

**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-JRG

**First Amended Complaint for Patent  
Infringement**

**JURY DEMANDED**

**First Amended Complaint for Patent Infringement**

Plaintiff Solas OLED Ltd. (“Solas”) files this first amended 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 No. 8,526,767 (“Patent-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 its customers and integrated into electronic devices sold in the United States.

**Plaintiff Solas OLED and the Patent-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. 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 1.

### **Defendants and the Accused Products.**

3. 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.

4. 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.

5. 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.**

6. 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).

7. 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 the Patent-in-Suit 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 Patent-in-Suit.

8. 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 '767 patent.**

9. On September 3, 2013, the United States Patent and Trademark Office issued U.S. Patent No. 8,526,767, entitled “Gesture Recognition.” Ex. 1.

10. 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.

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

12. 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.

13. 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.

14. 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*

15. 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;”**

16. 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**

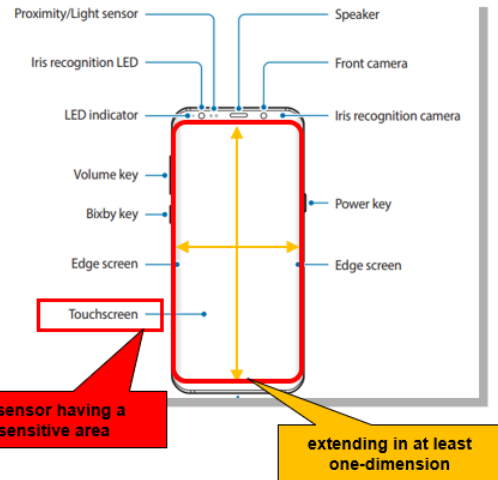
- Do not allow the touchscreen to come into contact with other electrical devices. Electrostatic discharges can cause the touchscreen to malfunction.
- To avoid damaging the touchscreen, do not tap it with anything sharp or apply excessive pressure to it with your fingertips.
- Leaving the touchscreen 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.



