JIG 8
35
36 #define SEC_TS_GESTURE_TYPE/
37 #define SEC_TS_GESTURE_CODE_SFAY 0x00
38 #define SEC_TS_GESTURE_CODE_DOUBLE_TAP 0x01
39 #define SEC_TS_GESTURE_CODE_FORCE 0x02
40
41 #define SEC_TS_GESTURE_ID/
42 #define SEC_TS_EVENT_PRESSURE_TOUCHED 0x00
43 #define SEC_TS_EVENT_PRESSURE_RELEASED 0x01

```

single-touch state machine module

gesture-recognition code

The Type B Multi-Touch protocol used by Samsung is a stateful protocol whereby individual touch events are identified and assigned a specific state (i.e. Press and Idle). Therefore, there are a plurality of single touch state machine modules for the plurality of individual touches detected as part of a multi-touch gesture.

MotionEvent

```

public final class MotionEvent
    extends InputEvent implements Parcelable
{
    java.lang.Object
    < android.view.InputEvent
    < android.view.MotionEvent

```

Motion events describe movements in terms of an action code and a set of axis values. The action code specifies the state change that occurred such as a pointer going down or up. The axis values describe the position and other movement properties.

gesture-recognition code

Track multiple pointers

When multiple pointers touch the screen at the same time, the system generates the following touch events:

- **ACTION_DOWN** – For the first pointer that touches the screen. This starts the gesture. The pointer data for this pointer is always at index 0 in the `MotionEvent`.
- **ACTION_POINTER_DOWN** – For extra pointers that enter the screen beyond the first. The pointer data for this pointer is at the index returned by `getActionIndex()`.
- **ACTION_MOVE** – A change has happened during a press gesture.
- **ACTION_POINTER_UP** – Sent when a non-primary pointer goes up.
- **ACTION_UP** – Sent when the last pointer leaves the screen.

You keep track of individual pointers within a `MotionEvent` via each pointer's index and ID.

- **Index:** A `MotionEvent` effectively stores information about each pointer in an array. The index of a pointer is its position within this array. Most of the `MotionEvent` methods you use to interact with pointers take the pointer index as a parameter, not the pointer ID.
- **ID:** Each pointer also has an ID mapping that stays persistent across touch events to allow tracking an individual pointer across the entire gesture.

The order in which individual pointers appear within a motion event is undefined. Thus the index of a pointer can change from one event to the next, but the pointer ID of a pointer is guaranteed to remain constant as long as the pointer remains active. Use the `getPointerId()` method to obtain a pointer's ID to track the pointer across all subsequent

gesture-recognition code

```

// Determine focal point
float sumX = 0, sumY = 0;
final int div = pointerCount - 1;
final float focusX;
final float focusY;
if (isAnchoredScaleMode()) {
    // In anchored scale mode, the focal pt is always where the double tap
    // or button down gesture started
    focusX = mAnchoredScaleStartX;
    focusY = mAnchoredScaleStartY;
    if (event.getXY() < focusY) {
        mEventBeforeOrAboveStartingGestureEvent = true;
    } else {
        mEventBeforeOrAboveStartingGestureEvent = false;
    }
} else {
    for (int i = 0; i < count; i++) {
        if (skipIndex == i) continue;
        sumX += event.getX(i);
        sumY += event.getY(i);
    }
    focusX = sumX / div;
    focusY = sumY / div;
}

// Determine average deviation from focal point
float devSumX = 0, devSumY = 0;
for (int i = 0; i < count; i++) {
    if (skipIndex == i) continue;
    // Convert the resulting diameter into a radius.
    devSumX += Math.abs(event.getX(i) - focusX);
    devSumY += Math.abs(event.getY(i) - focusY);
}
final float devX = devSumX / div;
final float devY = devSumY / div;

// Span is the average distance between touch points through the focal point.