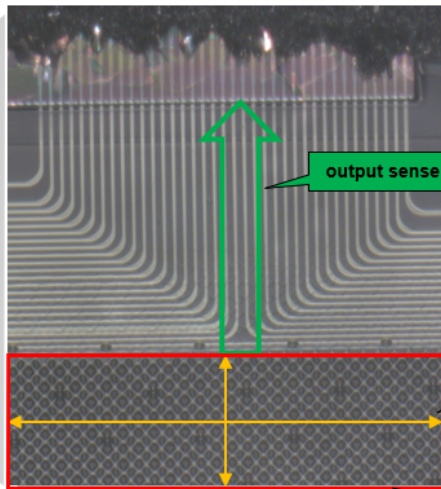
sensitive area

output sense signals responsive to proximity of an object



a sensor having a sensitive area

extending in at least one-dimension



output sense signals

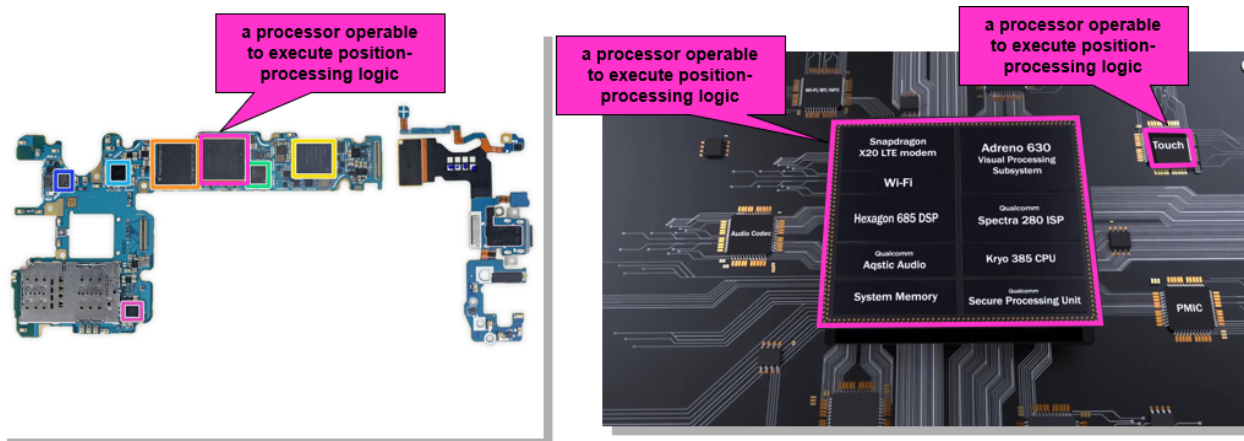
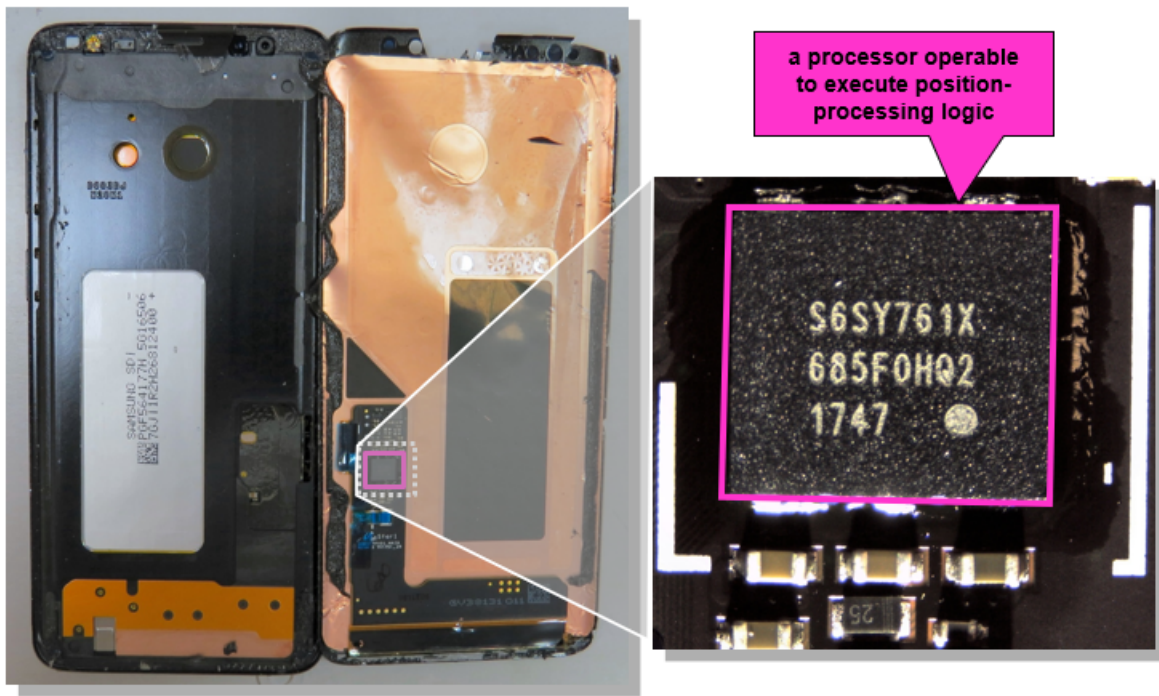
extending in at least one-dimension

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:”

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



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

```

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/ctype.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 cmd.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”

18. 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  };
    
```

```

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

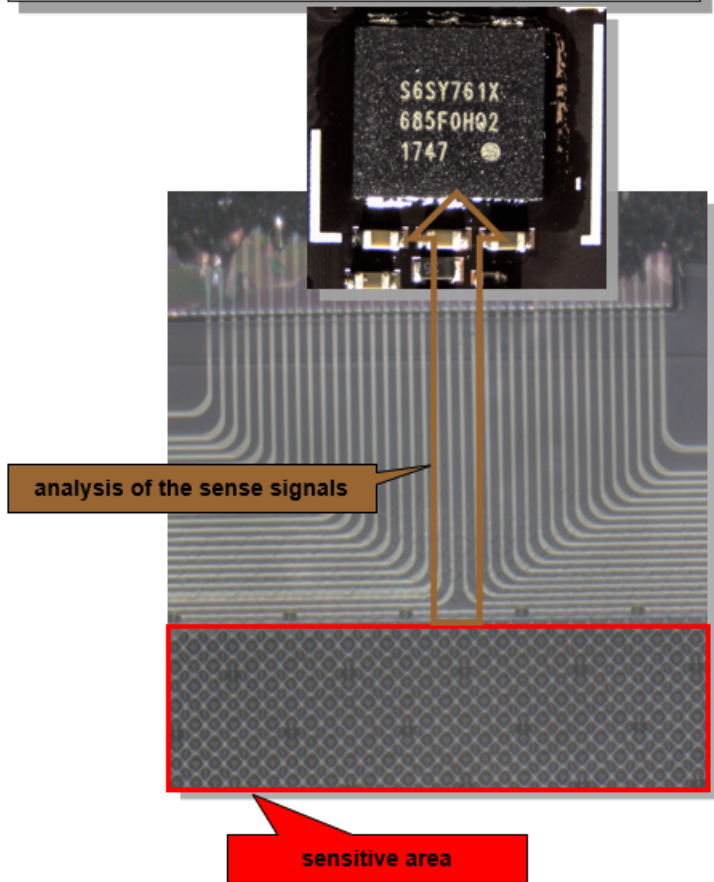
calculate positions of interactions

sensitive area

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 corresponding 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

```

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, sensor 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
    
```

**touches** (points to line 11)

**Interaction positions** (points to line 11)

**sensor** (points to line 15)

**a time series of data** (points to line 19)

```

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 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_REPORT and prepare to receive a new set
44 of events/packets.
45
46 The main difference a time series of data (reported via mt_slot event) 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.
    
```

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
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

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.

corresponding to touches

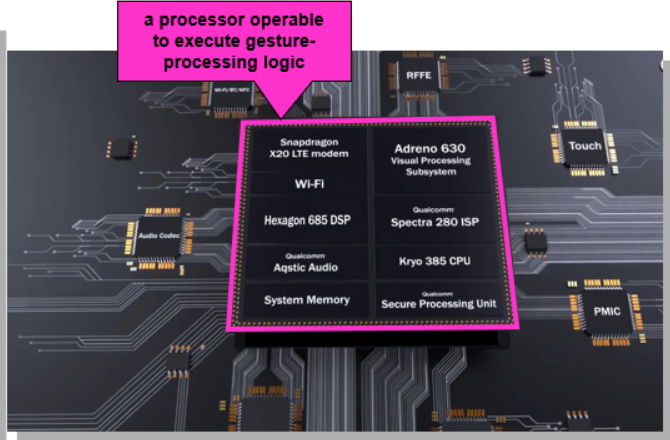
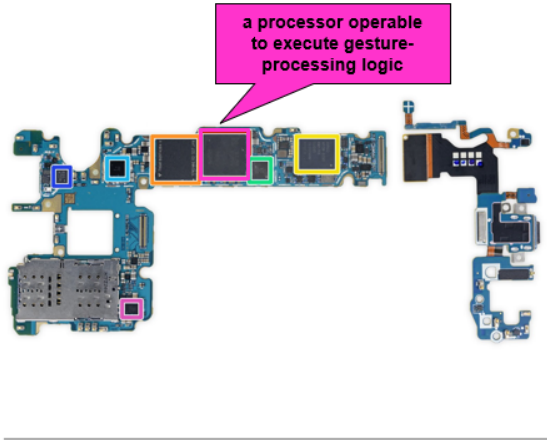
a time series of data

interaction positions

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

19. 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:”

20. 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 /* SEC status event id */
14 #define SEC_TS_COORDINATE_EVENT 0
15 #define SEC_TS_STATUS_EVENT 1
16 #define SEC_TS_GESTURE_EVENT 2
17 #define SEC_TS_EMPTY_EVENT 3
18
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 /* 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 /* SEC_TS_GESTURE_ID */
42 #define SEC_TS_EVENT_PRESSURE_TOUCHED 0x00
43 #define SEC_TS_EVENT_PRESSURE_RELEASED 0x01
44
45

```

single-touch states

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
}

```

gesture-recognition code

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.

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

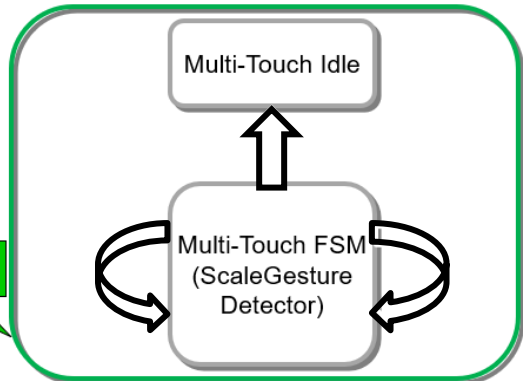
gesture-recognition code

```

// Determine focal point
float sumX = 0, sumY = 0;
final int div = pointerUp ? count - 1 : count;
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;
} else {
    if (event.getTV() < focusY) {
        mEventBeforeOrAboveStartingGestureEvent = true;
    } else {
        mEventBeforeOrAboveStartingGestureEvent = false;
    }
}
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



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;”

21. 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.