// i.e. the diameter of the circle with a radius of the average deviation from
// the focal point.
final float spanX = devX * 2;
final float spanY = devY * 2;
final float span;
if (isAnchoredScaleMode()) {
    span = spanY;
} else {
    span = (float) Math.hypot(spanX, spanY);
}

```

multi-touch state machine module

Multi-Touch Idle

Multi-Touch FSM
(ScaleGesture
Detector)

Android provides the ScaleGestureDetector multi-touch state machine module to detect multi-touch gestures such as pinch and stretch based on the provided MotionEvent (i.e. the outputs from the plurality of single touch state machines)

[1f] “a first one-touch state-machine module, the first one-touch state-machine module being operable to recognize at least a first one-touch gesture and generate a first output based on the first one-touch gesture;”

42. The Accused Products have a first one-touch state-machine module, the first one-touch state-machine module being operable to recognize at least a first one-touch gesture and generate a first output based on the first one-touch gesture:

a first one-touch state-machine module, the first one-touch state-machine module being operable to recognize at least a first one-touch gesture and generate a first output based on the first one-touch gesture;

MotionEvent

```
public final class MotionEvent
extends InputEvent implements Parcelable
```

recognize a first one-touch gesture

Motion events describe movements in terms of an action code and a set of axis values. The action code specifies the state change that occurs, such as a pointer going down or up. The axis values describe the position and other movement properties.

For example, when the user first touches the screen, the system delivers a touch event to the appropriate `View` with the action code `ACTION_DOWN` and a set of axis values that include the X and Y coordinates of the touch and information about the pressure, size and orientation of the contact area.

Some devices can report multiple movement traces at the same time. Multi-touch screens emit one movement trace for each finger. The individual fingers or other objects that generate movement traces are referred to as *pointers*. Motion events contain information about all of the pointers that are currently active even if some of them have not moved since the last event was delivered.

Device Types

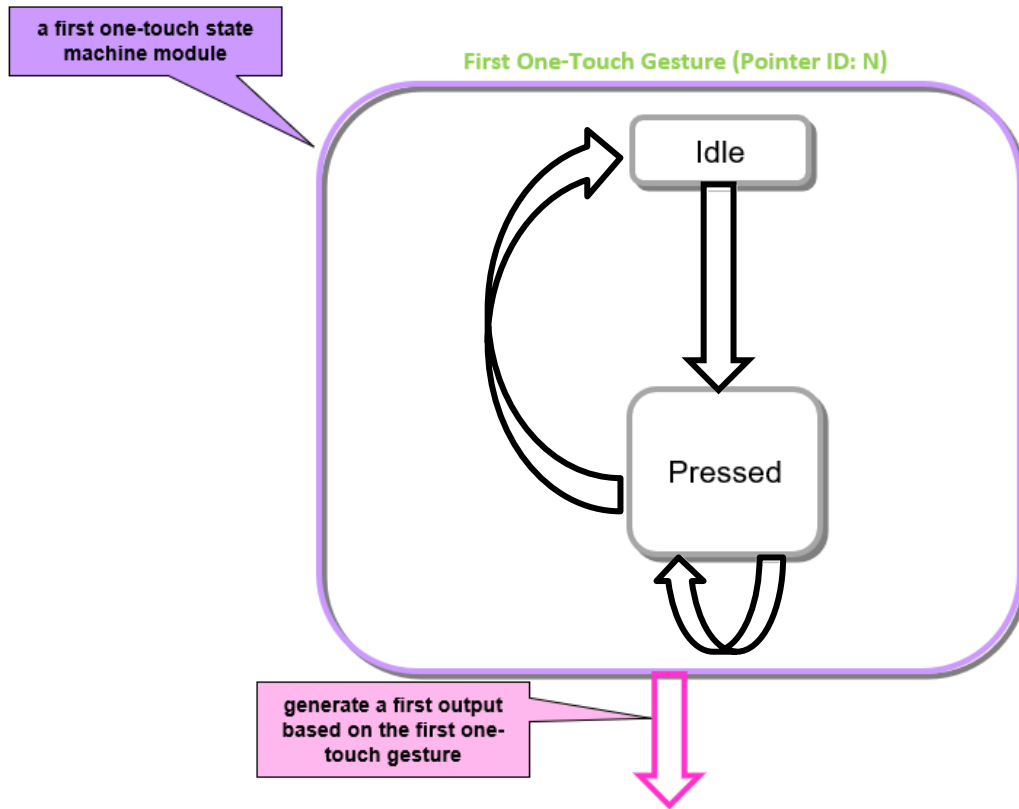
state machine module

The interpretation of the contents of a `MotionEvent` varies significantly depending on the source class of the device.

On pointing devices with source class `InputDevice.SOURCE_CLASS_POINTER` such as touch screens, the pointer coordinates specify absolute positions such as view X/Y coordinates. Each complete gesture is represented by a sequence of motion events with actions that describe pointer state transitions and movements. A gesture starts with a motion event with `ACTION_DOWN` that provides the location of the first pointer down. As each additional pointer that goes down or up, the framework will generate a motion event with `ACTION_POINTER_DOWN` or `ACTION_POINTER_UP` accordingly. Pointer movements are described by motion events with `ACTION_MOVE`. Finally, a gesture ends either when the final pointer goes up as represented by a motion event with `ACTION_UP` or when gesture is canceled with `ACTION_CANCEL`.

generate a first output based on the first one-touch gesture

recognize a first one-touch gesture



[1g] “a second one-touch state-machine module, the second one-touch state-machine module being operable to recognize at least a second one-touch gesture and generate a second output based on the second one-touch gesture; and”

43. The Accused Products have a second one-touch state-machine module, the second one-touch state-machine module being operable to recognize at least a second one-touch gesture and generate a second output based on the second one-touch gesture:

a second one-touch state-machine module, the second one-touch state-machine module being operable to recognize at least a second one-touch gesture and generate a second output based on the second one-touch gesture; and

MotionEvent

```
public final class MotionEvent
extends InputEvent implements Parcelable

java.lang.Object
├── android.view.InputEvent
└── android.view.MotionEvent
```

Motion events describe movements in terms of an action code and a set of axis values. The action code specifies the state change that occurred such as a pointer going down or up. The axis values describe the position and other movement properties.

For example, when the user first touches the screen, the system delivers a touch event to the appropriate `View` with the action code `ACTION_DOWN` and a set of axis values that include the X and Y coordinates of the touch and information about the pressure, size and orientation of the contact area.

Some devices can report multiple movement traces at the same time. Multi-touch screens emit one movement trace for each finger. The individual fingers or other objects that generate movement traces are referred to as *pointers*. Motion events contain information about all of the pointers that are currently active even if some of them have not moved since the last event was delivered.

Device Types

The interpretation of the contents of a `MotionEvent` varies significantly depending on the source class of the device.

On pointing devices with source class `InputDevice.SOURCE_CLASS_POINTER` such as touch screens, the pointer coordinates specify absolute positions such as view X/Y coordinates. Each complete gesture is represented by a sequence of motion events with actions that describe pointer state transitions and movements. A gesture starts with a motion event with `ACTION_DOWN` that provides the location of the first pointer down. As each additional pointer that goes down or up, the framework will generate a motion event with `ACTION_POINTER_DOWN` or `ACTION_POINTER_UP` accordingly. Pointer movements are described by motion events with `ACTION_MOVE`. Finally, a gesture ends either when the final pointer goes up as represented by a motion event with `ACTION_UP` or when gesture is canceled with `ACTION_CANCEL`.