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

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.

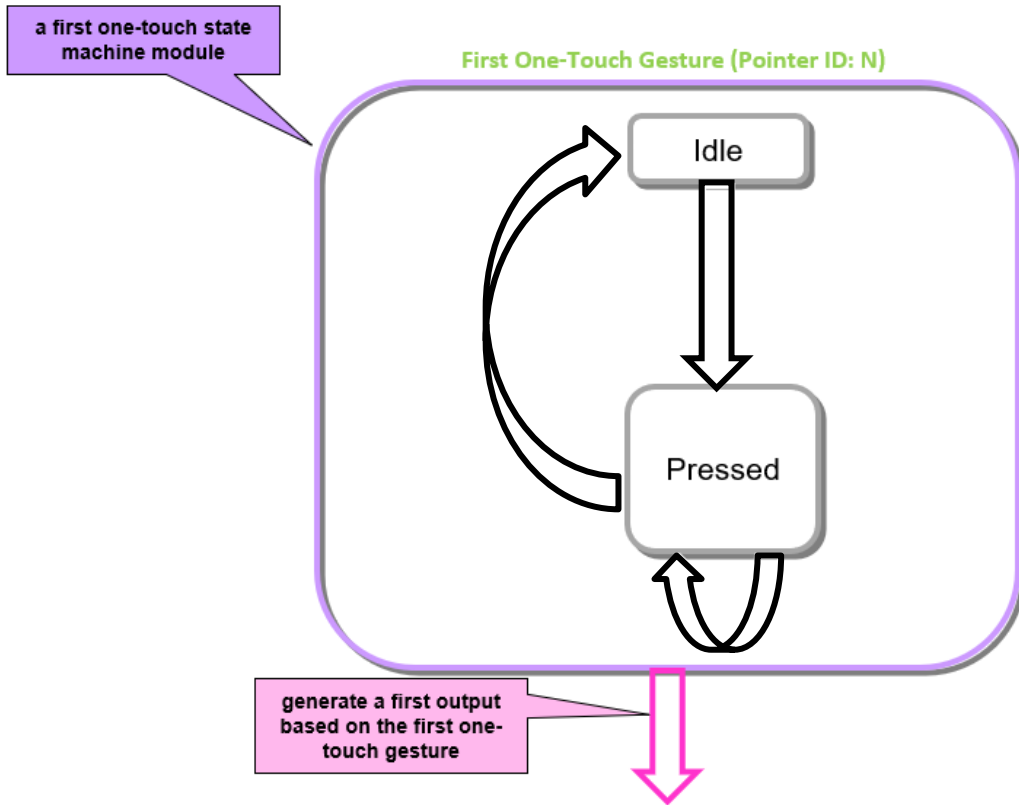
**state machine module**

**recognize a first one-touch gesture**

**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`.



[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”

22. 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`.

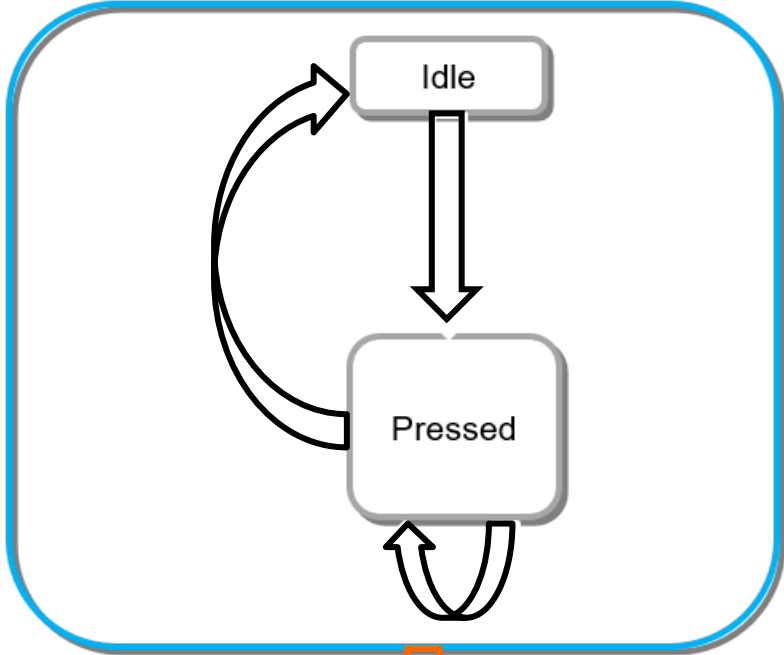
generate a second output based on the second one-touch gesture

state machine module

recognize a second one-touch gesture

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

a second one-touch state machine module

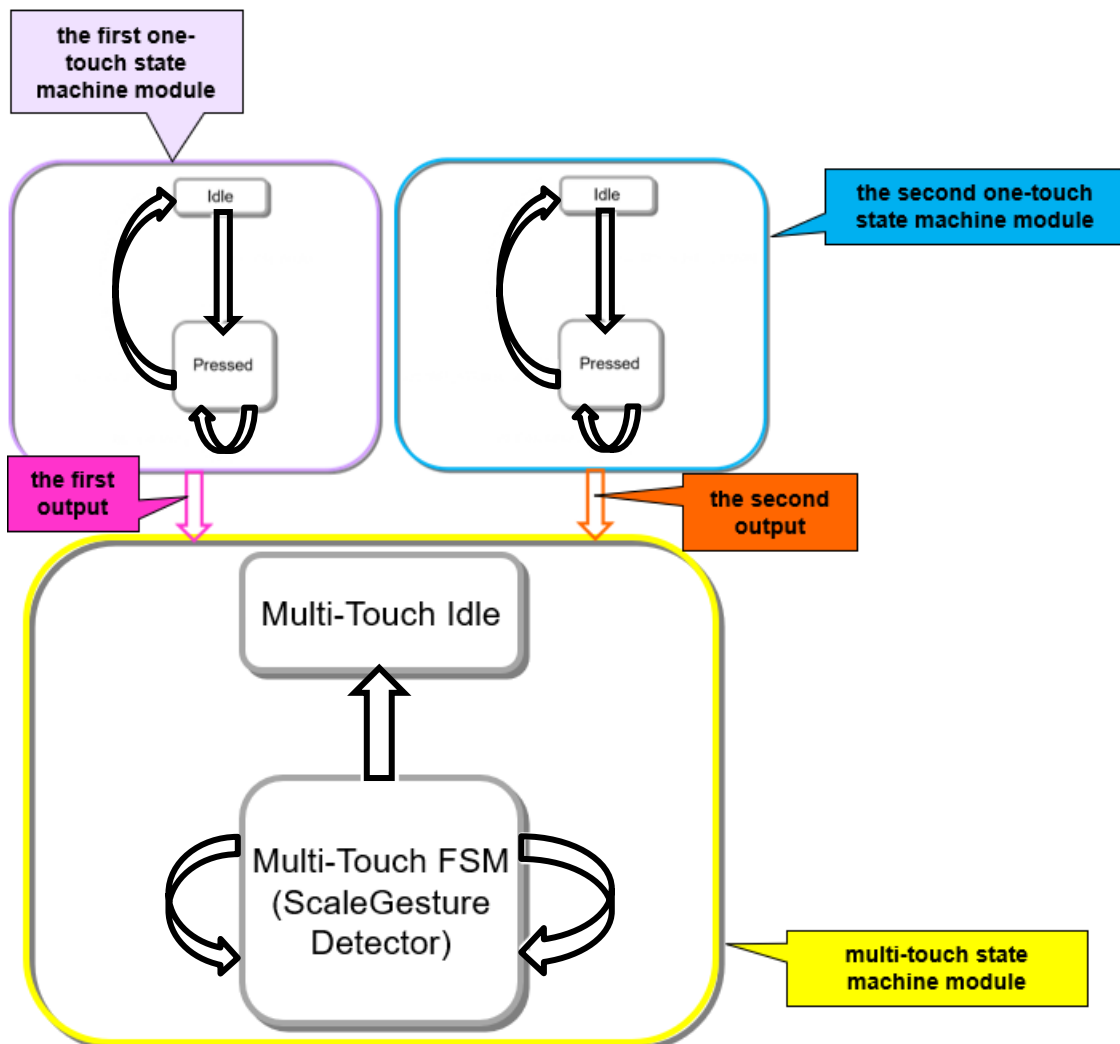


generate a second output based on the second one-touch gesture

[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”