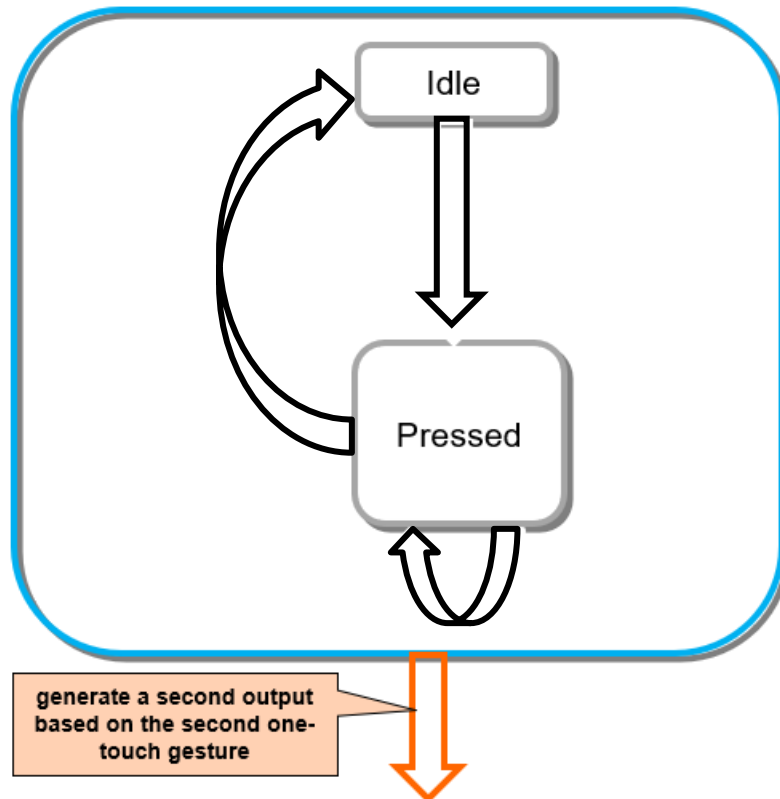
generate a second output based on the second one-touch gesture

recognize a second one-touch gesture

state machine module

a second one-touch state machine module

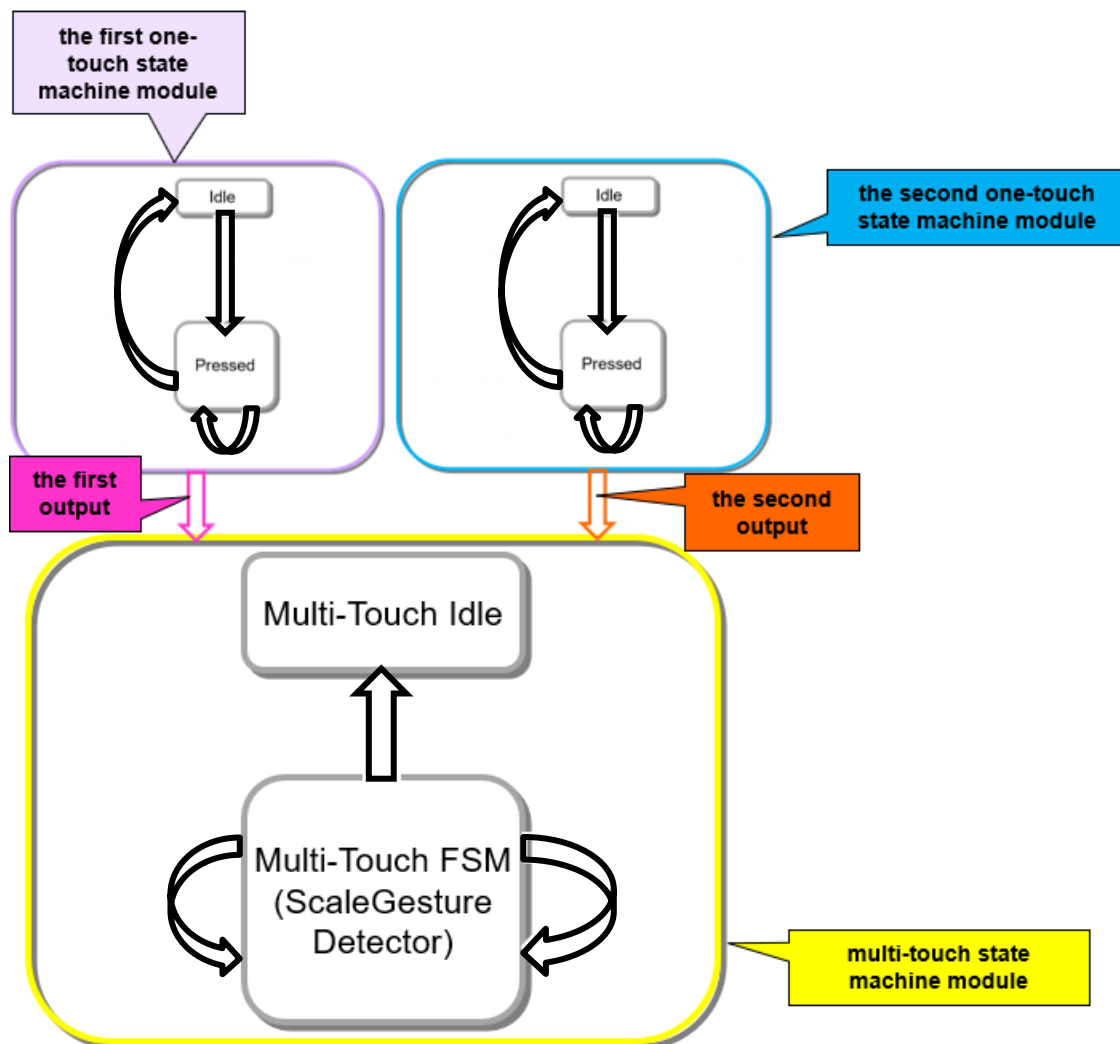
Second One-Touch Gesture (Pointer ID: N+1)



[1h] “a multi-touch state-machine module operable to: receive, directly from the first one-touch state-machine module, the first output; receive, directly from the second one-touch state-machine module, the second output; and”

44. The Accused Products have a multi-touch state-machine module operable to: receive, directly from the first one-touch state-machine module, the first output; receive, directly from the second one-touch state-machine module, the second output;