23. 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:

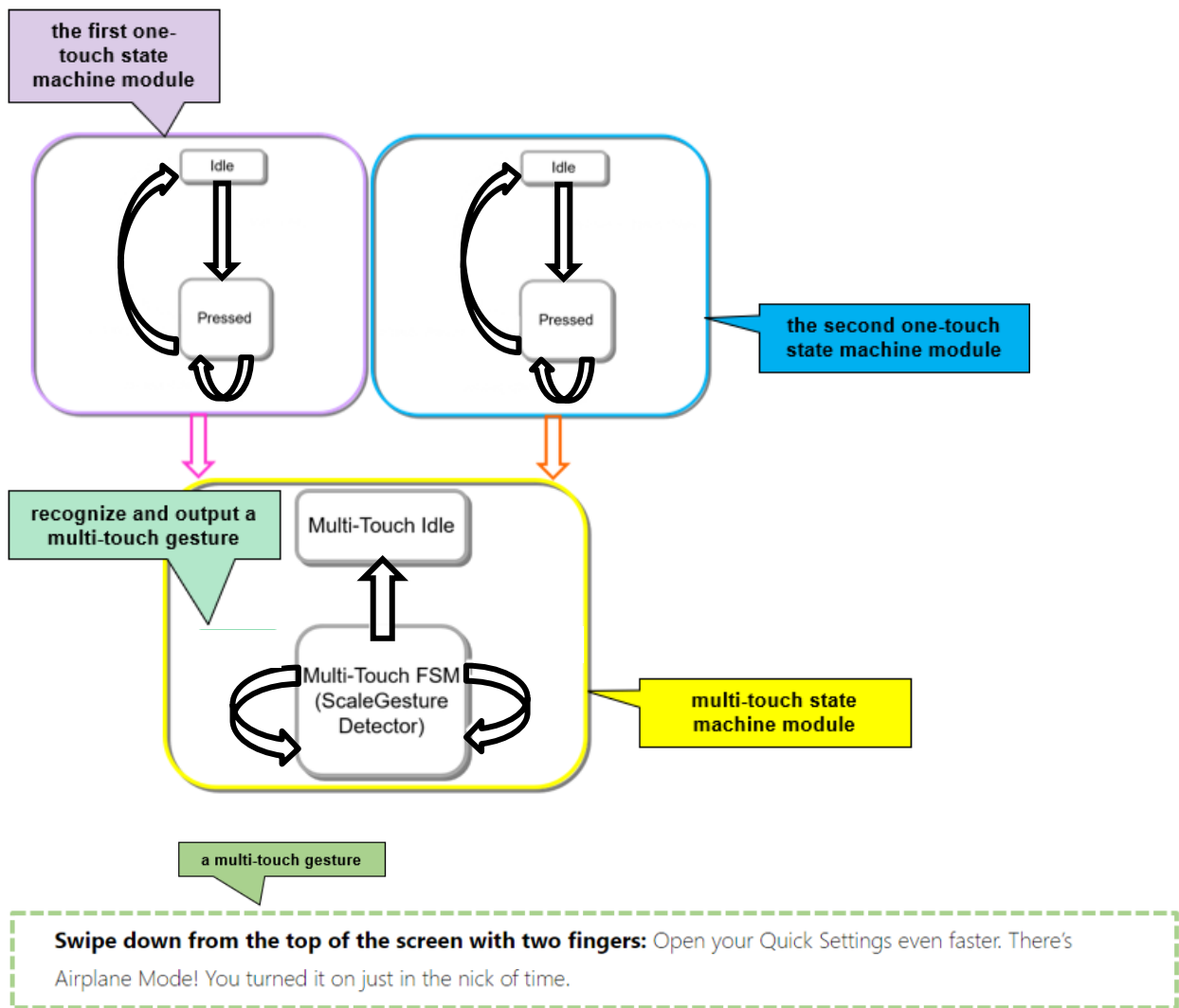
a multi-touch state-machine module operable to:  
receive, directly from the first one-touch state-machine module, the first output;  
receive, directly from the second one-touch state-machine module, the second output; and



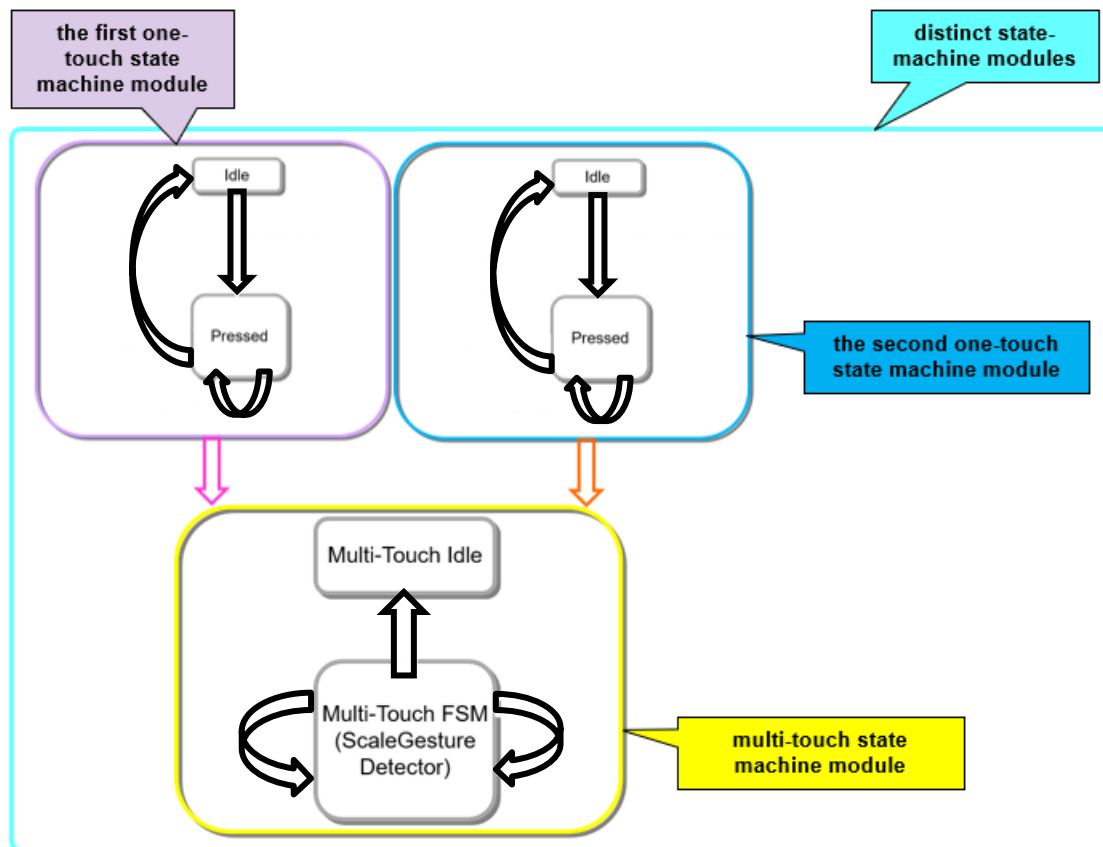
[1i] “recognize, based on at least the first and second outputs, at least one multi-touch gesture, the first one-touch state-machine module, the second one-touch state-machine module, and the multi-touch state-machine module being distinct state-machine modules; and output the recognized multi-touch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

24. 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.



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



25. 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.

26. 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.

27. 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.

28. 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.

29. 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.**

30. 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 '767 patent, and that the '767 patent is 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 Patent-in-Suit, as provided under 35 U.S.C. § 284;

C. A permanent injunction prohibiting Defendants from further acts of infringement of 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: April 16, 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](mailto:jw@wsfirm.com)  
Claire Abernathy Henry  
Texas State Bar No. 24053063  
Email: [claire@wsfirm.com](mailto:claire@wsfirm.com)  
Andrea L. Fair  
Texas State Bar No. 24078488  
Email: [andrea@wsfirm.com](mailto: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.**