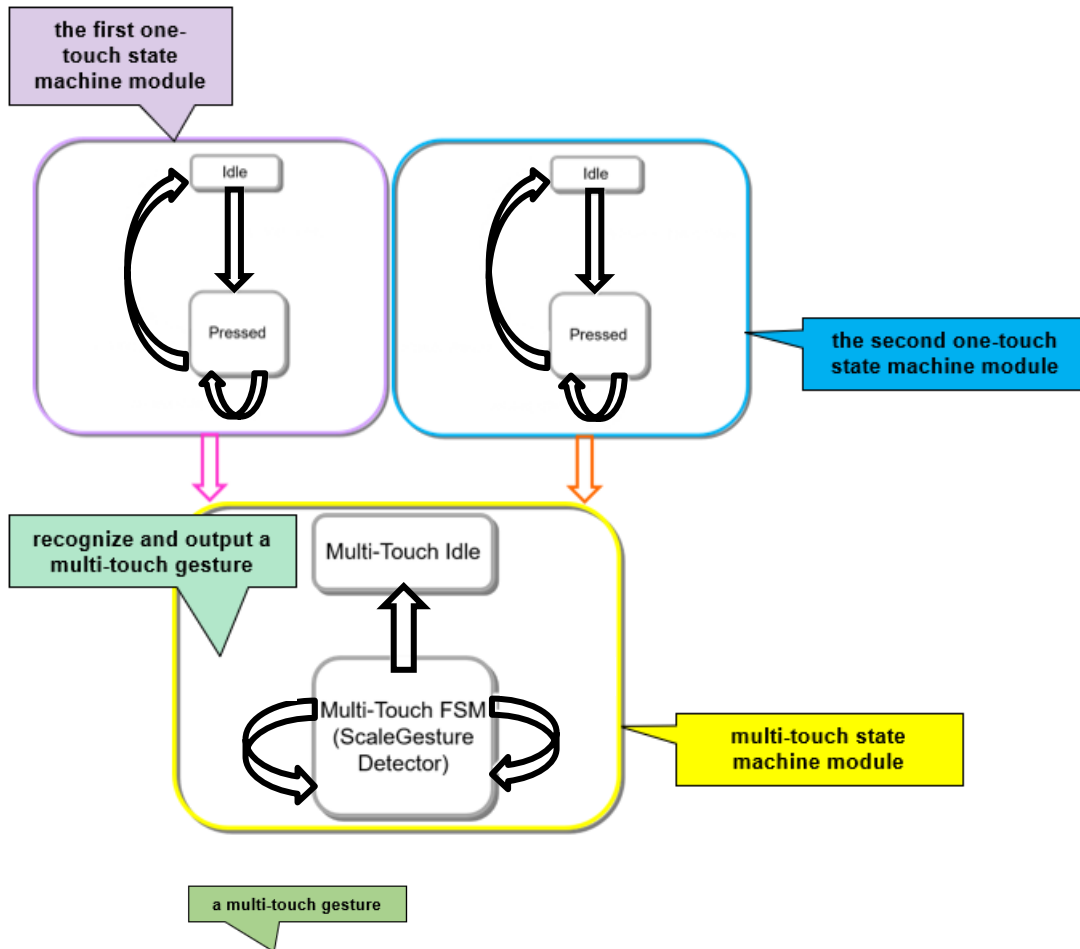
a multi-touch state-machine module operable to:
 receive, directly from the first one-touch state-machine module, the first output;
 receive, directly from the second one-touch state-machine module, the second output; and



[1i] “recognize, based on at least the first and second outputs, at least one multi-touch gesture, the first one-touch state-machine module, the second one-touch state-machine module, and the multi-touch state-machine module being distinct state-machine modules; and output the recognized multi-touch gesture.”

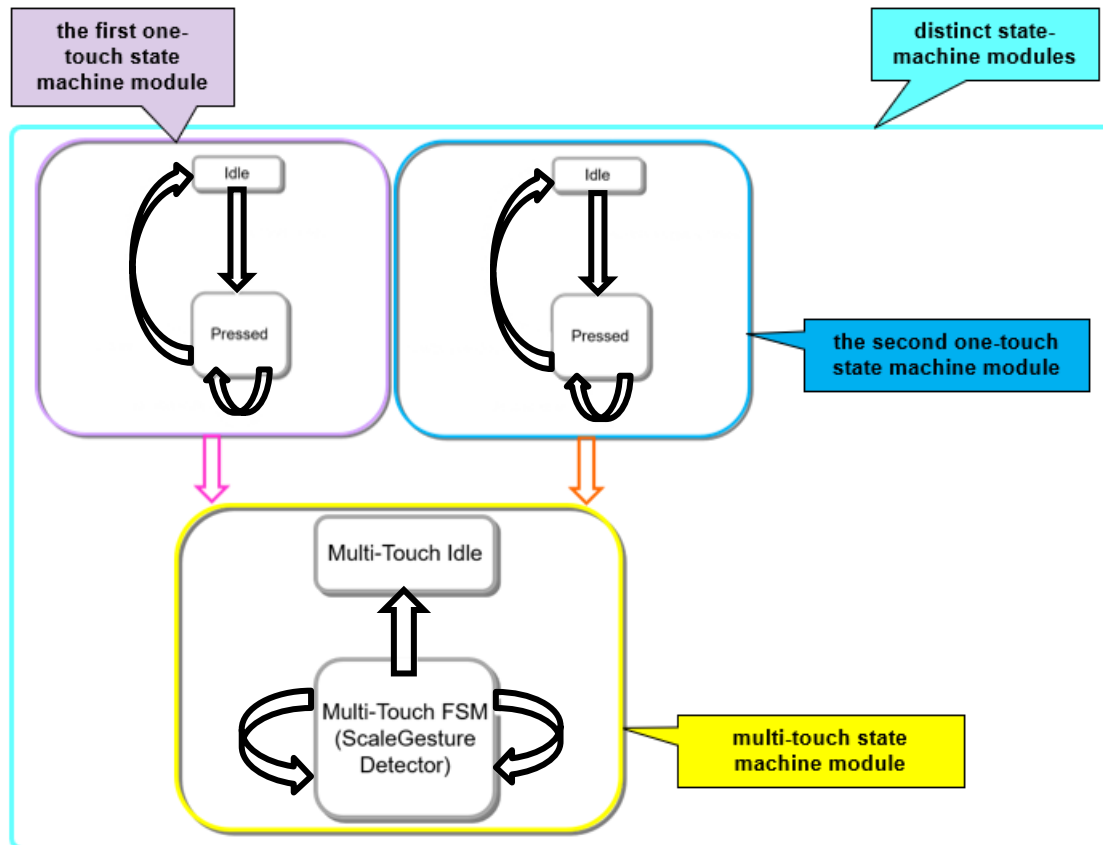
45. The Accused Products recognize, based on at least the first and second outputs, at least one multi-touch gesture, the first one-touch state-machine module, the second one-touch state-machine module, and the multi-touch state-machine module being distinct state-machine modules; and output the recognized multi-touch gesture:

recognize, based on at least the first and second outputs, at least one multi-touch gesture, the first one-touch state-machine module, the second one-touch state-machine module, and the multi-touch state-machine module being distinct state-machine modules; and output the recognized multi-touch gesture.



Swipe down from the top of the screen with two fingers: Open your Quick Settings even faster. There's Airplane Mode! You turned it on just in the nick of time.

<https://r2.community.samsung.com/t5/Tech-Talk/Guide-to-Touchscreen-Gestures/td-p/4095444>



46. Defendants also knowingly and intentionally induce and contribute to infringement of the '767 patent in violation of 35 U.S.C. §§ 271(b) and 271(c). Through the filing and service of this Complaint, Defendants have had knowledge of the '767 patent and the infringing nature of the Accused Products. Defendant SEC also has had knowledge of the '767 patent through the issuance of U.S. Patent No. 9,207,792 on December. 8, 2015 and assignment to SEC, which cites on its face the '767 patent. Despite this knowledge of the '767 patent, Defendants continue to actively encourage and instruct its customers to use and integrate the accused products in ways that directly infringe the '767 patent. Defendants do so knowing and intending that their customers will commit these infringing acts. Defendants also continue to make, use, offer for sale, sell, and/or import the Accused Products, despite their knowledge of the '767 patent, thereby specifically intending for and inducing its customers to infringe the '767

patent through the customers' normal and customary use of the Accused Products.

47. Defendants have infringed multiple claims of the '767 patent, including independent claim 1. By way of example only, the accused Samsung Galaxy S9 phone infringes an exemplary claim of the '767 patent, as in the description set forth above, which Solas provides without the benefit of information about the Accused Products obtained through discovery.

48. Defendants have known how the Accused Products are made and have known, or have been willfully blind to the fact, that making, using, offering to sell, and selling the Accused Products to their customers, would constitute willful infringement of the '767 patent. Those products imported into and sold within the United States include, without limitation, Samsung laptop computers, Galaxy tablets and phones.

49. Defendants have induced, and continue to induce, infringement of the '767 patent by actively encouraging others (including its customers) to use, offer to sell, sell, and import the Accused Products. On information and belief, these acts include providing information and instructions on the use of the Accused Products; providing information, education and instructions to its customers; providing the Accused Products to customers; and indemnifying patent infringement within the United States.

50. Solas has been damaged by Defendant's infringement of the '767 patent and is entitled to damages as provided for in 35 U.S.C. § 284, including reasonable royalty damages.

Jury demand.

51. Solas demands trial by jury of all issues.

Relief requested.

Solas prays for the following relief:

A. A judgment in favor of Solas that Defendants have infringed the '144 patent and the '767 patent, and that the '144 patent and the '767 patent are valid, enforceable, and patent-eligible;

B. A judgment and order requiring Defendants to pay Solas compensatory damages, costs, expenses, and pre- and post-judgment interest for its infringement of the asserted patents, as provided under 35 U.S.C. § 284;

C. A permanent injunction prohibiting Defendants from further acts of infringement of the '144 patent and the '767 patent;

D. A judgment and order requiring Defendants to provide an accounting and to pay supplemental damages to Solas, including, without limitation, pre-judgment and post-judgment interest;

E. A finding that this case is exceptional under 35 U.S.C. § 285, and an award of Solas' reasonable attorney's fees and costs; and

F. Any and all other relief to which Solas may be entitled.

Dated: March 22, 2021

Respectfully submitted,

/s/ Reza Mirzaie

Reza Mirzaie

CA State Bar No. 246953

Email: rmirzaie@raklaw.com

Marc Fenster

CA State Bar No. 181067

Email: mfenster@raklaw.com

Neil A. Rubin

CA State Bar No. 250761

Email: nrubin@raklaw.com

James S. Tsuei

CA State Bar No. 285530

Email: jtsuei@raklaw.com

RUSS AUGUST & KABAT

12424 Wilshire Blvd. 12th Floor

Los Angeles, CA 90025

Telephone: 310-826-7474

T. John Ward, Jr.
Texas State Bar No. 00794818
Email: jw@wsfirm.com
Claire Abernathy Henry
Texas State Bar No. 24053063
Email: claire@wsfirm.com
Andrea L. Fair
Texas State Bar No. 24078488
Email: andrea@wsfirm.com
WARD, SMITH & HILL, PLLC
PO Box 1231
Longview, Texas 75606-1231
(903) 757-6400 (telephone)
(903) 757-2323 (facsimile)

**ATTORNEYS FOR PLAINTIFF,
SOLAS OLED, LTD.**